

Enterprise Message API C++ Edition 3.6.2.G2

ENTERPRISE MESSAGE API CONFIGURATION GUIDE

Document Version: 3.6.2.G2
Date of issue: August 2021
Document ID: EMAC362CG.210



© **Refinitiv 2015 - 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	Definitions	1
1.4	Acronyms and Abbreviations	2
1.5	References	3
1.6	Document Conventions	3
1.6.1	<i>Typographic</i>	3
1.6.2	<i>Data Types</i>	3
1.6.3	<i>Field and Text Values</i>	4
1.6.4	<i>Boolean Values</i>	4
2	Configuration Overview	5
2.1	About Message API Configuration	5
2.2	Parameter Overview	5
2.3	Default Behaviors	6
3	Configuration Groups	7
3.1	ConsumerGroup	7
3.1.1	<i>Generic XML Schema for ConsumerGroup</i>	7
3.1.2	<i>Setting a Default Consumer</i>	7
3.1.3	<i>Configuring Consumers in a ConsumerGroup</i>	8
3.1.4	<i>Consumer Entry Parameters</i>	8
3.2	Provider Groups	14
3.2.1	<i>Generic XML Schema for Provider Group</i>	14
3.2.2	<i>Setting a Default Provider</i>	14
3.2.3	<i>Configuring a Provider in a ProviderGroup</i>	15
3.2.4	<i>Provider Entry Parameters</i>	15
3.3	Channel Group	21
3.3.1	<i>Generic XML Schema for ChannelGroup</i>	21
3.3.2	<i>Universal Channel Entry Parameters</i>	22
3.3.3	<i>Parameters for Use with Channel Type: RSSL_SOCKET</i>	24
3.3.4	<i>Parameters for Use with Channel Types: RSSL_HTTP</i>	25
3.3.5	<i>Parameters for Use with Channel Types: RSSL_WEBSOCKET</i>	26
3.3.6	<i>Parameters for Use with Channel Types: RSSL_ENCRYPTED</i>	26
3.3.7	<i>Parameters for Use with Channel Type: RSSL_RELIABLE_MCAST</i>	28
3.3.8	<i>Example XML Schema for Configuring ChannelSet</i>	30
3.3.9	<i>Example Programmatic Configuration for ChannelSet</i>	30
3.4	Server Group	32
3.4.1	<i>Generic XML Schema for ServerGroup</i>	32
3.4.2	<i>Server Entry Parameters</i>	32
3.4.3	<i>Parameters for Use with ServerType RSSL_ENCRYPTED</i>	34
3.4.4	<i>Parameters for Use with ServerType RSSL_WEBSOCKET</i>	35
3.5	Logger Group	36
3.5.1	<i>Generic XML Schema for LoggerGroup</i>	36
3.5.2	<i>Logger Entry Parameters</i>	36
3.6	Dictionary Group	38
3.6.1	<i>Generic XML Schema for DictionaryGroup</i>	38
3.6.2	<i>Dictionary Entry Parameters</i>	38
3.7	Directory Group	39
3.7.1	<i>Generic XML Schema for Directory Entry</i>	39

3.7.2	<i>Setting Default Directory.....</i>	39
3.7.3	<i>Configuring a Directory in a DirectoryGroup.....</i>	40
3.7.4	<i>Service Entry Parameters.....</i>	40
3.7.5	<i>InfoFilter Entry Parameters.....</i>	40
3.7.6	<i>StateFilter Entry Parameters</i>	44
3.7.7	<i>LoadFilter Entry Parameters.....</i>	45
3.7.8	<i>Status Entry Parameters.....</i>	45
4	Enterprise Message API Configuration Processing	46
4.1	Overview and Configuration Precedence.....	46
4.2	Default Configuration	46
4.2.1	<i>Default Consumer Configuration</i>	46
4.2.2	<i>Default Provider Configurations.....</i>	47
4.3	Processing the Enterprise Message API XML Configuration File	48
4.3.1	<i>Reading the Configuration File</i>	48
4.3.2	<i>Use of the Correct Order in the XML Schema</i>	50
4.3.3	<i>Processing the Consumer "Name"</i>	51
4.3.4	<i>Processing the Provider "Name"</i>	51
4.4	Configuring the Enterprise Message API Using Function Calls	51
4.4.1	<i>Configuration Function Calls.....</i>	51
4.4.2	<i>Using the host() Function: How "Host" and "Port" are Processed.....</i>	54
4.5	Programmatic Configuration	54
4.5.1	<i>OMM Data Structure.....</i>	54
4.5.2	<i>Creating a Programmatic Configuration for a Consumer.....</i>	55
4.5.3	<i>Example: Programmatic Configuration of a Consumer</i>	56
4.5.4	<i>Creating a Programmatic Configuration for a Provider.....</i>	58
4.5.5	<i>Example: Programmatic Configuration of a Provider.....</i>	59

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 C++ 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 C++ Edition API Developer's Guide*.

1.3 Definitions

DEFINITION	DESCRIPTION
Group	A related set of configuration parameters for a specific EMA component (e.g., ChannelGroup).
List	A list of components belonging to a group (e.g., ChannelList).
Component	A specific component (e.g., Channel). Because lists can have multiple components, each component must have a 'name' field for identification purposes.
Field	A configurable parameter.
Default Value	A default value is the value the API uses if a value is not specified by the user. In general, items with default values are required by the API.
Allowed value	Specific values or a range of values that the field allows.

Table 1: Definitions

1.4 Acronyms and Abbreviations

Generally, this guide avoids the use of acronyms. However, in diagrams, you might see one or more acronyms to conserve space. The following table provides a list that might be used in this guide's diagrams.

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)
JSON	JavaScript Object Notation
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 2: Acronyms and Abbreviations

1.5 References

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

1.6 Document Conventions

This document uses the following types of conventions:

- Typographic
- Data Types
- Field and Text Values

1.6.1 Typographic

This document uses the following types of conventions:

