Enterprise Message API C++ Edition 3.9.0.L3

ENTERPRISE MESSAGE API CONFIGURATION GUIDE

LSEG DATA & ANALYTICS

Contents

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 LSEG 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 LSEG 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	LSEG Real-Time Advanced Distribution Hub is the horizontally scalable service component within the LSEG Real-Time Distribution System providing high availability for publication and contribution messaging, subscription management with optional persistence, conflation and delay capabilities.
ADS	LSEG Real-Time Advanced Distribution Server is the horizontally scalable distribution component within the LSEG 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 LSEG 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 Rssl Wire Format when publishing or consuming content to/from LSEG Real-Time Distribution Systems.
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol (Secure)
JSON	JavaScript Object Notation
JWK	JSON Web Key. Defined by RFC 7517, a JWK is a JSON formatted public or private key.
JWKS	JSON Web Key Set, This is a set of JWK, placed in a JSON array.
JWT	JSON Web Token. Defined by RFC 7519, JWT allows users to create a signed claim token that can be used to validate a user.
OMM	Open Message Model
QoS	Quality of Service
RDM	Domain Model
DP	Delivery Platform: this platform is used for REST interactions. In the context of Real-Time APIs, an API gets authentication tokens and/or queries Service Discovery to get a list of Real-Time - Optimized endpoints using DP.
LSEG Real-Time Distribution System	LSEG Real-Time Distribution System is LSEG's financial market data distribution platform. It consists of the LSEG Real-Time Advanced Distribution Server and LSEG Real-Time Advanced Distribution Hub. Applications written to the LSEG 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	Source Sink Library

Table 2: Acronyms and Abbreviations

ACRONYM / TERM	MEANING
RTT	Round Trip Time, this definition is used for round trip latency monitoring feature.
RWF	Rssl Wire Format, an LSEG proprietary binary format for data representation.
LDF-D	Data Feed Direct
UML	Unified Modeling Language
UTF-8	8-bit Unicode Transformation Format

Table 2: Acronyms and Abbreviations

1.5 References

- Enterprise Message API C++ Edition *LSEG Domain Model Usage Guide*

- *API Concepts Guide*

- Enterprise Message API C++ Edition *Developers Guide*

- Enterprise Message API C++ Edition *Reference Manual*

- Enterprise Message API C++ Edition *Performance Tools Guide*

- The LSEG Developer Community

1.6 Documentation Feedback

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

- Send us your comments via email at ProductDocumentation@lseg.com.

- Add your comments to the PDF using Adobe's **Comment** feature. After adding your comments, submit the entire PDF to LSEG by clicking **Send File** in the **File** menu. Use the ProductDocumentation@lseg.com address.

1.7 Document Conventions

This document uses the following types of conventions:

- Typographic

- Data Types

- Field and Text Values

- Boolean Values

1.7.1   Typographic

This document uses the following types of conventions:

- C++ classes, methods, 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();
```

1.7.2   Data Types

Data types within the configuration repository are as follows:

DATA TYPE	DEFINITION
Double	Stores decimal numbers with double precision.
EmaString	String
Enumeration	Specific text, as indicated in the field description
Int64	Signed, long integer
UInt64	Unsigned, long integer

Table 3: Data Type Conventions

1.7.3   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.

1.7.4   Boolean Values

When configuring a Boolean expression, you can use any number; however Enterprise Message API 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.

- *WarmStandby*: Warm standby channel configuration describes a warm standby group definition which is used to enable the warm standby feature for consumer applications.

- *WarmStandbyServerInfo*: Warm standby server configuration describes a channel configuration data to a starting or standby server and a set of preferred service names for the service based warm standby.

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 above 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"**).

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 API behavior by providing a valid alternate value either through the use of EmaConfig.xml, function calls, or programmatic methods. For default values for each of the parameters, see the appropriate Configuration Group section.

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.

Consumer has a functionality to change certain configured parameters dynamically at runtime using IOCtl method call `OmmConsumer.modifyReactorChannelIOCtl()`. For details, refer to Section 4.5.1.

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: • **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: • **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: • **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 3.4.

Table 4: Consumer Group Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ChannelSet	EmaString	N/A	Specifies a comma-separated set of channels names. Each listed channel name should have an appropriate **<Channel>** entry in the **ChannelGroup**. Channels in the set will be tried with each reconnection attempt until a successful connection is made. For further details refer to Section 3.4.9. **NOTE:** If both **Channel** and **ChannelSet** are configured, then the Enterprise Message API uses the parameter that is configured last in the file. For example, if **<Channel>** is configured after **<ChannelSet>** then the Enterprise Message API uses **<Channel>**, but if **<ChannelSet>** is configured after **<Channel>** then the Enterprise Message API uses **<ChannelSet>**.
CloseChannelFromConverterFailure	UInt64	1	Specifies that the Enterprise Message API should close the channel if the Enterprise Message API fails to parse JSON messages or if the Enterprise Message API receives JSON error messages. Possible values are: • **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**.
Dictionary	EmaString	N/A	Specifies how the consumer should access its dictionaries (it must match the **Name** parameter from the appropriate **<Dictionary>** entry in the **DictionaryGroup** configuration). If **Dictionary** is not specified, the Enterprise Message API uses the channel's dictionary when needed. For further details on this default behavior, refer to Section 3.10.
DictionaryRequestTimeOut	UInt64	45,000	Specifies the amount of time (in milliseconds) the application has to download dictionaries from a provider before the **OmmConsumer** throws an exception. If set to **0**, the Enterprise Message API will wait for a response indefinitely. **NOTE:** If **ChannelSet** is configured: • 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	Specifies the amount of time (in milliseconds) the provider has to respond with a source directory refresh message before the **OmmConsumer** throws an exception. If set to **0**, the Enterprise Message API will wait for a response indefinitely. **NOTE:** If **ChannelSet** is configured: • 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 4: 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.
EnablePreferredHostOptions	Boolean	False	This is used to enable or disable the preferred host feature. This parameter may be altered at runtime using IOCtl.
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: • 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: • **0** (false): Do not expand enumerated fields. • **1** (true): Expand enumerated fields.
JsonTokenIncrementSize	UInt64	500	Sets the number of JSON token increment size for parsing JSON messages.
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 3.9.
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: • 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()`.

Table 4: Consumer Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
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.
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: • **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**.
NegativeUpdateTypeFilter	UInt64	0	Specifies the types of update messages that the client should NOT receive from the server. The value is a bitmask, where each bit indicates a specific update type to exclude. Possible bit values are defined in the **EmaRdm.h** header file.
ObeyOpenWindow	UInt64	1	Specifies whether the **OmmConsumer** obeys the **OpenWindow** from services advertised in a provider's Source Directory response. Available values include: • **0** (false) • **1** (true)
OutputBufferSize	UInt64	65535	Sets the size of the output buffer for the RWF/JSON conversion. ⚠️ **WARNING!** If the buffer size is not large enough, the RWF/JSON conversion fails.
PHDetectionTimeInterval	UInt64	0	Specifies time interval in second unit to switch over to a preferred host or WarmStandby group. A value of 0 indicates that the detection time interval is disabled. Precedence: if `detectionTimeSchedule` is specified, this parameter is ignored. This parameter may be altered at runtime using IOCtl. This timer will start whenever the API library finishes its connection to a server. It will reset to the max whenever the connection is complete or if there is a change due to IOCtl.
PHDetectionTimeSchedule	String	Empty string	Specifies Cron time format to switch over to a preferred host or WarmStandby group. `detectionTimeInterval`, if specified, is used instead if this member is set to empty. This parameter may be altered at runtime using IOCtl.
PHFallBackWithInWSBGroup	Boolean	False	Specifies whether to fallback within a WarmStandby group instead of moving into a preferred WSB group. When setting `fallBackWithInWSBGroup` to false, the library closes all connections in the current WarmStandby group and attempts to connect to the preferred group. This cutover expects the starting active server to be available; if it is available, the library will connect to starting active and standby servers in the preferred group. This parameter may be altered at runtime using IOCtl.

Table 4: Consumer Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
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.
PreferredChannelName	String	0	Specifies name of channel to set as preferred host. This parameter may be altered at runtime using IOCtl.
PreferredWSBChannelName	String	0	Specifies name of WarmStandby Group to set as preferred WarmStandby group. This parameter may be altered at runtime using IOCtl.
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. **NOTE:** If setting **ReconnectAttemptLimit** to **N > 0** or a specific value like 11, adjust the **LoginRequestTimeout** value from 45 seconds.
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.
RestEnableLog	UInt64	0	Enables REST request/response logging for OmmConsumer interactions (not including Service Discovery) via filestream to either a file (see **RestLogFileName**) or `stdout`. Ability to dynamically enable or disable this parameter is supported. You can specify a log destination in **RestLogFileName**.
RestEnableLogViaCallback	UInt64	0	Enables REST request/response logging via callback. This parameter works in conjunction with setting the callback: `restLoggingCallback`. That is, logs are available by servicing callback: `onRestLoggingEvent`. Ability to dynamically enable or disable this parameter is supported.
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`).

Table 4: Consumer Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
RestProxyHostName	EmaString	N/A	Specifies the address or host name of the proxy server to which the Enterprise Message API connects for REST requests: service discovery and auth token service. Any value provided by a function call overrides the setting in the configuration file.
			NOTE: This proxy overrides the proxy settings (ProxyHost) for REST requests in the Channel group. Refer to Section 3.4.4 and Section 3.4.5.
RestProxyPort	EmaString	N/A	Specifies the port of the proxy server to which the Enterprise Message API connects for REST requests: service discovery and auth token service. Any value provided by a function call overrides the setting in configuration file.
			NOTE: This proxy overrides the proxy settings (ProxyPort) for REST requests in the Channel group. Refer to Section 3.4.4 and Section 3.4.5.
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.
RestVerboseMode	UInt64	0	Enables verbose REST logging. Ability to dynamically enable or disable this parameter is supported.
SendJsonConvError	UInt64	0	Set the RWF/JSON conversion to send back to the provider conversion error in case it occurs. Possible values are: • **0** (false): Do not send conversion error. • **1** (true): Send conversion error.
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**.
SessionChannelSet	String	N/A	Specifies a comma-separated set of session channel names. Each listed channel name should have an appropriate <**SessionChannelInfo**> entry in the **SessionChannelGroup**. A connection to session channels in the set will be established simultaneously.
			NOTE: If **Channel**, **ChannelSet** and **SessionChannelSet** are configured, then the Enterprise Message API uses the last **SessionChannelSet** in the file or programmatic configuration.
SessionEnhancedItemRecovery	int	1	Specifies that the Enterprise Message API should immediately attempt to recover the item, when the current connection goes down, if the item's requested service is available on another connection in the session channel set.
ShouldInitializeCPUIDlib	UInt64	1	Specifies whether the Enterprise Message API configures low level Enterprise Transport API to initialize CpuId library (default behavior), or not. Possible values are: • **0** (false): ETA should not initialize CpuId library. • **1** (true): ETA should initialize CpuId library. For further details, refer to the *Transport API C Edition Developers Guide*.

Table 4: Consumer Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
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 Real-Time - Optimized. The valid range is from **0.05** to **0.95**.
UpdateTypeFilter	UInt64	65533	Specifies the type of update messages that the client is expected to receive from the server. The value is a bitmask, where each bit indicates a specific update type. Possible bit values are defined in the **EmaRdm.h** header file.
WarmStandbyChannelSet	EmaString	N/A	Specifies a comma-separated set of warm standby channels names. Each listed name should have an appropriate **WarmStandbyChannel** entry in **WarmStandbyGroup**. Warm standby channels in the set will be moved to another warm standby channel when it failed to establish a connection for the entire warm standby channel. **NOTE:** • If both **WarmStandbyChannelSet** and **ChannelSet** are configured, then EMA uses the parameter that is configured in the **WarmStandbyChannelSet** before moving to **ChannelSet**. • When using both **WarmStandby** and **ChannelSet**, set **reconnectAttemptLimit** to a value other than default (-1) to avoid retrying servers within a **WarmStandbyGroup** indefinitely. • To ensure all servers are attempted before login timeout occurs, adjust **ReconnectAttemptLimit**, **ReconnectMinDelay**, **ReconnectMaxDelay**, and **LoginRequestTimeOut**.
XmlTraceDump	UInt64	0	Sets the Enterprise Message API to trace dump RWF messages after converting them from JSON messages. Possible values are: • **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: • **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: • **0** (false): Do not trace ping messages. • **1** (true): Trace ping messages.
XmlTracePingOnly	UInt64	0	Sets the Enterprise Message API to trace incoming and outgoing ping messages. Allows to trace pure ping messages without any other trace data. Possible values are: • **0** (false): Do not trace ping messages. • **1** (true): Trace ping messages.

Table 4: Consumer Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
XmlTraceRead	UInt64	1	Sets the Enterprise Message API to trace incoming data. Possible values are: • **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: • **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: • **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: • **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: • **0** (false): Do not trace outgoing data. • **1** (true): Trace outgoing data.

Table 4: Consumer Group Parameters (Continued)

3.2   IProvider Group

An **IProviderGroup** contains two elements that represent an Interactive Provider configuration:

- A **DefaultIProvider** element, which you can use to specify a default **IProvider** component. If a default **IProvider** is not specified in the **IProviderGroup**, the Enterprise Message API uses the first interactive provider listed in the **IProviderList**. For details on configuring a default **IProvider**, refer to Section 3.2.2.

- An **IProviderList** element, which contains one or more **IProvider** components. Each component should be uniquely identified by a **<Name …/>** entry.

The interactive provider component is the highest-level abstraction within an application. It typically refers to **Server**, **Logger**, and/or **Directory** components which specify provider capabilities.

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

For details on configuring an **IProviderGroup**, refer to Section 3.2.3.

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

3.2.1   Generic XML Schema for Provider Group

The generic XML schema for an **IProviderGroup** is as follows:

```
<IProviderGroup>
    <DefaultIProvider value="VALUE"/>
    <IProviderList>
        <IProvider>
            <Name value="VALUE"/>
            ...
        </IProvider>
    </IProviderList>
</IProviderGroup>
```

3.2.2   Setting a Default IProvider

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

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

3.2.3   Configuring an IProvider in an IProviderGroup

To configure an **IProvider** component, add the appropriate parameters to the target provider in the XML schema, each on a unique line. For a list of available **IProvider** 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 **IProvider** 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   IProvider Entry Parameters

Use the following parameters when configuring an **IProvider**.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
AcceptDirMessageWithoutMinFilters	UInt64	0	Sets the **IProvider** to accept incoming directory request messages without the minimum required INFO and STATE directory filters. Possible values are: • **0** (false): The **IProvider** will not accept incoming directory messages that do not contain the minimum required INFO and STATE filters. • **1** (true): The **IProvider** will accept incoming directory messages that do not contain the minimum required INFO and STATE filters.
AcceptMessageSameKeyButDiffStream	UInt64	0	Sets the **IProvider** 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. Possible values are: • **0** (false): The **IProvider** will not accept incoming request messages that match an existing request with a different stream ID. • **1** (true): The **IProvider** will accept incoming request messages that match an existing request with a different stream ID.
AcceptMessageThatChangesService	UInt64	0	Sets the **IProvider** to accept incoming consumer request messages on existing item stream that specify a different service name than the currently requested stream's service. Possible values are: • **0** (false): The **IProvider** will not accept incoming request messages on an existing item stream that specify a different service. • **1** (true): The **IProvider** will accept incoming request messages on an existing item stream that specify a different service.
AcceptMessageWithoutAcceptingRequests	UInt64	0	Sets the **IProvider** to accept incoming request messages even though the source directory is not accepting requests.

Table 5: IProviderGroup Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
AcceptMessageWithoutBeingLogin	UInt64	0	Sets the **IProvider** to accept incoming request messages even though the interactive provider has not accepted a login request.
AcceptMessageWithoutQosInRange	UInt64	0	Sets the **IProvider** to accept incoming request messages even though the requesting QoS is not in the QoS range of the source directory.
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. Possible values include: • **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: • **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: • **0** (false): Do not catch unknown JSON keys. • **1** (true): Catch unknown JSON keys.
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: • **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**.
Directory	EmaString	N/A	Specifies source directory refresh information that the **IProvider** 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 **<Directory>** entry and default settings, refer to Section 3.11.
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 thanzero, the thread goes active only if notified about a received message.

Table 5: IProviderGroup Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
EnforceAckIDValidation	UInt64	0	Specifies whether **IProvider** has to validate the **AckId** attribute when an **AckMsg** calls `OmmIProvider::submit()`. If validation is turned on, then **AckId** must be equal to the **PostId** of **PostMsg** received by the **IProvider**. Available values include: • **1** (true): Validate the **AckId**. • **0** (false): Do not validate the **AckId**.
EnumTypeFragmentSize	UInt64	12288	Sets the maximum fragmentation size (in bytes) of enumerated types dictionary multi-part refresh messages.
FieldDictionaryFragmentSize	UInt64	8192	Sets the maximum fragmentation size (in bytes) of field dictionary multi-part refresh messages.
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: • **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 3.9.
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 `OmmIProvider::dispatch()`.
Name	EmaString	N/A	Specifies the name of this **IProvider** component. **Name** is required when creating an **IProvider** component. You can use any value for **Name**.
OutputBufferSize	UInt64	65535	Sets the size of the output buffer for the RWF/JSON conversion. **WARNING!** If the buffer size is not large enough, the RWF/JSON conversion fails.
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: • **1** (true): The **IProvider** does not require that a refresh message is sent prior to update messages. • **0** (false): The **IProvider** requires that a refresh message is sent prior to update messages.

Table 5: IProviderGroup Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
RequestTimeout	UInt64	15000	Specifies the length of time (in milliseconds) the **OmmIProvider** 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.
SendJsonConvError	UInt64	0	Set the RWF/JSON conversion to send back to the consumer conversion error in case it occurs. Possible values are: • **0** (false): Do not send conversion error. • **1** (true): Send conversion error.
Server	EmaString	N/A	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 3.8.
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**.
ShouldInitializeCPUIDlib	UInt64	1	Specifies whether the Enterprise Message API configures low level Enterprise Transport API to initialize CpuId library (default behavior), or not. Possible values are: • **0** (false): ETA should not initialize CpuId library. • **1** (true): ETA should initialize CpuId library. For further details, refer to the *Transport API C Edition Developers Guide*.
XmlTraceDump	UInt64	0	Sets the Enterprise Message API to trace dump RWF messages after converting them from JSON messages. Possible values are: • **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: • **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: • **0** (false): Do not trace ping messages. • **1** (true): Trace ping messages.

Table 5: IProviderGroup Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
XmlTracePingOnly	UInt64	0	Sets the Enterprise Message API to trace incoming and outgoing ping messages. Allows to trace pure ping messages without any other trace data. Possible values are: • **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: • **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: • **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: • **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: • **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: • **0** (false): Do not trace outgoing data. • **1** (true): Trace outgoing data.

Table 5: IProviderGroup Parameters (Continued)

3.3   NiProvider Group

An **NiProviderGroup** contains two elements that represent a Non-interactive Provider configuration:

- A **DefaultNiProvider** element, which you can use to specify a default **NiProvider** component. If a default **NiProvider** is not specified in the **NiProviderGroup**, the Enterprise Message API uses the first non-interactive provider listed in the **NiProviderList**. For details on configuring a default **NiProvider**, refer to Section 3.3.2.

- An **NiProviderList** element, which contains one or more **NiProvider** components. Each component should be uniquely identified by a **<Name …/>** entry).

The non-interactive provider component is the highest-level abstraction within an application. It typically refers to **Channel**, **Logger**, and/or **Directory** components which specify provider capabilities.

For a generic **NiProviderGroup** XML schema, refer to Section 3.3.1.

For details on configuring an **NiProviderGroup**, refer to Section 3.3.3.

For a list of parameters you can use in configuring an **NiProvider**, refer to Section 3.3.4.

3.3.1   Generic XML Schema for NiProviderGroup

The generic XML schema for an **NiProviderGroup** is as follows:

```
<NiProviderGroup>
    <DefaultNiProvider value="VALUE"/>
    <NiProviderList>
        <NiProvider>
            <Name value="VALUE"/>
            ...
        </NiProvider>
    </NiProviderList>
</NiProviderGroup>
```


3.3.2   Setting a Default NiProvider

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

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

3.3.3   Configuring an NiProvider in an NiProviderGroup

To configure an **NiProvider** component, add the appropriate parameters to the target provider in the XML schema, each on a unique line. For a list of available **NiProvider** parameters, refer to Section 3.3.4.

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

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

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

3.3.4   NiProvider Entry Parameters

Use the following parameters when configuring an **NiProvider**.

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: • **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: • **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: • **0** (false): Do not catch unknown JSON keys. • **1** (true): Catch unknown JSON keys.
Channel	EmaString	N/A	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 3.4.

Table 6: NiProviderGroup Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ChannelSet	EmaString	N/A	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 3.4.9.
			NOTE: If both **Channel** and **ChannelSet** are configured, the Enterprise Message API uses the parameter configured last (linearly) in the file. For example: • 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: • **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**.
Directory	EmaString	N/A	Specifies source directory refresh information that the **NiProvider** 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 **<Directory>** entry and default settings, refer to Section 3.11.
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 thanzero, the thread goes active only if notified about a received 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: • **0** (false): Do not expand enumerated fields. • **1** (true): Expand enumerated fields.

Table 6: NiProviderGroup Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
Logger	EmaString	N/A	Specifies a set of logging behavior the **NiProvider** 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 3.9.
LoginRequestTimeOut	UInt64	45,000	Specifies the amount of time (in milliseconds) the consuming component has to respond with a login refresh message before the `OmmNiProvider` throws an exception. If set to **0**, the Enterprise Message API will wait for a response indefinitely. **NOTE:** When **ChannelSet** is configured, the Enterprise Message API honors **LoginRequestTimeOut** only on its first connection. If the channel supporting the first connection goes down, the Enterprise Message API does not use **LoginRequestTimeOut** on subsequent connections.
MaxDispatchCountApiThread	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 `OmmNiProvider::dispatch()`.
MaxEventsInPool	Int64	-1	Specifies the maximum number of event objects in the event's pool.
MergeSourceDirectoryStreams	UInt64	1	Specifies if the Enterprise Message API merges all source directory streams (configured and user-submitted) into one stream. Possible values are: • **1** (true) • **0** (false)
Name	EmaString	N/A	Specifies the name of this **NiProvider** component. **Name** is required when creating an **NiProvider** component. You can use any value for **Name**.
OutputBufferSize	UInt64	65535	Sets the size of the output buffer for the RWF/JSON conversion. **WARNING!** If the buffer size is not large enough, the RWF/JSON conversion fails.
ReconnectAttemptLimit	Int64	-1	Specifies the maximum number of times the non-interactive provider will attempt to reconnect to a channel when it fails. If set to **-1**, the non-interactive provider continually attempts to reconnect.
ReconnectMaxDelay	Int64	5000	Sets the maximum amount of time the non-interactive provider will wait (in milliseconds) before attempting to reconnect a failed channel. Refer also to the **ReconnectMinDelay** parameter.

Table 6: NiProviderGroup Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ReconnectMinDelay	Int64	1000	Specifies the minimum amount of time the non-interactive provider will wait (in milliseconds) before attempting to reconnect a failed channel. This wait time increases with each connection attempt, from **ReconnectMinDelay** to **ReconnectMaxDelay**.
RecoverUserSubmitSourceDirectory	Int64	1	Specifies whether the Enterprise Message API recovers user-submitted source directories when recovering from a disconnect. Possible values include: • **1** (true): The API will recover user-submitted source directories automatically on recovery from a disconnect. • **0** (false): The API will not recover user-submitted source directories.
RefreshFirstRequired	UInt64	1	Specifies whether the Enterprise Message API requires the application to send a refresh message prior to sending update messages. Possible values include: • **1** (true): The **NiProvider** does not require that a refresh message is sent prior to update messages. • **0** (false): The **NiProvider** requires that a refresh message is sent prior to update messages.
RemoveItemsOnDisconnect	UInt64	1	Specifies whether the Enterprise Message API removes items from its internal hash table whenever it disconnects from the LSEG Real-Time Advanced Distribution Hub. Possible values include: • **1** (true) • **0** (false)
RequestTimeout	UInt64	15000	Specifies the length of time (in milliseconds) the `OmmNiProvider` 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.
SendJsonConvError	UInt64	0	Set the RWF/JSON conversion to send back to the consumer conversion error when it occurs. Possible values are: • **0** (false): Do not send conversion error. • **1** (true): Send conversion error.
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**.
ShouldInitializeCPUIDlib	UInt64	1	Specifies whether the Enterprise Message API configures low level Enterprise Transport API to initialize CpuId library (default behavior), or not. Possible values are: • **0** (false): ETA should not initialize CpuId library. • **1** (true): ETA should initialize CpuId library. For further details, refer to the *Transport API C Edition Developers Guide*.

Table 6: NiProviderGroup Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
XmlTraceDump	UInt64	0	Sets the Enterprise Message API to trace dump RWF messages after converting them from JSON messages. Possible values are: • **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: • **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: • **0** (false): Do not trace ping messages. • **1** (true): Trace ping messages.
XmlTracePingOnly	UInt64	0	Sets the Enterprise Message API to trace incoming and outgoing ping messages. Allows to trace pure ping messages without any other trace data. Possible values are: • **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: • **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**. Possible values are: • **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: • **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: • **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: • **0** (false): Do not trace outgoing data. • **1** (true): Trace outgoing data.

Table 6: NiProviderGroup Parameters (Continued)

3.4   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 3.3.4.

- For a generic **ChannelGroup** XML schema, refer to Section 3.4.1.

- For a list of universal parameters you can use in configuring any type of **Channel** regardless of the channel type, refer to Section 3.4.2.

- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_SOCKET**, refer to Section 3.4.4.

- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_ENCRYPTED**, refer to Section 3.4.7.

- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_HTTP**, refer to Section 3.4.7.

- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_RELIABLE_MCAST**, refer to Section 3.4.8.

3.4.1   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>
```