- C++ 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;
    OmmConsumer consumer( OmmConsumerConfig().operationModel(
OmmConsumerConfig::UserDispatchEnum ).host( "localhost:14002" ).username( "user" ) );
    consumer.registerClient( ReqMsg().domainType( MMT_MARKET_BY_PRICE ).serviceName(
"DIRECT_FEED" ).name( "BBH.ITC" ).privateStream( true ), client );
    unsigned long long startTime = getCurrentTime();
```

1.6.2 Data Types

Data types within the configuration repository are as follows:

DATA TYPE	DEFINITION
Double	Stores decimal numbers with double precision.
EmaString	String

Table 3: Data Type Conventions

DATA TYPE	DEFINITION
Enumeration	Specific text, as indicated in the field description
Int64	Signed, long integer
UInt64	Unsigned, long integer

Table 3: Data Type Conventions (Continued)

1.6.3 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., `RSSL_SOCKET`), in the software, text values are represented as numbers (required for programmatic configuration). When introduced, enumerations are listed along with their textual values.

1.6.4 Boolean Values

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

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

2 Configuration 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, logger, 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), logger, and directory groups.
- **Channel:** Channel configuration data describe various connection alternatives and provide configuration alternatives for those connections.
- **Logger:** Logger configuration data specify logging alternatives and associated parameters.
- **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 example, according to the Enterprise Message API's global default behavior:

- The Enterprise Message API's logs its messages at a **LoggerSeverity** level of **Success** to a file named **emaLog_pid.log** (where *pid* is the process ID). You can manually change the **LoggerSeverity** and the log filename by using **EmaConfig.xml**.
- The Enterprise Message API does not XML trace to file (equivalent to **XmlTraceToFile value="0"**). You need to add this parameter only if you want to turn on XML tracing. If you turn on XML tracing (a non-default behavior), the Enterprise Message API will trace to a file named **EmaTrace** (equivalent to **XmlTraceFileName value="EmaTrace"**).

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	EmaString	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	EmaString	14002 (for consumers) 14003 (for non-interactive providers)	Specifies the port number on the server to which the application connects.
DefaultConsumer	EmaString	EmaConsumer	If consumer components are configured, the Enterprise Message API ignores this parameter.
DefaultIProvider	EmaString	EmaIProvider	If interactive provider components are configured, the Enterprise Message API ignores this parameter.
DefaultNiProvider	EmaString	EmaNiProvider	If non-interactive provider components are configured, the Enterprise Message API ignores this parameter.
LoggerSeverity	Enumeration	Success	Sets the level at which the Enterprise Message API logs events. For details on logging severity levels and their enumerations, refer to Section 3.5.2.
LoggerType	Enumeration	File	Specifies the destination for output messages. The parameter value can be either File or Stdout . For details on selecting a loggerType and its enumerations, refer to Section 3.5.2.
FileName	EmaString	"emaLog_ <i>pid</i> .log"	Specifies the base name of log file (used when LoggerType value="File"); the Enterprise Message API automatically appends _pid.log to the base name, where pid is the logger's process id number.
RdmFieldDictionaryFileName	EmaString	./RDMFieldDictionary	Specifies the path and name of the RdmFieldDictionary file.
EnumTypeDefFileName	EmaString	./enumtype.def	Specifies the path and name of the enumtypeDef dictionary file.

Table 4: 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**, **Logger**, 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 logger schemas, you specify the desired logger schema by adding the following parameter inside the appropriate **Consumer** section:

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

Consumer components can use different logger 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
CatchUnhandledException	UInt64	1	Specifies whether the Enterprise Message API catches unhandled exceptions thrown from methods executed on the Enterprise Message API's thread or whether the Enterprise Message API lets the application handle them. Available values include: <ul style="list-style-type: none"> 0 (false): the Enterprise Message API passes unhandled exceptions to the operating system. 1 (true): Whenever the Enterprise Message API catches unhandled exceptions in its thread, the Enterprise Message API logs an error message and then terminates the thread.
CatchUnknownJsonFids	UInt64	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	UInt64	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	EmaString	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.

Table 5: Consumer Group Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ChannelSet	EmaString	N/A	<p>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 3.3.8.</p> <p>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>.</p>
CloseChannelFromConverterFailure	UInt64	1	<p>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:</p> <ul style="list-style-type: none"> • 0 (false): Do not close the channel. • 1 (true): Close the channel.
DefaultServiceID	UInt64	1	<p>Specifies a default service ID for RWF/JSON conversion if both service name and ID are missing. The maximum allowable value is 65535.</p>
Dictionary	EmaString	N/A	<p>Specifies how the consumer should access its dictionaries (it must match the Name parameter from the appropriate <Dictionary> entry in the DictionaryGroup configuration).</p> <p>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 3.6.</p>
DictionaryRequestTimeout	UInt64	45,000	<p>Specifies the amount of time (in milliseconds) the application has to download dictionaries from a provider before the OmmConsumer throws an exception.</p> <p>If set to 0, the Enterprise Message API will wait for a response indefinitely.</p> <p>NOTE: If ChannelSet is configured:</p> <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	UInt64	45,000	<p>Specifies the amount of time (in milliseconds) the provider has to respond with a source directory refresh message before the OmmConsumer throws an exception.</p> <p>If set to 0, the Enterprise Message API will wait for a response indefinitely.</p> <p>NOTE: If ChannelSet is configured:</p> <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.

Table 5: Consumer Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
DispatchTimeoutApiThread	Int64	-1	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 less than zero, the Enterprise Message API internal thread goes active only if it gets notified about a received message.
EnableRtt	UInt64	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	UInt64	100,000	Specifies the number of items the application expects to request. If set to 0 , the Enterprise Message API resets it to 513 . For better performance, the application can set this to the approximate number of item requests it expects.
JsonExpandedEnumFields	UInt64	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.
Logger	EmaString	N/A	Specifies a set of logging behavior the Consumer should exhibit (it must match the Name parameter from the appropriate <Logger> entry in the LoggerGroup configuration). If Logger is not specified, the Enterprise Message API uses a set of logger default behaviors. For further details on the <Logger> entry and default settings, refer to Section 3.5.
LoginRequestTimeOut	UInt64	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	UInt64	100	Specifies the maximum number of messages the Enterprise Message API dispatches before taking a real-time break.
MaxDispatchCountUserThread	UInt64	100	Specifies the maximum number of messages the Enterprise Message API can dispatch in a single call to the OmmConsumer::dispatch() .
MaxOutstandingPosts	UInt64	100,000	Specifies the maximum allowable number of on-stream posts waiting for an acknowledgment before the OmmConsumer disconnects.
MaxEventsInPool	Int64	-1	Specifies the maximum number of event objects in the event's pool.

Table 5: Consumer Group Parameters (Continued)


PARAMETER	TYPE	DEFAULT	DESCRIPTION
MsgKeyInUpdates	UInt64	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	EmaString	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	UInt64	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)
OutputBufferSize	UInt64	65535	Sets the size of the output buffer for the RWF/JSON conversion. <div>  WARNING! If the buffer size is not large enough, the RWF/JSON conversion fails. </div>
PipePort	Int64	9001	Specifies the internal communication port. You might need to adjust this port if it conflicts with other processes on the machine.
PostAckTimeout	UInt64	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	Int64	-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.
ReconnectMaxDelay	Int64	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	Int64	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	Int64	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	Int64	-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	UInt64	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.

Table 5: Consumer Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
RestEnableLog	UInt64	0	Enables Rest request/response logging. Available values include: <ul style="list-style-type: none"> • 0 (false) • Any value > 0 (true) You can specify a log destination in RestLogFileName .
RestLogFileName	EmaString	N/A	Allow redirecting REST logs (enabled by restEnableLog) to some specified file or stream. If this value is not set, the log is sent to standard output (stdout).
RestRequestTimeout	UInt64	90	Specifies the timeout (in seconds) 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	UInt64	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 .
XmlTraceDump	UInt64	0	Sets the Enterprise Message API to trace dump RWF messages after converting them from JSON messages. Possible values are: <ul style="list-style-type: none"> • 0 (false): Do not trace dump data. • 1 (true): Trace dump data.
XmlTraceFileName	EmaString	EmaTrace	Sets the name of the file to which to write XML trace output if tracing is selected.
XmlTraceHex	UInt64	0	Sets whether to print incoming and outgoing messages in hexadecimal format. Possible values are: <ul style="list-style-type: none"> • 0 (false): Do not print messages in hexadecimal format. • 1 (true): Print messages in hexadecimal format.
XmlTraceMaxFileSize	UInt64	100000000	Specifies the maximum size (in bytes) for the trace file.
XmlTracePing	UInt64	0	Sets the Enterprise Message API to trace incoming and outgoing ping messages. Possible values are: <ul style="list-style-type: none"> • 0 (false): Do not trace ping messages. • 1 (true): Trace ping messages.
XmlTraceRead	UInt64	1	Sets the Enterprise Message API to trace incoming data. Possible values are: <ul style="list-style-type: none"> • 0 (false): Do not trace incoming data. • 1 (true): Trace incoming data
XmlTraceToFile	UInt64	0	Sets whether the Enterprise Message API traces its messages to an XML file whose name is set by XmlTraceFileName . Available values are: <ul style="list-style-type: none"> • 0 (false): Turns off tracing. • 1 (true): Turns on tracing to an XML file.

Table 5: Consumer Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
XmlTraceToMultipleFiles	UInt64	0	Specifies whether to write the XML trace to multiple files. Possible values are: <ul style="list-style-type: none"> 1 (true): the Enterprise Message API writes the XML trace to a new file if the current file size reaches the XmlTraceMaxFileSize. 0 (false): the Enterprise Message API stops writing the XML trace if the current file reaches the XmlTraceMaxFileSize.