3.4.2   Universal Channel Entry Parameters

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

For additional information on how to set the **Channel** connection type using the **ChannelType** and **EncryptedProtocolType** parameters, refer to Section 3.4.3.

PARAMETER NAME	TYPE	DEFAULT	NOTES
ChannelType	Enumeration	RSSL_SOCKET	Specifies the type of channel or connection used to connect to the server. Calling the host function can change this field. For details on this event, refer to Section 4.4.2. Use enumeration values with Enterprise Message API's programmatic configuration (for further details, refer to Section 4.6). Available values include: • **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.
DirectWrite	UInt64	0	Specifies whether to set the direct socket write flag when sending data on a channel. When the flag is set, every package is sent on the wire immediately on the submit call. If direct write is not set, the package might be placed into an internal queue which is later flushed onto the wire. Possible values are: • **0**: Send data without the direct socket write flag. • **1**: Send data with the direct socket write flag.
EnableSessionManagement	UInt64	0	Specifies whether the channel manages the authentication token on behalf of the user. If set to **1**, the channel obtains the authentication token and refreshes it as needed on behalf of the user. The default setting is **0**. You can use this parameter only in with Enterprise Message API consumers. When **EnableSessionManagement** is set and used with implicit Service Discovery, the application must configure ChannelType to be **RSSL_ENCRYPTED** because endpoints obtained by querying RDP Service Discovery are encrypted endpoints.
GuaranteedOutputBuffers	UInt64	100	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. For details on RsslChannel and **maxFragmentSize**, refer to the *Transport API C++ Edition Developers Guide*.
HighWaterMark	UInt64	6000	Specifies the upper buffer-usage threshold for the channel.

Table 7: Universal <Channel> Parameters

PARAMETER NAME	TYPE	DEFAULT	NOTES
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	""	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. **InterfaceName** is for use in systems that have multiple network interface cards. If unspecified, the default network interface is used.
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	100	Specifies the number of buffers used to read data. Buffers are sized according to **maxFragmentSize**. For details on RsslChannel and **maxFragmentSize**, refer to the *Transport API C++ Edition Developers Guide*.
ServiceDiscoveryRetryCount	UInt64	3	Specifies the number of times the RTSDK API attempts to reconnect a channel before forcing the API to retry service discovery. Used only when: • Host and port are unspecified. Refer to Section 3.4.4. • **EnableSessionManagement** is set to **1**. For details on service discovery, refer to the *Enterprise Message API C++ Edition Developers Guide*. **NOTE:** You can use this parameter only with Enterprise Message API consumers. API will not retry to get an endpoint from the service discovery when the value is **0**.
SysRecvBufSize	UInt64	On Windows: 65535 On Linux: Specified by system	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	On Windows: 65535 On Linux: Specified by system	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)

3.4.3   EMA Channel Connection Types

Following are sample snippets from the configuration file that show how to set up the Channel connection type:

```
<EncryptedProtocolType value="EncryptedProtocolType::RSSL_WEBSOCKET"/>
<ChannelType value="ChannelType::RSSL_ENCRYPTED"/>
```

The following table summarizes possible Channel connection types and parameter values that you can use to set them.

CHANNEL CONNECTION TYPE	CHANNELTYPE	ENCRYPTEDPROTOCOLTYPE	SUBPROTOCOL LIST POSSIBILITIES
Unencrypted Websocket	RSSL_WEBSOCKET	Not used	The following possibilities apply: • If the **WsProtocols** parameter is empty or mismatched with server support, websocket connections are denied. • Default: **WsProtocols value="tr_json2, rssl.rwf, rssl.json.v2"**
Encrypted Websocket	RSSL_ENCRYPTED	RSSL_WEBSOCKET	
Unencrypted Socket	RSSL_SOCKET	Not used	Not used; RWF is implied.
Encrypted Socket	RSSL_ENCRYPTED	RSSL_SOCKET	

Table 8: Channel Settings for Socket and Websocket Connection Types

3.4.4   Parameters for Use with Channel Type: RSSL_SOCKET

In addition to the universal parameters listed in Section 3.4.2, 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	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 4.6). Available values include: • **None** (0) • **ZLib** (1) • **LZ4** (2) **NOTE:** A server can be configured to force a particular compression type, regardless of client settings.

Table 9: Parameters for Channel Type: RSSL_SOCKET

PARAMETER NAME	TYPE	DEFAULT	NOTES
DirectWrite	UInt32	0	Specifies whether to set the direct socket write flag when sending data on a channel. When the flag is set, every package is sent on the wire immediately on the submit call. If direct write is not set, the package might be placed into an internal queue which is later flushed onto the wire. Possible values are: • **0**: Send data without the direct socket write flag. • **1**: Send data with the direct socket write flag. For more information on the direct socket write flag and **rsslWrite**, refer to the *Transport API C Edition Developers Guide*.
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.
Port	EmaString	14002	Specifies the port on the remote server to which the Enterprise Message API connects.
ProxyConnectionTimeout	UInt64	40	Specifies the maximum time in seconds a connection with proxy is allowed to be established.
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. **NOTE:** This parameter can be used to invoke REST requests such as service discovery and obtain an authentication token. They have lower precedence than the proxy settings (**RestProxyHostName**) specified in the Consumer group. Refer to Section 3.1.4.
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. **NOTE:** This parameter can be used to invoke REST requests such as service discovery and obtain an authentication token. They have lower precedence than the proxy settings (**RestProxyPort**) specified in the Consumer group. Refer to Section 3.1.4.
TcpNodelay	UInt64	1	Specifies whether to use Nagle's algorithm when sending data. Available values are: • **0**: Send data using Nagle's algorithm. • **1**: Send data without delay.

Table 9: Parameters for Channel Type: RSSL_SOCKET (Continued)

3.4.5   Parameters for Use with Channel Types: RSSL_HTTP

In addition to the universal parameters listed in Section 3.4.2, 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 4.6). Available values include: • **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. **NOTE:** This parameter can be used to invoke REST requests such as service discovery and obtain an authentication token. They have lower precedence than the proxy settings (**RestProxyHostName**) specified in the Consumer group. Refer to Section 3.1.4.
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. **NOTE:** This parameter can be used to invoke REST requests such as service discovery and obtain an authentication token. They have lower precedence than the proxy settings (**RestProxyPort**) specified in the Consumer group. Refer to Section 3.1.4.
TcpNodelay	UInt64	1	Specifies whether to use Nagle's algorithm when sending data. Available values are: • **0**: Send data using Nagle's algorithm. • **1**: Send data without delay.