XmlTraceToStdout	UInt64	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.
XmlTraceWrite	UInt64	1	Sets the Enterprise Message API to trace outgoing data. Possible values are: <ul style="list-style-type: none"> 0 (false): Do not trace outgoing data. 1 (true): Trace outgoing data.

Table 5: 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), **Logger**, 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 provider 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 logger schemas, you specify the desired logger schema by adding the following parameter inside the appropriate **Provider** section:

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

If your provider component needs more than one logger 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	UInt64	0	Used only with IPProvider . Sets the IPProvider to accept incoming directory request messages without the minimum required INFO and STATE directory filters.
AcceptMessageSameKeyButDiffStream	UInt64	0	Used only with IPProvider . Sets the IPProvider 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	UInt64	0	Used only with IPProvider . Sets the IPProvider to accept incoming request messages for reissuing the service name of an existing item stream.
AcceptMessageWithoutAcceptingRequests	UInt64	0	Used only with IPProvider . Sets the IPProvider to accept incoming request messages even though the source directory is not accepting requests.
AcceptMessageWithoutBeingLogin	UInt64	0	Used only with IPProvider . Sets the IPProvider to accept incoming request messages even though the interactive provider has not accepted a login request.
AcceptMessageWithoutQosInRange	UInt64	0	Used only with IPProvider . Sets the IPProvider to accept incoming request messages even though the requesting QoS is not in the QoS range of the source directory.

Table 6: *ProviderGroup* Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
CatchUnhandledException	UInt64	1	Specifies whether the Enterprise Message API catches unhandled exceptions thrown from methods executed on the its thread or whether the Enterprise Message API lets the application handle them. Available values include: <ul style="list-style-type: none"> 1 (true): Whenever the Enterprise Message API catches unhandled exceptions in its thread, the Enterprise Message API logs an error message and then terminates the thread. 0 (false): the Enterprise Message API passes unhandled exceptions to the operating system.
CatchUnknownJsonFids	UInt64	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	UInt64	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	EmaString	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	EmaString	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 3.3.8. 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	UInt64	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	UInt64	1	Specifies a default service ID for RWF/JSON conversion if both service name and ID are missing. The maximum allowable value is 65535 .

Table 6: *ProviderGroup* Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
Directory	EmaString	N/A	Specifies source directory refresh information that the Provider sends after establishing a connection (this must match the Name parameter from the appropriate <Directory> entry in the DirectoryGroup configuration). If Directory is not specified, the Enterprise Message API uses a hard coded configuration. For further details on the <Logger> entry and default settings, refer to Section 3.7.
DispatchTimeoutApiThread	Int64	-1	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 less than zero, the thread goes active only if notified about a received message.
EnforceAckIDValidation	UInt64	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.
EnumTypeFragmentSize	UInt64	12288	Used only with IProvider . Sets the maximum enumeration types fragmentation size (in bytes) for each multi-part refresh message.
FieldDictionaryFragmentSize	UInt64	8192	Used only with IProvider . Sets the maximum field dictionary fragmentation size (in bytes) for each multi-part refresh message.
ItemCountHint	UInt64	100,000	Specifies the number of items the application expects to maintain. If set to 0, the Enterprise Message API resets it to 513. For better performance, the application can set this to the approximate number of items it maintains.
JsonExpandedEnumFields	UInt64	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.
Logger	EmaString	N/A	Specifies a set of logging behavior the Provider should exhibit (it must match the Name parameter from the appropriate <Logger> entry in the LoggerGroup configuration). If Logger is not specified, the Enterprise Message API uses a set of logger default behaviors. For further details on the <Logger> entry and default settings, refer to Section 3.5.

Table 6: *ProviderGroup* Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
LoginRequestTimeout	UInt64	45,000	<p>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.</p> <p>If set to 0, the Enterprise Message API will wait for a response indefinitely.</p> <p>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.</p>
MaxDispatchCountApiThread	UInt64	100	Specifies the maximum number of messages the Enterprise Message API dispatches before taking a real-time break.
MaxDispatchCountUserThread	UInt64	100	Specifies the maximum number of messages the Enterprise Message API can dispatch in a single call to the OmmProvider::dispatch() .
MaxEventsInPool	Int64	-1	Specifies the maximum number of event objects in the event's pool.
MergeSourceDirectoryStreams	UInt64	1	<p>Used only with NiProvider. Specifies whether the Enterprise Message API merges all source directory streams (configured and user-submitted) into one stream:</p> <ul style="list-style-type: none"> • 1 (true) • 0 (false)
Name	EmaString	N/A	Specifies the name of this Provider component. Name is required when creating a Provider component. You can use any value for Name .
OutputBufferSize	UInt64	65535	<p>Sets the size of the output buffer for the RWF/JSON conversion.</p> <p>WARNING! If the buffer size is not large enough, the RWF/JSON conversion fails.</p>
PipePort	Int64	See Description	<p>Specifies the internal communication port. You might need to adjust this port if it conflicts with other processes on the machine.</p> <ul style="list-style-type: none"> • NiProvider uses a default of 9001. • IProvider uses a default of 9002.
ReconnectAttemptLimit	Int64	-1	<p>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.</p> <p>If set to -1, the consumer and non-interactive provider continually attempt to reconnect.</p>
ReconnectMaxDelay	Int64	5000	<p>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.</p>

Table 6: *ProviderGroup* Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ReconnectMinDelay	Int64	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	UInt64	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	UInt64	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	UInt64	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	UInt64	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	EmaString	N/A	Used only with IProvider . Specifies the channel that the IProvider 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.
ServiceCountHint	UInt64	513	Sets the size of directory structures for managing services. If the application specifies 0 , the Enterprise Message API resets it to 513 .
XmlTraceDump	UInt64	0	Sets the Enterprise Message API to trace dump RWF messages after converting them from JSON messages. Possible values are: <ul style="list-style-type: none"> • 0 (false): Do not trace dump data. • 1 (true): Trace dump data.
XmlTraceFileName	EmaString	EmaTrace	Sets the name of the file to which to write XML trace output if tracing is selected.

Table 6: *ProviderGroup* Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
XmlTraceHex	UInt64	0	Sets whether to print incoming and outgoing messages in hexadecimal format. Possible values are: <ul style="list-style-type: none"> • 0 (false): Do not print messages in hexadecimal format. • 1 (true): Print messages in hexadecimal format.
XmlTraceMaxFileSize	UInt64	100000000	Specifies the maximum size (in bytes) for the trace file.
XmlTracePing	UInt64	0	Sets the Enterprise Message API to trace incoming and outgoing ping messages. Possible values are: <ul style="list-style-type: none"> • 0 (false): Do not trace ping messages. • 1 (true): Trace ping messages.
XmlTraceRead	UInt64	1	Sets the Enterprise Message API to trace incoming data. Possible values are: <ul style="list-style-type: none"> • 0 (false): Do not trace incoming data. • 1 (true): Trace incoming data
XmlTraceToFile	UInt64	0	Sets whether the Enterprise Message API traces its messages to an XML file whose name is set by XmlTraceFileName . Available values are: <ul style="list-style-type: none"> • 0 (false): Turns off tracing. • 1 (true): Turns on tracing to an XML file.
XmlTraceToMultipleFiles	UInt64	0	Specifies whether to write the XML trace to multiple files. Possible values are: <ul style="list-style-type: none"> • 1 (true): the Enterprise Message API writes the XML trace to a new file if the current file size reaches the XmlTraceMaxFileSize. • 0 (false): the Enterprise Message API stops writing the XML trace if the current file reaches the XmlTraceMaxFileSize.
XmlTraceToStdout	UInt64	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.
XmlTraceWrite	UInt64	1	Sets the Enterprise Message API to trace outgoing data. Possible values are: <ul style="list-style-type: none"> • 0 (false): Do not trace outgoing data. • 1 (true): Trace outgoing data.

Table 6: *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 3.3.6.
- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_HTTP**, refer to Section 3.3.6.
- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_RELIABLE_MCAST**, refer to Section 3.3.7.

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	Enumeration	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 enumeration values 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"> • RSSL_SOCKET (0) • RSSL_ENCRYPTED (1): Supported on Windows OS and Linux. • RSSL_HTTP (2): Supported only on Windows OS • RSSL_RELIABLE_MCAST (4) • RSSL_WEBSOCKET (7)
ConnectionPingTimeout	UInt64	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	UInt64	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	UInt64	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	UInt64	6144	Specifies the upper buffer-usage threshold for the channel.
InitializationTimeout	UInt64	5 (10 when used with RSSL_ENCRYPTED ChannelType)	Specifies the time (in seconds) to wait for the successful initialization of a channel.
InterfaceName	EmaString	""	<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 7: Universal <Channel> Parameters

PARAMETER NAME	TYPE	DEFAULT	NOTES
Location	EmaString	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	EmaString		Specifies the Channel 's name.
NumInputBuffers	UInt64	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	UInt64	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	UInt64	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 7: 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	UInt64	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	Enumeration	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. Use enumeration values 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 (0) • ZLib (1) • LZ4 (2) <p>NOTE: A server can be configured to force a particular compression type, regardless of client settings.</p>
Host	EmaString	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.
LibcurlName	EmaString	""	<p>Specifies the filename of the Libcurl library. If the file is not stored in the working directory, you must also specify the directory in this parameter (e.g., /usr/lib64/libcurl.so).</p> <p>By default:</p> <ul style="list-style-type: none"> • On Linux, the Enterprise Message API attempts to load libcurl.so (determined by your environment variables). • On Windows for release builds, the Enterprise Message API attempts to load libcurl.dll from a directory determined by your environment (e.g., such as the Path variable and others). • On Windows for debug builds, the Enterprise Message API attempts to load libcurl-d.dll from a directory determined by your environment (e.g., such as the Path variable and others).