Table 10: Parameters for Channel Type: RSSL_HTTP

3.4.6   Parameters for Use with Channel Types: RSSL_WEBSOCKET

In addition to the universal parameters listed in Section 3.4.2, 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 11: Parameters for Channel Types: RSSL_WEBSOCKET

3.4.7   Parameters for Use with Channel Types: RSSL_ENCRYPTED

In addition to the universal parameters listed in Section 3.4.2, and the parameters listed in the section specific to the protocol type you use (i.e., Section 3.4.4 for socket connections or Section 3.4.5 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_SOCKET On Linux: RSSL_SOCKET RSSL_WEBSOCKET	Specifies the type of protocol used for this encrypted connection. • **RSSL_SOCKET** (0) • **RSSL_HTTP** (2): Supported only on WinINet-based connections using the Windows OS (for backwards compatibility). • **RSSL_WEBSOCKET** (7) **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 3.4.7.
OpenSSLCAStore	EmaString	""	Specifies either: • The directory that contains the certificate authority store or, • The certificate authority certificate (a file). If you specify a file, LSEG recommends that you include the directory; otherwise your system will use environment variables to locate the file. If `openSSLCAStore` is blank or `'\0'`, Enterprise Message API's default behavior depends on the Operating System as follows: • Windows: the API loads root certificates from Windows root certificate store into OpenSSL. To manage additional certificates, use the Windows OS certificate manager. • Linux: the API's behavior is handled by OpenSSL as follows: - 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	4	When using OpenSSL, `SecurityProtocol` specifies the combination of flag values that set the version(s) of the TLS encryption protocol used for this connection.

Table 12: Parameters for Channel Types: RSSL_ENCRYPTED

3.4.8   Parameters for Use with Channel Type: RSSL_RELIABLE_MCAST

In addition to the universal parameters listed in Section 3.4.2, 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 *ADH Software Installation Manual*.

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

PARAMETER NAME	TYPE	DEFAULT	NOTES
DisconnectOnGap	UInt64	0	Specifies whether the underlying connection should be closed if a multicast gap situation is detected. • **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**. • **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 13: 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 * 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 13: Parameters for Channel Type: RSSL_RELIABLE_MCAST (Continued)

3.4.9   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>
```

3.4.10   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();
```

3.5   Warm Standby Channel Group

WarmStandbyGroup is used only with the **Consumer**.

The **WarmStandbyGroup** contains a **WarmStandbyList**, which contains one or more **WarmStandbyChannel** entries (each uniquely identified by a **<Name …/>** entry). Each warm standby channel includes a set of warm standby channel parameters for specifying a starting active server, standby server(s), and a warm standby mode.

3.5.1   Generic XML Schema for WarmStandbyGroup

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

```
<WarmStandbyGroup>
    <WarmStandbyList>
        <WarmStandbyChannel>
            <Name value="VALUE"/>
            ...
        </WarmStandbyChannel>
    </WarmStandbyList>
</WarmStandbyGroup>
```

3.5.2   Warm Standby Channel Entry Parameters

Use the following parameters when configuring the **WarmStandbyGroup**.

PARAMETER NAME	TYPE	DEFAULT	NOTES
StartingActiveServer	EmaString	N/A	Specifies a starting server information to establish an initial connection. The name should have appropriate **<WarmStandbyServerInfo>** entry in the **<WarmStandbyServerInfoGroup>**.
StandbyServerSet	EmaString	N/A	Specifies a comma-separated set of warm standby server information to establish connection(s). The name should have appropriate **<WarmStandbyServerInfo>** entry in the **<WarmStandbyServerInfoGroup>**.
WarmStandbyMode	Enumeration	LOGIN_BASED	Specifies the warm standby mode used by warm standby channel. Use enumeration values with Enterprise Message API programmatic configuration (for details, refer to Section 4.6). Available values include: • LOGIN_BASED(1) • SERVICE_BASED(2)

Table 14: Warm Standby Channel Entry Parameters

3.6   Warm Standby ServerInfo Group

WarmStandbyServerInfoGroup is used only with the **Consumer**.

The **WarmStandbyServerInfoGroup** contains a **WarmStandbyServerInfoList**, which contains one or more **WarmStandbyServerInfo** entries (each uniquely identified by a **<Name …/>** entry). Each warm standby server information is associated with a channel name and a set of preferred service names for service based warm standby.

When configuring for per-service based warm standby mode, the preferred service name set may be used to identify that channel as the starting channel for a given service.

3.6.1   Generic XML Schema for WarmStandbyServerInfoGroup

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

```
<WarmStandbyServerInfoGroup>
     <WarmStandbyServerInfoList>
         <WarmStandbyServerInfo>
              <Name value="VALUE"/>

              ...
         </WarmStandbyServerInfo>
     </WarmStandbyServerInfoList>
</WarmStandbyServerInfoGroup>
```

3.6.2   Warm Standby ServerInfo Entry Parameters

Use the following parameters when configuring the **WarmStandbyServerInfoGroup**.

PARAMETER NAME	TYPE	DEFAULT	NOTES
Channel	EmaString	N/A	Specifies the channel that the warm standby channel should use. This channel must match the **Name** parameter from the appropriate **<Channel>** entry in the **ChannelGroup** configuration. For details on the **<Channel>** entry and default behaviors, refer to Section 3.4.
PerServiceNameSet	EmaString	N/A	Specifies a comma-separated set of preferred service name(s) for this server information. **NOTE:** This is used for the service based warm standby only.

Table 15: Warm Standby ServerInfo Entry Parameters

3.7   SessionChannel Group

SessionChannelGroup is used only with the Consumer for the request routing feature in order to route requests to multiple connections.

The **SessionChannelGroup** contains a **SessionChannelList**, which contains one or more **SessionChannelInfo** entries, each uniquely identified by a <**Name .../**> entry. Each session channel info includes **ChannelSet** and/or **WarmStandbyChannelSet** for specifying a connection and reconnect parameters.

3.7.1   Generic XML Schema for SessionChannelGroup

The top-level XML schema for the **SessionChannelGroup** in as follows:

```
<SessionChannelGroup>
    <SessionChannelList>
        <SessionChannelInfo>
            <Name value="VALUE"/>
            …
        </SessionChannelInfo>
    </SessionChannelList>
</SessionChannelGroup>
```

3.7.2   SessionChannel Entry Parameters

Use the following parameters when configuring the **SessionChannelGroup**.

PARAMETER NAME	TYPE	DEFAULT	NOTES
ChannelSet	String	N/A	Specifies a comma-separated set of channels names. Each listed channel name should have an appropriate <**Channel**> entry in the **ChannelGroup**. Channels in the set will be tried with each reconnection attempt until a successful connection is made. For further details refer to Section 3.4.9.
EnablePreferredHostOptions	Boolean	False	This is used to enable or disable the preferred host feature. This parameter may be altered at runtime using IOCtl.
			NOTE: This parameter must also be set to true to enable preferred host on this **SessionChannel**.
Name	String	N/A	Specifies the name of this Session Channel component. Name is required when creating a Session Channel component. This will be specified in the Consumer's **SessionChannelSet**. You can use any value for name.
PHDetectionTimeInterval	UInt64	0	Specifies time interval in second unit to switch over to a preferred host or WarmStandby group. A value of 0 indicates that the detection time interval is disabled. Precedence: if `detectionTimeSchedule` is specified, this parameter is ignored. This parameter may be altered at runtime using IOCtl. This timer will start whenever the API library finishes its connection to a server. It will reset to the max whenever the connection is complete or if there is a change due to IOCtl.
			NOTE: This parameter has higher precedence than the **PHDetectionTimeInterval** parameter specified in the Consumer entry.

Table 16: SessionChannel Entry Parameters

PARAMETER NAME	TYPE	DEFAULT	NOTES
PHDetectionTimeSchedule	String	Empty string	Specifies Cron time format to switch over to a preferred host or WarmStandby group. `detectionTimeInterval`, if specified, is used instead if this member is set to empty. This parameter may be altered at runtime using IOCtl.
			NOTE: This parameter has higher precedence than the **PHDetectionTimeSchedule** parameter specified in the Consumer entry.
PHFallBackWithInWSBGroup	Boolean	False	Specifies whether to fallback within a WarmStandby group instead of moving into a preferred WSB group. When setting `fallBackWithInWSBGroup` to false, the library closes all connections in the current WarmStandby group and attempts to connect to the preferred group. This cutover expects the starting active server to be available; if it is available, the library will connect to starting active and standby servers in the preferred group. This parameter may be altered at runtime using IOCtl.
			NOTE: This parameter has higher precedence than the **PHFallBackWithInWSBGroup** parameter specified in the Consumer entry.
PreferredChannelName	String	0	Specifies name of channel to set as preferred host. This parameter may be altered at runtime using IOCtl.
PreferredWSBChannelName	String	0	Specifies name of WarmStandby Group to set as preferred WarmStandby group. This parameter may be altered at runtime using IOCtl.
ReconnectAttemptLimit	int	-1	Specifies the maximum number of times the consumer and non-interactive provider attempt to reconnect to a channel when it fails. If set to **-1**, the consumer and non-interactive provider continually attempt to reconnect.
			NOTE: This parameter has higher precedence than the **ReconnectAttemptLimit** parameter specified in the Consumer entry.
ReconnectMaxDelay	int	5000	Sets the maximum amount of time the consumer and non-interactive provider wait (in milliseconds) before attempting to reconnect a failed channel. Refer also to the **ReconnectMinDelay** parameter.
			NOTE: This parameter has higher precedence than the **ReconnectMaxDelay** parameter specified in the Consumer entry.
ReconnectMinDelay	int	1000	Specifies the minimum amount of time the consumer and non-interactive provider wait (in milliseconds) before attempting to reconnect a failed channel. This waits time increases with each connection attempt, from **ReconnectMinDelay** to **ReconnectMaxDelay**.
			NOTE: This parameter has higher precedence than the **ReconnectMinDelay** parameter specified in Consumer entry.