Port	EmaString	14002	Specifies the port on the remote server to which the Enterprise Message API connects.
ProxyHost	EmaString	""	<p>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.</p> <p>Any value provided by a function call overrides the setting in configuration file.</p>
ProxyPort	EmaString	""	<p>Specifies the port on the proxy to which the Enterprise Message API connects.</p> <p>Any value provided by a function call overrides the setting in configuration file.</p>
TcpNodelay	UInt64	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 8: 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	UInt64	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	Enumeration	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 enumeration values 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 (0) • ZLib (1) • LZ4 (2)
			NOTE: A server can be configured to force a particular compression type, regardless of client settings.
Host	EmaString	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	EmaString	""	Specifies the object name to pass along with the underlying URL in HTTP and HTTPS connection messages.
Port	EmaString	14002	Specifies the port on the remote server to which the Enterprise Message API connects.
ProxyHost	EmaString	""	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	EmaString	""	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	UInt64	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 9: Parameters for Channel Type: RSSL_HTTP

3.3.5 Parameters for Use with Channel Types: RSSL_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 **RSSL_WEBSOCKET**. A **RSSL_WEBSOCKET** channel does not support the LZ4 compression type.

PARAMETER NAME	TYPE	DEFAULT	NOTES
WsMaxMsgSize	UInt64	61440	Specifies the maximum size of messages that the WebSocket transport can send or read.
WsProtocols	EmaString	tr_json2, rssl.rwf, rssl.json.v2	Specifies a list of supported/preferred protocols in order of preference from highest to lowest.

Table 10: Parameters for Channel Types: RSSL_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	Enumeration	On Windows: RSSL_HTTP On Linux: RSSL_SOCKET RSSL_WEBSOCKET	Specifies the type of protocol used for this encrypted connection. <ul style="list-style-type: none"> RSSL_SOCKET (0) RSSL_HTTP (2): Supported only on WinINet-based connections using the Windows OS (for backwards compatibility). RSSL_WEBSOCKET (7) <p>NOTE: Include the parameters for the chosen protocol type specific to that type. For example, if you choose RSSL_SOCKET, the Enterprise Message API also expects to use the parameters listed in Section 3.3.3.</p>
LibcryptoName	EmaString	""	Specifies the filename of the libcrypto library. If the file is not stored in the working directory, you must also specify the directory in this parameter (e.g., /usr/lib64/libcrypto.so). As a rule, the Enterprise Message API attempts to load libssl 1.1.x before attempting to load libssl 1.0.x. Because the Libssl library filename changes based on version and platform, the Enterprise Message API manifests the following defaults: <ul style="list-style-type: none"> On Linux, in version 1.1.x, the Enterprise Message API attempts to load libcrypto.so.11. On Linux, in version 1.0.x, the Enterprise Message API attempts to load libcrypto.so.10. On Windows, in version 1.1.x, the Enterprise Message API attempts to load libcrypto-1_1-x64.dll. On Windows, in version 1.0.x, the Enterprise Message API attempts to load libeay32.dll.

Table 11: Parameters for Channel Types: RSSL_ENCRYPTED

PARAMETER NAME	TYPE	DEFAULT	NOTES
LibsslName	EmaString	""	<p>Specifies the filename of the Libssl library. If the file is not stored in the working directory, you must also specify the directory in this parameter (e.g., <code>/usr/lib64/libssl.so</code>).</p> <p>As a rule, the Enterprise Message API attempts to load libssl 1.1.x before attempting to load libssl 1.0.x. Because the Libssl library filename changes based on version and platform, the Enterprise Message API manifests the following defaults:</p> <ul style="list-style-type: none"> On Linux, in version 1.1.x, the Enterprise Message API attempts to load libssl.so.11. On Linux, in version 1.0.x, the Enterprise Message API attempts to load libssl.so.10. On Windows, in version 1.1.x, the Enterprise Message API attempts to load libssl-1_1-x64.dll. On Windows, in version 1.0.x, the Enterprise Message API attempts to load ssleay32.dll.
OpenSSLCAStore	EmaString	""	<p>Specifies either:</p> <ul style="list-style-type: none"> The directory that contains the certificate authority store or, The certificate authority certificate (i.e., a file). If you specify a file, Refinitiv recommends that you include the directory; otherwise your system will use the environment variables to locate the file. <p>If openSSLCAStore is blank or <code>'\0'</code>, Enterprise Message API's default behavior is dependent on the Operating System as follows:</p> <ul style="list-style-type: none"> On Windows: the Enterprise Message API loads root certificates from Windows's root certificate store into OpenSSL. You can manage additional certificates using the Windows OS certificate manager. On Linux: the Enterprise Message API's behavior is handled entirely by OpenSSL as follows: <ul style="list-style-type: none"> If the OpenSSL library was provided by the OS vendor, refer to their documentation on where the certificate authority store is located. If the OpenSSL library was built from source, default behavior is defined by the build configuration.
SecurityProtocol	UInt64	7	<p>When using OpenSSL, SecurityProtocol specifies the combination of flag values that set the version(s) of the TLS encryption protocol used for this connection:</p> <p>Currently, the only valid TLS version is TLS 1.2 (0x4).</p>

Table 11: Parameters for Channel Types: RSSL_ENCRYPTED (Continued)

3.3.7 Parameters for Use with Channel Type: RSSL_RELIABLE_MCAST

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_RELIABLE_MCAST**.

Several of these parameters configure how the channel sends a Host Status Messages on the network, while others configure how the channel manages RRCP packet transmission. For further details on the Host Status Message (HSM) concept, on configuring HSMs, and on RRCP packet transmission, refer to the *ADS* or *AHD Software Installation Manuals*.

Additionally several parameters are designed for use with a Refinitiv Real-Time Distribution System infrastructure tool called **rrdump**. **rrdump** is a monitoring utility available in the Refinitiv Real-Time Distribution System Infrastructure Tools package. For more information on **rrdump**, refer to either of the *ADS* and *ADH Software Installation Manuals*.

PARAMETER NAME	TYPE	DEFAULT	NOTES
DisconnectOnGap	UInt64	0	Specifies whether the underlying connection should be closed if a multicast gap situation is detected. <ul style="list-style-type: none"> 0 (false): 0 is the default value which means the underlying connection is not closed if a multicast gap situation occurs. 1 (true): Sets the underlying connection to close if a multicast gap situation occurs.
HsmInterface	EmaString	""	Specifies the Host Status Message (HSM) interface. By default, HsmInterface is set to the host machine's default interface.
HsmInterval	UInt64		The interval (in seconds) over which HSM packets are sent. You can use rrdump to change the value of hsmInterval . Thus, after starting the application, you can stop and restart HSM publication as needed. The default interval is 0 (disabled) which suspends host status message publication.
HsmMultAddress	EmaString	""	Specifies the multicast address over which this channel sends HSM packets. Enterprise Message API configuration allows for the use of defined aliases.
HsmPort	EmaString	""	Specifies the multicast port to which this channel sends HSM packets.
ndata	UInt64	7	Specifies the maximum number of retransmissions to attempt for an unacknowledged point-to-point packet.
nmissing	UInt64	128	Specifies the maximum number of missed consecutive multicast packets, from a particular node, from which RRCP requests retransmits.
nrreq	UInt64	3	Specifies the maximum number of retransmit requests that can be sent for a missing packet.
PacketTTL	UInt64	5	Sets the lifespan (in hops) of the data packet through the multicast network, which can prevent the packet from circulating indefinitely. It has a range of 0 - 255 . <ul style="list-style-type: none"> 0 means the message can be sent only to other applications on the same machine. A value of 255 sets the message to travel through the network indefinitely.
pktPoolLimitHigh	UInt64	190000	Specifies the high-water mark for the RRCP packet pool. If this limit is reached, no further RRCP packets are allocated until usage falls below the low-water mark (as set by pktPoolLimitLow).

Table 12: Parameters for Channel Type: RSSL_RELIABLE_MCAST

PARAMETER NAME	TYPE	DEFAULT	NOTES
pktPoolLimitLow	UInt64	180000	Specifies the low-water mark for the RRCP packet pool. If RRCP packet allocation gets frozen (due to pktPoolLimitHigh having been reached), additional RRCP packets are allocated only when usage falls below the pktPoolLimitLow setting. pktPoolLimitLow should be greater than $3 * \text{userQLimit}$.
RecvAddress	EmaString	""	Specifies the multicast address to which this channel connects for receiving data.
RecvPort	EmaString	""	Specifies the multicast port to which this channel connects for receiving data.
SendAddress	EmaString	""	Specifies the multicast address to which this channel connects for sending data.
SendPort	EmaString	""	Specifies the multicast port to which this channel connects for sending data.
tbchold	UInt64	3	Specifies the maximum time that RRCP holds a transmitted broadcast packet in case the packet needs to be retransmitted. tbchold is specified in RRCP clock ticks (100 milliseconds), so a value of 2 means 200 milliseconds.
tcpControlPort	EmaString	""	Specifies the port to use for the RRCP tcpControlPort . This port is used when troubleshooting RRCP using the rrdump tool. A setting of -1 disables tcpControlPort .
tdata	UInt64	1	Specifies the time that RRCP waits before retransmitting an unacknowledged point-to-point data message. tdata is specified in RRCP clock ticks of 100 milliseconds, thus a value of 2 means 200 milliseconds.
tpphold	UInt64	3	Specifies the maximum time that RRCP holds a transmitted point-to-point packet in case the packet needs to be retransmitted. tpphold is specified in RRCP clock ticks (100 milliseconds), so a value of 2 means 200 milliseconds.
trreq	UInt64	4	Specifies the amount of time that RRCP waits before "resending" a retransmit request for a missed multicast packet. trreq is specified in RRCP clock ticks (100 milliseconds), so a value of 2 means 200 milliseconds.
twait	UInt64	3	Specifies the duration of time for which RRCP ignores additional retransmit requests for a data packet that it has already retransmitted. This time period starts with the receipt of the first request for retransmission. twait is specified in RRCP clock ticks (100 milliseconds), so a value of 2 means 200 milliseconds.
UnicastPort	EmaString	""	Port to which this connection connects for unicast messages (i.e., ack/nak messages and any retransmit messages). This value also configures a TCP listening port for use with the rrdump tool.
userQLimit	UInt64	65535	Specifies the maximum backlog of messages allowed on an application's inbound message queue. If userQLimit is exceeded, the RRCP protocol engine begins to discard messages for that application until the backlog decreases.

Table 12: Parameters for Channel Type: RSSL_RELIABLE_MCAST (Continued)

3.3.8 Example XML Schema for Configuring ChannelSet

The following is an example **ChannelSet** configuration within the XML schema:

```
<ConsumerGroup>
  <ConsumerList>
    <Consumer>
      <Name value="Consumer_1"/>
      <!-- ChannelSet is optional -->
      <ChannelSet value="Channel_1, Channel_2"/>
      <!-- Logger is optional: defaulted to "File + Success" -->
      <Logger value="Logger_1"/>
      <!-- Dictionary is optional: defaulted to "ChannelDictionary" -->
      <Dictionary value="Dictionary_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="122.1.1.100"/>
      <Port value="14008"/>
    </Channel>
  </ChannelList>
</ChannelGroup>
```

3.3.9 Example Programmatic Configuration for ChannelSet

The following is an example programmatic **ChannelSet** configuration.

```
Map configMap;
Map innerMap;
ElementList elementList;
elementList.addAscii( "DefaultConsumer", "Consumer_1" );
innerMap.addKeyAscii( "Consumer_1", MapEntry::AddEnum, ElementList()
.addAscii( "ChannelSet", "Channel_1, Channel_2" )
.addAscii( "Logger", "Logger_1" )
.addAscii( "Dictionary", "Dictionary_1" ).complete() ).complete();
elementList.addMap( "ConsumerList", innerMap );
elementList.complete();
innerMap.clear();
```

```

configMap.addKeyAscii( "ConsumerGroup", MapEntry::AddEnum, elementList );
elementList.clear();
innerMap.addKeyAscii( "Channel_1", MapEntry::AddEnum, ElementList()
.addEnum( "ChannelType", 0 )
.addAscii( "InterfaceName", "localhost" )
.addAscii( "Host", "localhost" )
.addAscii( "Port", "14002" ).complete() )
innerMap.addKeyAscii( "Channel_2", MapEntry::AddEnum, ElementList()
.addEnum( "ChannelType", 0 )
.addAscii( "InterfaceName", "localhost" )
.addAscii( "Host", "121.1.1.100" )
.addAscii( "Port", "14008" ).complete() ).complete();
elementList.addMap( "ChannelList", innerMap );
elementList.complete();
innerMap.clear();
configMap.addKeyAscii( "ChannelGroup", MapEntry::AddEnum, elementList );
elementList.clear();

innerMap.addKeyAscii( "Logger_1", MapEntry::AddEnum,
ElementList()
.addEnum( "LoggerType", 0 )
.addAscii( "FileName", "logFile" )
.addEnum( "LoggerSeverity", 1 ).complete() ).complete();
elementList.addMap( "LoggerList", innerMap );
elementList.complete();
innerMap.clear();
configMap.addKeyAscii( "LoggerGroup", MapEntry::AddEnum, elementList );
elementList.clear();
innerMap.addKeyAscii( "Dictionary_1", MapEntry::AddEnum,
ElementList()
.addEnum( "DictionaryType", 1 )
.addAscii( "RdmFieldDictionaryFileName", "./RDMFieldDictionary" )
.addAscii( "EnumTypeDefFileName", "./enumtype.def" ).complete() ).complete();
elementList.addMap( "DictionaryList", innerMap );
elementList.complete();
configMap.addKeyAscii( "DictionaryGroup", MapEntry::AddEnum, elementList );
elementList.clear();
configMap.complete();

```

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	UInt64	20000	Configures the minimum length of time (in milliseconds) to use as a timeout for a connected channel.
ConnectionPingTimeout	UInt64	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	UInt64	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.

Table 13: Universal <Server> Parameters

PARAMETER NAME	TYPE	DEFAULT	NOTES
CompressionType	Enumeration	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 enumeration values 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 (0) • ZLib (1) • LZ4 (2) <p>NOTE: You can configure a server to force a particular compression type, regardless of client settings.</p>
GuaranteedOutputBuffers	UInt64	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 C++ Edition Developers Guide</i>.</p>
HighWaterMark	UInt64	6144	Specifies the upper buffer-usage threshold for the channel.
InitializationTimeout	UInt64	60	Specifies the time (in seconds) to wait for the successful initialization of a channel.
InterfaceName	EmaString	""	<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>
MaxFragmentSize	UInt64	6144	Specifies the maximum size of a message fragment that can be sent without being fragmented and then reassembled upon delivery.
Name	EmaString		Specifies the Server's name.
NumInputBuffers	UInt64	10	<p>Specifies the number of buffers used to read data. Buffers are sized according to maxFragmentSize.</p> <p>For details on RsslChannel and maxFragmentSize, refer to the <i>Transport API C++ Edition Developers Guide</i>.</p>
Port	EmaString	14002	Specifies the port on the remote server to which the Enterprise Message API connects.
ServerSharedSocket	UInt64	0	<p>Specifies whether the server allows socket sharing. Available values include:</p> <ul style="list-style-type: none"> • 0: The server does not allow socket sharing. (this is the default behavior) • 1: The server allows socket sharing. <p>Socket sharing is available only with certain patch levels on Linux 6. Applications that intend to use this feature on Linux 6 must rebuild the RTSDK library (librssl) natively on a Linux 6 platform with the appropriate patch level that supports socket sharing.</p> <p>For further details on ServerSharedSocket, refer to the <i>Transport API C++ Edition Developers Guide</i>.</p>

Table 13: Universal <Server> Parameters (Continued)

PARAMETER NAME	TYPE	DEFAULT	NOTES
ServerType	Enumeration	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 enumeration values 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 RSSL_WEBSOCKET (7).</p> <p>NOTE: Setting ServerType to RSSL_SOCKET or RSSL_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	UInt64	65535	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	UInt64	65535	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	UInt64	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 13: Universal <Server> Parameters (Continued)

3.4.3 Parameters for Use with ServerType RSSL_ENCRYPTED

You can use the following parameters when **ServerType** is set to **RSSL_ENCRYPTED**.

PARAMETER NAME	TYPE	DEFAULT	DESCRIPTION
LibcryptoName	EmaString	Refer to the description	<p>Specifies the filename of the libcrypto library. If the file is not stored in the working directory, you must also specify the directory in this parameter (e.g., <code>/usr/lib64/libcrypto.so</code>).</p> <p>As a rule, EMA attempts to load libssl 1.1.x before attempting to load libssl 1.0.x. Because the Libssl library filename changes based on version and platform, EMA manifests the following defaults:</p> <ul style="list-style-type: none"> On Linux, in version 1.1.x, EMA attempts to load libcrypto.so.11. On Linux, in version 1.0.x, EMA attempts to load libcrypto.so.10. On Windows, in version 1.1.x, EMA attempts to load libcrypto-1_1-x64.dll. On Windows, in version 1.0.x, EMA attempts to load libeay32.dll.
LibsslName	EmaString	Refer to the description	<p>Specifies the filename of the Libssl library. If the file is not stored in the working directory, you must also specify the directory in this parameter (e.g., <code>/usr/lib64/libssl.so</code>). As a rule, EMA attempts to load libssl 1.1.x before attempting to load libssl 1.0.x. Because the Libssl library filename changes based on version and platform, EMA manifests the following defaults:</p> <ul style="list-style-type: none"> On Linux, in version 1.1.x, EMA attempts to load libssl.so.11. On Linux, in version 1.0.x, EMA attempts to load libssl.so.10. On Windows, in version 1.1.x, EMA attempts to load libssl-1_1-x64.dll. On Windows, in version 1.0.x, EMA attempts to load ssleay32.dll.
ServerCert	EmaString	""	Required. Specifies the filename of the server's certificate.
ServerPrivateKey	EmaString	""	Required. Specifies the filename of the server's private key.
CipherSuite	EmaString	""	Specifies an OpenSSL-formatted string of ciphers. By default, both EMA client and EMA server connections use cipher selections recommended by OWASP. Refer to ETA's rsstTransport.h for the current version's default ciphers.
DHParams	EmaString	""	Specifies the filename of a DH parameters file. By default, EMA will load a built-in DH parameter set.

Table 14: RSSL_ENCRYPTED ServerType Parameters

3.4.4 Parameters for Use with ServerType RSSL_WEBSOCKET

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

PARAMETER NAME	TYPE	DEFAULT	DESCRIPTION
WsProtocols	EmaString	tr_json2, rssl.rwf, rssl.json.v2	<p>Specifies a list of supported protocols in order of preference. Current protocols supported include rssl.json.v2, rssl.rwf, and tr_json2.</p>

Table 15: RSSL_WEBSOCKET ServerType Parameter

3.5 Logger Group

LoggerGroup contains a **LoggerList**, which contains one or more **Logger** components (each uniquely identified by a **<Name .../>** entry). A **Logger** component defines the parameters and behaviors for a single logging utility.

3.5.1 Generic XML Schema for LoggerGroup

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

```
<LoggerGroup>
  <LoggerList>
    <Logger>
      <Name value="..." />
      ...
    </Logger>
  </LoggerList>
</LoggerGroup>
```

3.5.2 Logger Entry Parameters

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

PARAMETER NAME	TYPE	DEFAULT	NOTES
FileName	EmaString	"emaLog_pid.log"	The Enterprise Message API ignores this parameter if LoggerType is set to Stdout (1).
IncludeDateInLoggerOutput	UInt64	0	Sets whether to include the date in the Enterprise Message API's log messages. Possible values are: <ul style="list-style-type: none"> 0 (false): Include only the time, omitting the date. 1 (true): Include both date and time.
Name	EmaString		Sets a unique name for the Logger component in the LoggerList .
LoggerSeverity	Enumeration	Success	Severity levels aggregate messages so that a severity level includes all messages from higher levels (e.g., a setting of 1 includes any messages normally printed at levels 2 and 3). Use enumeration values with the Enterprise Message API's programmatic configuration (for details, refer to in Section 4.5). Possible values are: <ul style="list-style-type: none"> LoggerSeverity::Verbose (0) LoggerSeverity::Success (1) LoggerSeverity::Warning (2) LoggerSeverity::Error (3) LoggerSeverity::NoLogMsg (4)

Table 16: Logger Group Parameters

PARAMETER NAME	TYPE	DEFAULT	NOTES
LoggerType	Enumeration	File	<p>Specifies the logging mechanism.</p> <p>Use enumeration values with the Enterprise Message API's programmatic configuration (for details, refer to in Section 4.5).</p> <p>Possible values are:</p> <ul style="list-style-type: none"> • LoggerType::File: The Enterprise Message API logs to the file specified in the parameter FileName. • LoggerType::Stdout: The Enterprise Message API logs to stdout.

Table 16: Logger Group Parameters (Continued)

3.6 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.

3.6.1 Generic XML Schema for DictionaryGroup

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

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

3.6.2 Dictionary Entry Parameters

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

PARAMETER NAME	TYPE	DEFAULT	NOTES
DictionaryType	Enumeration	ChannelDictionary	Specifies the dictionary loading mode. Use enumeration values 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	EmaString		Sets the location of the EnumTypeDef file.
EnumTypeDefItemName	EmaString	RWFEnum	Sets the name of the EnumTypeDef item specified in the source directory InfoFilter.DictionariesProvided , and InfoFilter.DictionariesUsed elements.
Name	EmaString		Sets a unique name for a Dictionary component in the DictionaryList .
RdmFieldDictionaryFileName	EmaString		Sets the location of the RdmFieldDictionary .
RdmFieldDictionaryItemName	EmaString	RWFFld	Sets the name of the RdmFieldDictionary item specified in the source directory InfoFilter.DictionariesProvided , and InfoFilter.DictionariesUsed elements.

Table 17: Dictionary Group Parameters

3.7 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**).

3.7.1 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>
        <LoadFilter>
          ...
        </LoadFilter>
      </Service>
    ...
  </Directory>
  ...
</DirectoryList>
</DirectoryGroup>
```