Table 16: SessionChannel Entry Parameters

PARAMETER NAME	TYPE	DEFAULT	NOTES
WarmStandbyChannelSet	String	N/A	Specifies a comma-separated set of WarmStandby channels names. Each listed name should have an appropriate <**WarmStandbyChannel**> entry in **WarmStandbyGroup**. WarmStandby channels in the set will be moved to another WarmStandby channel when it failed to establish a connection for the entire WarmStandby channel.
			NOTE: • If both **WarmStandbyChannelSet** and **ChannelSet** are configured, then EMA uses the parameter that is configured in the **WarmStandbyChannelSet** before moving to **ChannelSet**. • When using both **WarmStandby** and **ChannelSet**, set **reconnectAttemptLimit** to a value other than default (-1) to avoid retrying servers within a **WarmStandbyGroup** indefinitely.
XmlTraceDump	UInt64	0	Sets this session channel to trace dump RWF messages after converting them from JSON messages. Possible values are: • **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: • **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 this session channel to trace incoming and outgoing ping messages. Possible values are: • **0** (false): Do not trace ping messages. • **1** (true): Trace ping messages.
XmlTracePingOnly	UInt64	0	Sets this session channel to trace incoming and outgoing ping messages. Allows to trace pure ping messages without any other trace data. Possible values are: • **0** (false): Do not trace ping messages. • **1** (true): Trace ping messages.
XmlTraceRead	UInt64	1	Sets this session channel to trace incoming data. Possible values are: • **0** (false): Do not trace incoming data. • **1** (true): Trace incoming data
XmlTraceToFile	UInt64	0	Sets whether this session channel traces its messages to an XML file whose name is set by **XmlTraceFileName**. Available values are: • **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: • **1** (true): this session channel writes the XML trace to a new file if the current file size reaches the **XmlTraceMaxFileSize**. • **0** (false): this session channel stops writing the XML trace if the current file reaches the **XmlTraceMaxFileSize**.

Table 16: SessionChannel Entry Parameters

PARAMETER NAME	TYPE	DEFAULT	NOTES
XmlTraceToStdout	UInt64	0	Specifies whether this session channel traces its messages in XML format to stdout. Possible values are: • **0** (false): Turns off tracing. • **1** (true): Turns on tracing to stdout.
XmlTraceWrite	UInt64	1	Sets the this session channel to trace outgoing data. Possible values are: • **0** (false): Do not trace outgoing data. • **1** (true): Trace outgoing data.

Table 16: SessionChannel Entry Parameters

3.8   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 3.8.1.

- For a list of parameters you can use in configuring **Server**, refer to Section 3.8.2.

3.8.1   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>
```

3.8.2   Server Entry Parameters

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

For additional information on how to set the **Server** connection type using the **ServerType** parameter, refer to Section 3.8.3.

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.
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 4.6. Available values include: • **None** (0) • **ZLib** (1) • **LZ4** (2)
			NOTE: You can configure a server to force a particular compression type, regardless of client settings.
DirectWrite	UInt32	0	Specifies whether to set the direct socket write flag when sending data on a channel. When the flag is set, every package is sent on the wire immediately on the submit call. If direct write is not set, the package might be placed into an internal queue which is later flushed onto the wire. Possible values are: • **0**: Send data without the direct socket write flag. • **1**: Send data with the direct socket write flag. For more information on the direct socket write flag and **rsslWrite**, refer to the *Transport API C Edition Developers Guide*.
GuaranteedOutputBuffers	UInt64	100	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. For details on **RsslChannel** and **maxFragmentSize**, refer to the *Transport API C++ Edition Developers Guide*.
HighWaterMark	UInt64	6000	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.

Table 17: Universal <Server> Parameters

PARAMETER NAME	TYPE	DEFAULT	NOTES
InterfaceName	EmaString	""	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. **InterfaceName** is for use in systems that have multiple network interface cards. If unspecified, the default network interface is used.
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	100	Specifies the number of buffers used to read data. Buffers are sized according to **maxFragmentSize**. For details on `RsslChannel` and **maxFragmentSize**, refer to the *Transport API C++ Edition Developers Guide*.
Port	EmaString	14002	Specifies the port on the remote server to which the Enterprise Message API connects.
ServerSharedSocket	UInt64	0	Specifies whether the server allows socket sharing. Available values include: • **0**: The server does not allow socket sharing. (this is the default behavior) • **1**: The server allows socket sharing. 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. For further details on **ServerSharedSocket**, refer to the *Transport API C++ Edition Developers Guide*.
ServerType	Enumeration	RSSL_SOCKET	Specifies the type of channel or connection used to connect to the server. Calling the host function can change this field. For details on this event, refer to Section 4.4.2. Use enumeration values with Enterprise Message API's programmatic configuration. For further details, refer to Section 4.6. Available values include **RSSL_SOCKET (0)**, **RSSL_ENCRYPTED (1)**, or **RSSL_WEBSOCKET (7)**. **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.
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	Specifies whether to use Nagle's algorithm when sending data. Available values are: • **0**: Send data using Nagle's algorithm. • **1**: Send data without delay.

Table 17: Universal <Server> Parameters (Continued)

3.8.3   EMA Server Connection Types

Following is a sample snippet from the configuration file that shows how to set up the ServerType parameter:

```
<ServerType value="ServerType::RSSL_SOCKET"/>
```

The following table summarizes possible Server connection types and parameter values that you can use to set them.

SERVER CONNECTION TYPE	SERVERTYPE VALUE	SUBPROTOCOL LIST POSSIBILITIES
Unencrypted Socket	RSSL_SOCKET	The following possibilities apply:
Unencrypted Websocket	RSSL_WEBSOCKET	• By default, either websocket or socket incoming connections are accepted.
Encrypted Socket	RSSL_ENCRYPTED	• If the **WsProtocols** parameter is empty or mismatched with incoming connections, websocket connections are denied. • Socket connection logic ignores **WsProtocols configuration** and supports only RWF. • Default:
Encrypted Websocket		**WsProtocols value="tr_json2, rssl.rwf, rssl.json.v2"**

Table 18: Server Settings for Socket and Websocket Connection Types

3.8.4   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
CipherSuite	EmaString	""	Specifies an OpenSSL-formatted string of ciphers. By default, both Enterprise Message API client and Enterprise Message API server connections use cipher selections recommended by OWASP. Refer to Enterprise Transport API's **rsslTransport.h** for the current version's default ciphers.
DHParams	EmaString	""	Specifies the filename of a DH parameters file. By default, Enterprise Message API will load a built-in DH parameter set.
SecurityProtocol	UInt64	12(0x04 and 0x08 representing TLS1.2 and TLS1.3 respectively)	This is an unsigned integer representing a combination of flags specified in Enterprise Transport API's **RsslEncryptionProtocolTypes** defined in **rsslTransport.h**.
ServerCert	EmaString	""	**Required**. Specifies the filename of the server's certificate.
ServerPrivateKey	EmaString	""	**Required**. Specifies the filename of the server's private key.

Table 19: RSSL_ENCRYPTED ServerType Parameters

3.8.5   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	Specifies a list of supported protocols in order of preference. Current protocols supported include `rssl.json.v2`, `rssl.rwf`, and `tr_json2`.

Table 20: RSSL_WEBSOCKET ServerType Parameter

3.9   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.

3.9.1   Generic XML Schema for LoggerGroup

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

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

3.9.2   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"	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. 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: • **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**.

Table 21: Logger Group Parameters

PARAMETER NAME	TYPE	DEFAULT	NOTES
NumberOfLogFiles	UInt64	0	Specifies the number of log files that are rolled. Possible values are: • 0: A file with unique pid name is created. Log file rolling is disabled by default. • 1..4294967296: A file with name "**emaLog_N.log**" is created. Where "**emaLog**" is configured by "**FileName**" parameter of this section, and "**N**" is a sequential number starting from 1 till "**NumberOfLogFiles**". There are four scenarios with parameters **NumberOfLogFiles** and **MaxLogFileSize**: **Scenario 1:** Defaults Value: **NumberOfLogFiles**=0; **MaxLogFileSize**=0 A file with unique pid name is created and grows indefinitely. **Scenario 2:** **NumberOfLogFiles** > 0; **MaxLogFileSize**=0 Filename #1 is created and roll over 1st file in case of restart. **Scenario 3:** **NumberOfLogFiles** > 0; **MaxLogFileSize** > 0 Log will be rolled for size of file and for number of files. **Scenario 4:** **NumberOfLogFiles**=0; **MaxLogFileSize** > 0 A file with unique pid name is created up to the size of file and then stop logging.
MaxLogFileSize	UInt64	0	Specifies the default behavior of log file size limit (used when `LoggerType value="File"`). Possible values are: • 0: A log file grows indefinitely. Log file size limit is disabled. • 1..4294967296: A log file grows till "**MaxLogFileSize**". When file size reaches the value of "**MaxLogFileSize**" it is closed and a new file with the next sequential number is opened. See "**NumberOfLogFiles**".
LoggerSeverity	Enumeration	Success	Sets the level at which the Enterprise Message API logs events. 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 4.6). Possible values are: • LoggerSeverity::Verbose (0) • LoggerSeverity::Success (1) • LoggerSeverity::Warning (2) • LoggerSeverity::Error (3) • LoggerSeverity::NoLogMsg (4)

Table 21: Logger Group Parameters (Continued)

PARAMETER NAME	TYPE	DEFAULT	NOTES
LoggerType	Enumeration	File	Specifies the logging mechanism. Use enumeration values with the Enterprise Message API's programmatic configuration (for details, refer to Section 4.6). Possible values are: • **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 21: Logger Group Parameters (Continued)

3.10   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.

3.10.1   Generic XML Schema for DictionaryGroup

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

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

3.10.2   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 4.6). Possible values are: • **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	./enumtype.def	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	./RDMFieldDictionary	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 22: Dictionary Group Parameters

3.11   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**).

3.11.1   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>
```

3.11.2   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"/>
```

3.11.3   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 23: Directory Entry Parameters

3.11.4   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 24: Service Entry Parameters

3.11.5   InfoFilter Entry Parameters

The Enterprise Message API uses the following **InfoFilter** parameters to set filters on the types of information it sends overs 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.
IsSource	UInt64	0	Specifies whether the source of data sent on this service is its original publisher: • **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 3.11.5.1.
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 3.11.5.2.

Table 25: Source Directory Info Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
AcceptingConsumerStatus	UInt64	1	Indicates whether a service can accept and process messages related to Source Mirroring. • 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 3.11.5.3.
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 3.11.5.4.
SupportsQoSRange	UInt64	0	Indicates whether the provider supports a QoS range when requesting an item. • 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 *RDM C++ Edition Usage Guide*.
SupportsOutOfBandSnapshots	UInt64	For non-interactive provider: 0	Indicates whether the provider supports Snapshot requests after the OpenLimit has been reached: • 0: The provider does not support snapshot requests. • 1: The providers supports snapshot requests. For details on **OpenLimit**, refer to the *RDM C++ Edition Usage Guide*.

Table 25: Source Directory Info Parameters (Continued)

3.11.5.1   Capabilities Entry 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 26: CapabilitiesEntry Parameter

3.11.5.2   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 **RWFEnum** 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 27: DictionariesProvided Parameter

3.11.5.3   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 RWFEnum 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 28: DictionariesUsedEntry Parameter

3.11.5.4   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 29: QoSEntry Section and Associated Parameters

3.11.6   StateFilter Entry Parameters

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

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ServiceState	UInt64	N/A	Specifies whether the service is up or down: • 0: Service is down • 1: Service is up
AcceptingRequests	UInt64	For non-interactive provider: 0	Specifies whether the service accepts request messages: • 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 30: StateFilter Parameters

3.11.7   LoadFilter Entry Parameters

Use the following parameters to configure the **Service**'s **LoadFilter** (as specified in **EmaConfig.xml**) which communicates the service 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 a time.
LoadFactor	UInt64	1	Indicates the current workload on the source providing data.

Table 31: LoadFilter Entry Parameters

3.11.8   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 32: Status Entry Parameters

3.11.9   Setting Directory with Multiple Dictionaries Provided for IProvider

The following **EmaConfig.xml** example shows how to set up a Directory for the interactive provider. Note that the **DictionariesProvided** section of the Directory contains two **DictionariesProvided** entries. This feature allows the provider to supply multiple dictionaries to the consumer.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<EmaConfig>

<IProviderGroup>
    <DefaultIProvider value="Provider_1"/>
    <IProviderList>
      <IProvider>
          <Name value="Provider_1"/>
          <Server value="Server_1"/>
          <Directory value="Directory_1"/>
      </IProvider>
    </IProviderList>
</IProviderGroup>

<ServerGroup>
    <ServerList>
      <Server>
          <Name value="Server_1"/>
          <ServerType value="ServerType::RSSL_SOCKET"/>
          <CompressionType value="CompressionType::None"/>
          <Port value="14002"/>
      </Server>
  </ServerList>
</ServerGroup>

<DirectoryGroup>
   <DefaultDirectory value="Directory_1"/>
   <DirectoryList>
      <Directory>
          <Name value="Directory_1"/>
          <Service>
             <Name value="DIRECT_FEED"/>
             <InfoFilter>
                <ServiceId value="1"/>
                <Vendor value="TestLab"/>
                <DictionariesProvided>
                   <DictionariesProvidedEntry value="Dictionary_1"/>
                   <DictionariesProvidedEntry value="Dictionary_2"/>
                </DictionariesProvided>
                <DictionariesUsed>
                   <DictionariesUsedEntry value="Dictionary_1"/>
                </DictionariesUsed>
                <Vendor value="company name"/>
                <IsSource value="0"/>
                <Capabilities>
```

```
                    <CapabilitiesEntry value="MMT_MARKET_BY_ORDER"/>
                    <CapabilitiesEntry value="MMT_DICTIONARY"/>
                </Capabilities>
                <QoS>
                    <QoSEntry>
                        <Timeliness value="Timeliness::RealTime"/>
                        <Rate value="Rate::TickByTick"/>
                    </QoSEntry>
                </QoS>
                <ItemList value="#.itemlist"/>
                <SupportsOutOfBandSnapshots value="0"/>
            </InfoFilter>
            <StateFilter>
                <ServiceState value="1"/>
                <AcceptingRequests value="1"/>
            </StateFilter>
        </Service>
    </Directory>
    </DirectoryList>
</DirectoryGroup>

<DictionaryGroup>
    <DictionaryList>
        <Dictionary>
            <Name value="Dictionary_1"/>
            <DictionaryType value="DictionaryType::FileDictionary"/>
            <RdmFieldDictionaryFileName value="./RDMFieldDictionary"/>
            <EnumTypeDefFileName value="./enumtype.def"/>
        </Dictionary>
        <Dictionary>
            <Name value="Dictionary_2"/>
            <DictionaryType value="DictionaryType::FileDictionary"/>
            <RdmFieldDictionaryFileName value="./RDMFieldDictionary_ID2"/>
            <EnumTypeDefFileName value="./enumtype_ID2.def"/>
            <RdmFieldDictionaryItemName value="RWFFld_ID2"/>
            <EnumTypeDefItemName value="RWFEnum_ID2"/>
        </Dictionary>
    </DictionaryList>
</DictionaryGroup>

</EmaConfig>
```

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.

4.2.2   Default *Provider* Configurations

Each Enterprise Message API provider-type application must eventually instantiate an *OmmProvider* object. Constructors for `OmmProvider` require a **Omm*Provider*Config** 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 LSEG 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**, 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.