3.7.2 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" />
```

3.7.3 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	EmaString	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 18: Directory Entry Parameters

3.7.4 Service Entry Parameters

The Service Entry resembles the RDM's Source Directory Domain payload. For further details, refer to the *Enterprise Message API C++ 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	EmaString	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.
LoadFilter	Component Name	N/A	Specifies LoadFilter values for the given Service . LoadFilter values set a filter on the types of incoming information.
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 19: Service Entry Parameters

3.7.5 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	UInt64	N/A	Specifies the Service 's unique identifier. Available values include 0 - 65535.
Vendor	EmaString	N/A	Specifies the name of the vendor that provides the service.

Table 20: Source Directory Info Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
IsSource	UInt64	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.
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 3.7.5.1.
ItemList	EmaString	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 3.7.5.2.
AcceptingConsumerStatus	UInt64	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 3.7.5.3.
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 3.7.5.4.
SupportsQoSRange	UInt64	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 C++ Edition Usage Guide</i> .
SupportsOutOfBandSnapshots	UInt64	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 C++ Edition Usage Guide</i> .

Table 20: Source Directory Info Parameters (Continued)

3.7.5.1 CapabilitiesEntry Parameter

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

PARAMETER	TYPE	DEFAULT	DESCRIPTION
CapabilitiesEntry	UInt64 or EmaString	N/A	Specifies the message domain type supported by the Service component. Accepted names are listed in the emaRdm.h file. NOTE: You can set CapabilitiesEntry to be an RDM domain number or name (e.g. 6 or MMT_MARKET_PRICE).

Table 21: CapabilitiesEntry Parameter

3.7.5.2 DictionariesProvided Entry Parameter

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

PARAMETER	TYPE	DEFAULT	DESCRIPTION
DictionariesProvidedEntry	EmaString	RWFFId for RdmFieldDictionaryItemName RWFEEnum 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 22: DictionariesProvided Parameter

3.7.5.3 DictionariesUsed Entry Parameter

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

PARAMETER	TYPE	DEFAULT	DESCRIPTION
DictionariesUsedEntry	EmaString	RWFFId for RdmFieldDictionaryItemName RWFEEnum 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 23: DictionariesUsedEntry Parameter

3.7.5.4 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	UInt64 or EmaString	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.h file.
Rate	UInt64 or EmaString	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.h file.

Table 24: QoSEntry Section and Associated Parameters

3.7.6 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	UInt64	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	UInt64	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 25: StateFilter Parameters

3.7.7 LoadFilter Entry Parameters

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

PARAMETER	TYPE	DEFAULT	DESCRIPTION
OpenLimit	UInt64	0	Specifies the maximum number of streaming items that the client can open for this service.
OpenWindow	UInt64	1	Specifies the maximum number of outstanding requests (i.e., requests for items not yet open) that the service allows at any given time.
LoadFactor	UInt64	1	Indicates the current workload on the source providing data.

Table 26: LoadFilter Entry Parameters

3.7.8 Status Entry Parameters

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

PARAMETER	TYPE	DEFAULT	DESCRIPTION
StreamState	EmaString	StreamState::Open	Specifies the state of the item stream.
			NOTE: Acceptable StreamState values are listed in the OmmState.h file.
DataState	EmaString	DataState::Ok	Specifies the state of the item data.
			NOTE: Acceptable DataState values are listed in the OmmState.h file.
StatusCode	EmaString	StatusCode::None	Specifies the item status code.
			NOTE: Codes and their meanings are listed in the OmmState.h file.
StatusText	EmaString	""	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 27: Status Entry Parameters

4 Enterprise Message API 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 **Logger**, **Channel**, or **Dictionary** in their respective lists. Instead, if an application wants to use a specific channel, logger, 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 **Logger** or **Channel** in their respective lists. Instead, if an application wants to use a specific channel, logger, 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 **Logger** or **Server** in their respective lists. Instead, if an application wants to use a specific server, logger, or dictionary configuration, the application must explicitly configure it in the appropriate **IProvider** section of the XML file.

4.3 Processing the Enterprise Message API 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**, **Logger**, **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 C++ 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 the Working Directory

If **OmmConsumerConfig** lacks an argument, the application attempts to open a configuration file named **EmaConfig.xml** in the current working directory:

- If **EmaConfig.xml** exists and contains valid XML, the Enterprise Message API uses the XML to modify its configuration.
- If **EmaConfig.xml** exists, but is empty or contains malformed XML, the application uses the default configuration (for details on the default configuration, refer to Section 4.2).
- If **EmaConfig.xml** does not exist, 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 the working directory, have the application create an **OmmConsumerConfig** object (for details on this object, refer to the *Enterprise Message API C++ Developers Guide*) and pass it to the **OmmConsumer** object as follows:

```
OmmConsumerConfig config();

OmmConsumer consumer(config);
```

For complete details, you can refer to the example `100__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.

When the application finds the configuration file, the application processes it and applies the custom configuration to the default configuration. However, other errors can occur during processing such as the following:

- The file is empty
- The file cannot be opened
- Memory cannot be allocated for reading the file
- The file cannot be read
- The file contains invalid XML

If any of the preceding errors happen, the application throws an `IceException`, and the text associated with the application will indicate which error occurred.

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 C++ 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(PATH);

OmmConsumer consumer(config);
```

For an example of how to specify a custom configuration file name, refer to *Example 111 (111__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** which logs to a file named **emaLogfile**.

```
<ConsumerGroup>
  <ConsumerList>
    <Consumer>
      <Name value="Consumer_1"/>
      <Logger value="Logger_2"/>
    </Consumer>
  </ConsumerList>
</ConsumerGroup>
<LoggerGroup>
  <LoggerList>
    <Logger>
      <Name value="Logger_2"/>
      <LoggerType value="LoggerType::File"/>
      <FileName value="emaLogfile"/>
    </Logger>
  </LoggerList>
</LoggerGroup>
```

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

```
<FileName value="emaLogfile"/>
```

In this case, the Enterprise Message API application relies on its hard-coded behavior and uses the filename **emaLog_pid.log**.

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 **FileName** parameter needs to be contained in a **Logger** component entry):

- Configuration 1:

```
<LoggerGroup>
  <FileName value="Name"/>
  <LoggerList>
    ...
```

- Configuration 2:

```
<LoggerGroup>
  <LoggerList>
    <FileName value="Name"/>
    ...
```

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 the Enterprise Message API Using Function Calls

From an application standpoint, instantiating **OmmConsumerConfig** and **OmmNiProviderConfig** objects creates the initial configuration from the **DefaultXML.h** and 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 Configuration Function Calls

4.4.1.1 OmmConsumerConfig Class Function Calls

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

FUNCTION	DESCRIPTION
addAdminMsg(const ReqMsg&)	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.
applicationId(const EmaString &)	Sets the applicationId variable. applicationId has no default value.
clear()	Clears existing content from the OmmConsumerConfig object.
config(const Data&)	Passes in the consumer's programmatic configuration.

Table 28: OmmConsumerConfig Class Function Calls

FUNCTION	DESCRIPTION
consumerName(const EmaString &)	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.
clientId(String)	Sets the clientId variable. clientId has no default value. clientId specifies a unique ID for application making the request to the RDP token service.
host(const EmaString &)	Sets the host and port parameters. For details, refer to Section 4.4.2.
operationModel(OperationModel)	Sets the operation model to either OmmConsumerConfig::ApiDispatchEnum (which is the default) or OmmConsumerConfig::UserDispatchEnum .
password(const EmaString &)	Sets the password variable. password has no default value.
position(const EmaString &)	Sets the position variable. position has no default value.
username(const EmaString &)	Sets the username variable. If username is not set, the application extracts a username from the run-time environment.

Table 28: OmmConsumerConfig Class Function Calls (Continued)

4.4.1.2 Class Function Calls

You can use the following function calls in an Enterprise Message API **Provider** application. For further details on variables, refer to the *Enterprise Message API C++ RDM Usage Guide*. Certain function 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.

FUNCTION	DESCRIPTION
addAdminMsg(const ReqMsg&)	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 C++ RDM Usage Guide</i> .
addAdminMsg(const RefreshMsg&)	Populates part of or all of the initial directory refresh message according to the specification discussed in the <i>Enterprise Message API C++ RDM Usage Guide</i> .
adminControlDirectory(AdminControl)	Specifies whether the API or the user controls the sending of Directory refresh messages. Available values include: <ul style="list-style-type: none"> OmmProviderConfig::ApiControlEnum (which is the default) OmmProviderConfig::UserControlEnum For details on control models, refer to OmmProviderConfig.h .
applicationId(const EmaString &)	Used only with NiProvider . Sets the applicationId variable. applicationId has no default value.
clear()	Clears existing content from the OmmProviderConfig object.
config(const Data&)	Passes in the provider's programmatic configuration.
host(const EmaString &)	Used only with NiProvider . Sets the host and port parameters. For details, refer to Section 4.4.2.
instanceId(const EmaString&)	Used only with NiProvider . Sets the instanceId variable. instanceId has no default value.

Table 29: OmmProviderConfig Class Function Calls

FUNCTION	DESCRIPTION
operationModel(OperationModel)	Specifies whether the API or the user controls the thread (i.e., the operation model). Available values include: <ul style="list-style-type: none"> • OmmProviderConfig::ApiDispatchEnum(which is the default) • OmmProviderConfig::UserDispatchEnum For details on operation models, refer to OmmProviderConfig.h .
password(const EmaString &)	Used only with NiProvider . Sets the password variable. password has no default value.
port()	Sets the port parameters.
position(const EmaString &)	Used only with NiProvider . Sets the position variable. position has no default value.
providerName(const EmaString &)	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.
username(const EmaString &)	Used only with NiProvider . Sets the username variable. If username is not set, the application extracts a username from the run-time environment.

Table 29: OmmProviderConfig Class Function Calls (Continued)

4.4.2 Using the `host()` Function: How “Host” and “Port” 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(const EmaString&)` 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(const EmaString&)` 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:

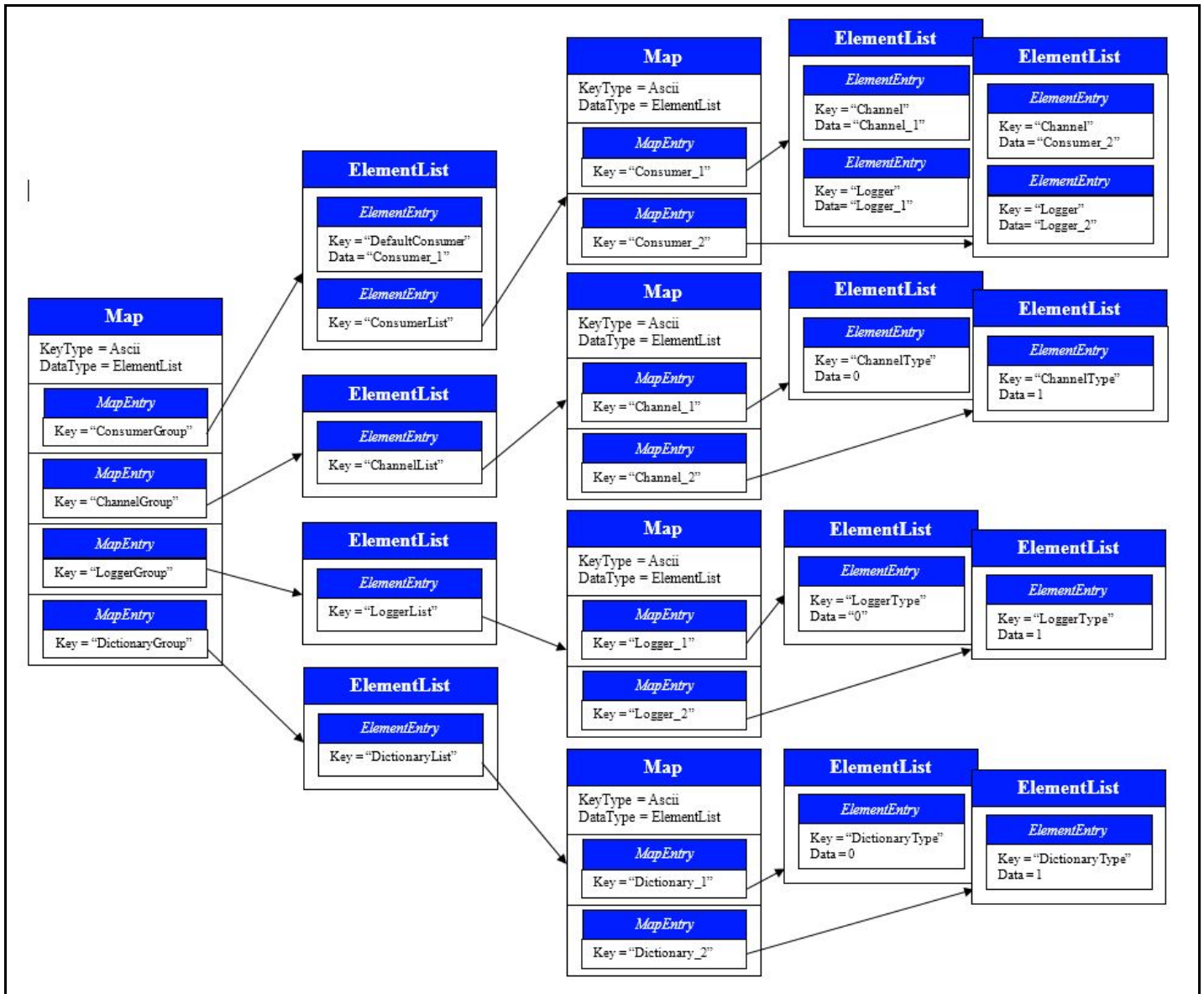
- The 1st tier lists the Enterprise Message API’s Consumer, IProvider, NiProvider, Channel, Logger, Directory, and Dictionary components; each of which has its own list in the 2nd tier.
- The 2nd tier includes each component’s list and the default consumers and providers for use when loading configuration 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 a Consumer

The following sample illustrates the Enterprise Message API programmatic configuration of a consumer:

```
Map configMap;
Map innerMap;
ElementList elementList;

elementList.addAscii( "DefaultConsumer", "Consumer_1" );

innerMap.addKeyAscii( "Consumer_1", MapEntry::AddEnum, ElementList()
    .addAscii( "Channel", "Channel_1" )
    .addAscii( "Logger", "Logger_1" )
    .addAscii( "Dictionary", "Dictionary_1" )
    .addUInt( "ItemCountHint", 5000 )
    .addUInt( "ServiceCountHint", 5000 )
    .addUInt( "ObeyOpenWindow", 0 )
    .addUInt( "PostAckTimeout", 5000 )
    .addUInt( "RequestTimeout", 5000 )
    .addInt( "ReconnectAttemptLimit", 10 )
    .addInt( "ReconnectMinDelay", 2000 )
    .addInt( "ReconnectMaxDelay", 6000 )
    .addUInt( "MaxOutstandingPosts", 5000 )
    .addInt( "DispatchTimeoutApiThread", 1 )
    .addUInt( "CatchUnhandledException", 0 )
    .addUInt( "MaxDispatchCountApiThread", 500 )
    .addUInt( "MaxDispatchCountUserThread", 500 )
    .addInt( "ReactorEventFdPort", 45000 )
    .addInt( "PipePort", 4001 ).complete() ).complete();
    .addAscii( "XmlTraceFileName", "MyXMLTrace" )
    .addInt( "XmlTraceMaxFileSize", 50000000 )
    .addUInt( "XmlTraceToFile", 1 )
    .addUInt( "XmlTraceToStdout", 0 )
    .addUInt( "XmlTraceToMultipleFiles", 1 )
    .addUInt( "XmlTraceWrite", 1 )
    .addUInt( "XmlTraceRead", 1 )
    .addUInt( "XmlTracePing", 1 )
    .addUInt( "MsgKeyInUpdates", 1 ).complete() ).complete();

elementList.addMap( "ConsumerList", innerMap );

elementList.complete();
innerMap.clear();

configMap.addKeyAscii( "ConsumerGroup", MapEntry::AddEnum, elementList );
elementList.clear();

innerMap.addKeyAscii( "Channel_1", MapEntry::AddEnum, ElementList()
    .addEnum( "ChannelType", 0 )
    .addAscii( "InterfaceName", "localhost" )
```

```

        .addEnum("CompressionType", 1)
        .addUInt( "GuaranteedOutputBuffers", 5000 )
        .addUInt( "ConnectionPingTimeout", 50000 )
        .addAscii( "Host", "localhost" )
        .addAscii("Port", "14002" )
        .addUInt( "TcpNodelay", 0 ).complete() ).complete();

elementList.addMap( "ChannelList", innerMap );

elementList.complete();
innerMap.clear();
configMap.addKeyAscii( "ChannelGroup", MapEntry::AddEnum, elementList );
elementList.clear();

innerMap.addKeyAscii( "Logger_1", MapEntry::AddEnum,
    ElementList()
        .addEnum( "LoggerType", 0 )
        .addAscii( "FileName", "logFile" )
        .addEnum( "LoggerSeverity", 1 ).complete() ).complete();

elementList.addMap( "LoggerList", innerMap );

elementList.complete();
innerMap.clear();

configMap.addKeyAscii( "LoggerGroup", MapEntry::AddEnum, elementList );
elementList.clear();

innerMap.addKeyAscii( "Dictionary_1", MapEntry::AddEnum,
    ElementList()
        .addEnum( "DictionaryType", 1 )
        .addAscii( "RdmFieldDictionaryFileName", "./RDMFieldDictionary" )
        .addAscii( "EnumTypeDefFileName", "./enumtype.def" ).complete() ).complete();

elementList.addMap( "DictionaryList", innerMap );

elementList.complete();

configMap.addKeyAscii( "DictionaryGroup", MapEntry::AddEnum, elementList );
elementList.clear();

configMap.complete();

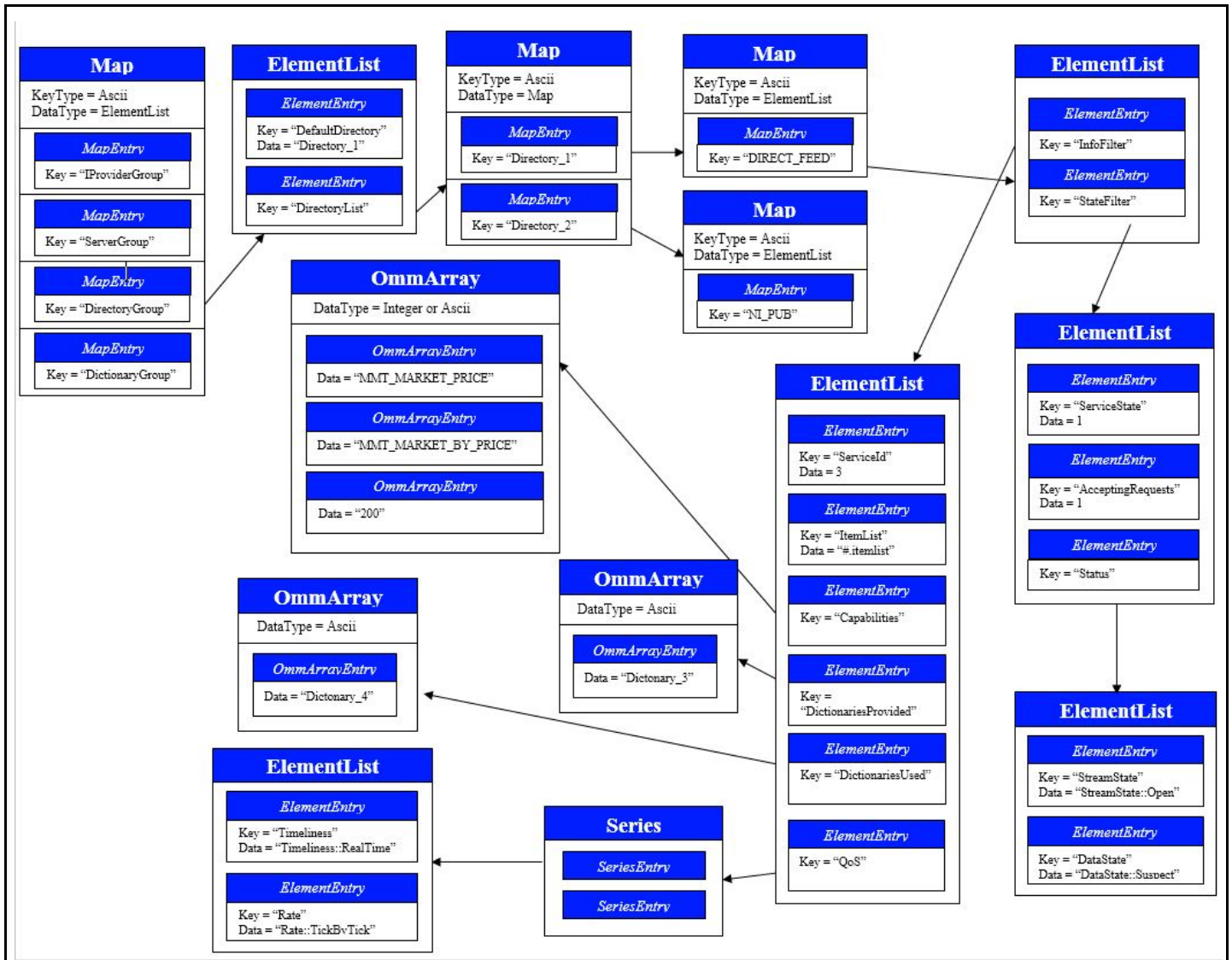
```

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 *IProvider*

4.5.5 Example: Programmatic Configuration of a Provider

The following sample illustrates the Enterprise Message API programmatic configuration of a provider:

```
Map outermostMap, innerMap;
ElementList elementList;

elementList.addAscii("DefaultIPProvider", "Provider_1");

innerMap.addKeyAscii("Provider_1", MapEntry::AddEnum, ElementList()
    .addAscii("Server", "Server_1")
    .addAscii("Logger", "Logger_1")
    .addAscii("Directory", "Directory_1")
    .addUInt("ItemCountHint", 5000)
    .addUInt("ServiceCountHint", 5000)
    .addUInt("RequestTimeout", 5000)
    .addInt("DispatchTimeoutApiThread", 1)
    .addUInt("CatchUnhandledException", 0)
    .addUInt("MaxDispatchCountApiThread", 500)
    .addUInt("MaxDispatchCountUserThread", 500)
    .addAscii("XmlTraceFileName", " MyXMLTrace")
    .addInt("XmlTraceMaxFileSize", 70000000)
    .addUInt("XmlTraceToFile", 0)
    .addUInt("XmlTraceToStdout", 1)
    .addUInt("XmlTraceToMultipleFiles", 0)
    .addUInt("XmlTraceWrite", 0)
    .addUInt("XmlTraceRead", 0)
    .addUInt("XmlTracePing", 1)
    .addUInt("XmlTraceHex", 1)
    .addInt("PipePort", 9696)
    .addUInt("RefreshFirstRequired", 1)
    .addUInt("AcceptDirMessageWithoutMinFilters", 0)
    .addUInt("AcceptMessageSameKeyButDiffStream", 0)
    .complete()
    .complete();

elementList.addMap("IPProviderList", innerMap).complete();
innerMap.clear();

outermostMap.addKeyAscii("IPProviderGroup", MapEntry::AddEnum, elementList);
elementList.clear();

innerMap.addKeyAscii("Server_1", MapEntry::AddEnum, ElementList()
    .addEnum("ServerType", 0)
    .addEnum("CompressionType", 1)
    .addUInt("GuaranteedOutputBuffers", 5000)
    .addUInt("NumInputBuffers", 5000)
    .addUInt("ConnectionPingTimeout", 70000)
    .addAscii("Port", "14002")
```

```

        .addUInt("TcpNodeDelay", 1)
        .complete()
        .complete();

elementList.addMap("ServerList", innerMap).complete();
innerMap.clear();

outermostMap.addKeyAscii("ServerGroup", MapEntry::AddEnum, elementList);
elementList.clear();

innerMap.addKeyAscii("Logger_1", MapEntry::AddEnum,
    ElementList()
        .addEnum("LoggerType", 0)
        .addAscii("FileName", "logFileProv")
        .addEnum("LoggerSeverity", 1).complete()
        .complete();

elementList.addMap("LoggerList", innerMap).complete();
innerMap.clear();

outermostMap.addKeyAscii("LoggerGroup", MapEntry::AddEnum, elementList);
elementList.clear();

innerMap.addKeyAscii("Dictionary_1", MapEntry::AddEnum,
    ElementList()
        .addEnum("DictionaryType", 0)
        .addAscii("RdmFieldDictionaryItemName", "RWFFld")
        .addAscii("EnumTypeDefItemName", "RWFEnum")
        .addAscii("RdmFieldDictionaryFileName", "./RDMFieldDictionary")
        .addAscii("EnumTypeDefFileName",
            "./enumtype.def").complete().complete();

elementList.addMap("DictionaryList", innerMap).complete();
innerMap.clear();

outermostMap.addKeyAscii("DictionaryGroup", MapEntry::AddEnum, elementList);
elementList.clear();

Map serviceMap;

serviceMap.addKeyAscii("DIRECT_FEED", MapEntry::AddEnum,
    ElementList()
        .addElementList("InfoFilter",
            ElementList().addUInt("ServiceId", 1)
                .addAscii("Vendor", "Vendor")
                .addUInt("IsSource", 1)
                .addUInt("AcceptingConsumerStatus", 1)
                .addUInt("SupportsQoSRange", 1)
                .addUInt("SupportsOutOfBandSnapshots", 1)
                .addAscii("ItemList", "#.itemlist")

```



```

        .addArray("Capabilities",
            OmmArray().addAscii("MMT_MARKET_PRICE")
                .addAscii("MMT_MARKET_BY_PRICE")
                .addAscii("MMT_MARKET_BY_ORDER")
                .addAscii("130")
                .complete())
        .addArray("DictionariesProvided",
            OmmArray().addAscii("Dictionary_1")
                .complete())
        .addArray("DictionariesUsed",
            OmmArray().addAscii("Dictionary_1")
                .complete())
        .addSeries("QoS",
            Series()
                .add(
                    ElementList().addAscii("Timeliness", "Timeliness::RealTime")
                        .addAscii("Rate", "Rate::TickByTick")
                        .complete())
                .complete())
        .complete())
    .addElementList("StateFilter",
        ElementList().addUInt("ServiceState", 1)
            .addUInt("AcceptingRequests", 1)
            .addElementList("Status",
                ElementList()
                    .addAscii("StreamState", "StreamState::Open")
                    .addAscii("DataState", "DataState::Suspect")
                    .addAscii("StatusCode", "StatusCode::DacsDown")
                    .addAscii("StatusText", "dacsDown")
                    .complete())
            .complete())
    .complete())
    .complete())

innerMap.addKeyAscii("Directory_1", MapEntry::AddEnum, serviceMap).complete();
elementList.clear();

elementList.addMap("DirectoryList", innerMap).complete();
outermostMap.addKeyAscii("DirectoryGroup", MapEntry::AddEnum, elementList).complete();

...

OmmProvider
provider( OmmIProviderConfig().config(outermostMap).operationModel(
    OmmIProviderConfig::UserDispatchEnum ), appClient );

```

© 2015 - 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: EMAC362CG.210
Date of issue: June 2021