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, refer to the example *100__MarketPrice__Streaming* included with the LSEG 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 *Enterprise Message API LSEG Domain Model (RDM) Usage Guide* specific to the programming language you use to override the default administrative request. Application may call multiple times prior to initialization.
addLoginMsgCredential(const ReqMsg& reqMsg, const EmaString& channelList) addLoginMsgCredential(const ReqMsg& reqMsg, const EmaString& channelList, const OmmLoginCredentialConsumerClient& client) addLoginMsgCredential(const ReqMsg& reqMsg, const EmaString& channelList, const OmmLoginCredentialConsumerClient& client, void* closure)	Allows the user to configure multiple login credentials for an OmmConsumerConfig's channels. Sets a login request message to be used with the channels defined in the `channelList`. `channelList` is a comma separated string of channel names defined in the Channel Group (see Section 3.4). All login request messages for any given OMMConsumer configuration need to have the same SingleOpen and Allow Suspect Data Verification options set. If a channel is not specified through this configuration, a login request message that has been set via `AddAdminMsg` will be used. If no message has been set through `AddAdminMsg,` the default Enterprise Message API login request will be used. If the `OmmLoginCredentialConsumerClient` is set, the Enterprise Message API will call back the user on reconnection to update the UserName or Authentication Extended members of the login request. This will not be called if the channel has session management enabled. If specified, the closure pointer will be returned to the user in the `OmmLoginCredentialConsumerClient::onLoginCredentialRenewal` event callback.
addOAuth2Credential(const OAuth2Credential& credential) addOAuth2Credential(const OAuth2Credential& credential, const OmmOAuth2ConsumerClient& client) addOAuth2Credential(const OAuth2Credential& credential, const OmmOAuth2ConsumerClient& client, void* closure);	Allows the user to configure multiple oAuth credentials for an OmmConsumerConfig's channels. Sets oAuth credentials to be used with channels that have `EnableSessionManagement` turned on. The OAuth2Credential object allows the user to specify a comma separated string of channel names defined in the Channel Group (see Section 3.4). If a set of oAuth credentials is defined through the username, password, clientId, and clientSecret, then these will be used if a configured channel is not explicitly referenced. If specified, the closure pointer will be returned to the user in the `OmmOAuth2ConsumerClient::onCredentialRenewal` event callback. For more information about the oAuth functionality, refer to the *Enterprise Message API Developers Guide*.

Table 33: `OmmConsumerConfig` **Class Function Calls**

FUNCTION	DESCRIPTION
addServiceList(const ServiceList&)	Adds the indicated Service List to **OmmConsumerConfig**. The Service List name can be specified in **reqMsg.ServiceListName** to indicate that the request is for any service in the Service List. For more information, see section "Service List" in chapter "Request Routing" in the *Enterprise Message API Developers Guide*.
applicationId(const EmaString&)	Sets the **applicationId** variable. **applicationId** has no default value.
audience(String)	Optional. Used only with Version 2 oAuthClientCredential with JWT. Sets the audience claim for the JWT.
channelType(ConnectionTypes)	Optional. Specifies the channel type used by the current consumer. Use **EmaConfig::ConnectionTypeEnum** to set allowed connection type.
clear()	Clears existing content from the **OmmConsumerConfig** object.
clientId(const EmaString&)	**Required**. Specifies an authentication parameter. • Version 1 Authentication: a unique ID for application used to make an authentication request. • Version 2 Authentication: a unique ID provisioned as part of Service Account used to make an authentication request. For details on Version 1 and Version 2 Authentication, refer to the *Enterprise Message API Developers Guide*, Section "Consuming Data from the Cloud".
clientJwk(String)	**Required** for Version 2 oAuthClientCredential with JWT. Sets the JWK formatted private key used to create the JWT. The JWT is used to authenticate with the RDP token service.
clientSecret(const EmaString&)	Sets the client secret. Optional for Version 1 oAuthPasswordGrant authentication. **Required** for Version 2 oAuthClientCredential authentication and provisioned as part of Service Account.
config(const Data&)	Passes in the consumer's programmatic configuration.
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.
dataDictionary(const DataDictionary&, bool shouldCopyIntoAPI)	Optional. Specifies the DataDictionary object with a mandatory bool or flag. If flag is true, the DataDictionary object will be copied into API space; otherwise, it will be passed in as a reference. Overrides DataDictionary object provided via **EmaConfig.xml** or programmatic configuration.
encryptedProtocolType(EncryptedProtocol Types)	Optional. Specifies the encrypted protocol type used by the current consumer. Use **EmaConfig::EncryptedProtocolTypeEnum** to set allowed encrypted protocol type.
host(const EmaString& host)	Sets the host and port parameters. For details, refer to Section 4.4.2. Sample value: "localhost:14002".
libcurlName(const EmaString&)	Optional. Specifies the location of **libcurl**, which is used by EMA for handling any RDP interactions or proxy connections.
negativeUpdateTypeFilter(const UInt64)	Specifies the types of update messages that the client should NOT receive from the server. The value is represented as a bitmask, with each bit indicating a specific update type to exclude. Possible bit values are defined in the **EmaRdm.h** header file.

Table 33: **OmmConsumerConfig** **Class Function Calls(Continued)**

FUNCTION	DESCRIPTION
operationModel(OperationModel)	Optional. Sets the operation model to either of these: • **OperationModel.API_DISPATCH** (default) • **OperationModel.USER_DISPATCH**
password(const EmaString&)	**Required** for Version 1 oAuthPasswordGrant authentication. Specifies the password used together with the username to obtain the access token.
position(const EmaString&)	Sets the **position** variable. **position** has no default value.
proxyDomain(const EmaString&)	Optional. Specifies the domain for an NTLM or Kerberos authenticated proxy.
proxyPasswd(const EmaString&)	Optional. Sets password for proxy authentication.
proxyUserName(const EmaString&)	Optional. Sets username for proxy authentication.
restLoggingCallback (OmmRestLoggingClient&)	Optional. When this callback is set or specified in OmmConsumer client config (defined as **OmmRestLoggingClient**), REST message logs (if enabled) may be received by EMA application via **onRestLoggingEvent**.
restProxyDomain(const EmaString&)	Optional. Specifies the domain of the user to authenticate to the proxy server for Rest requests: service discovery and auth token service. Needed for NTLM or for Negotiate/Kerberos or for Kerberos authentication protocols. **restProxyDomain** should be the same as the domain in the "realms" and "domain_realm" sections of the Kerberos configuration file.
restProxyHostName(const EmaString&)	Optional. Specifies the address or host name of the proxy server for REST requests: service discovery and auth token service.
restProxyPasswd(const EmaString&)	Optional. Specifies the password to authenticate to the proxy server for REST requests: service discovery and auth token service. Needed for all authentication protocols.
restProxyPort(const EmaString&)	Optional. Specifies the port number of the proxy server for REST requests: service discovery and auth token service.
restProxyUserName(const EmaString&)	Optional. Specifies the user name to authenticate to the proxy server for REST requests: service discovery and auth token service. Needed for all authentication protocols.
ServiceDiscoveryUrl(const EmaString&)	Optional. Specifies a URL to override the default for the RDP service discovery to get global endpoints. Default value is **https://api.refinitiv.com/streaming/pricing/v1/**.
sslCAStore(const EmaString&)	Optional. Specifies the location (either directory or file path) of the Certificate Authority store. This is optional. By default, EMA will load the system's default Certificate Authority store.
takeExclusiveSignOnControl(bool)	Optional. Specifies exclusive sign on control to force sign-out of other applications using the same credentials. Default value is **True**.
tokenScope(const EmaString&)	Optional for Version 1 and Version 2 authentication. Specifies token scope to override the default for limiting the scope of generated token from the token service. Defaults to **trapi.streaming.pricing.read**.
tokenServiceUrlV1(const EmaString&)	Optional. Specifies a URL to override the default for token service V1 oAuthPasswordGrant to perform authentication to get access and refresh tokens. Default value is **https://api.refinitiv.com/auth/oauth2/v1/token**.
tokenServiceUrlV2(const EmaString&)	Optional. Specifies a URL to override the default for token service V2 oAuthClientCredentials to perform authentication to get access and refresh tokens. Default value is **https://api.refinitiv.com/auth/oauth2/v2/token**.

Table 33: **OmmConsumerConfig** Class Function Calls(Continued)

FUNCTION	DESCRIPTION
tunnelingLibCryptoName(const EmaString&)	Optional. Specifies the name of the **libcrypto.so** shared library for Encrypted connections. See *Enterprise Message API C++ Edition Developers Guide* for details on default libcrypto names per platform and OpenSSL version.
tunnelingLibSslName(const EmaString&)	Optional. Specifies the name of the **libssl.so** shared library for Encrypted connections. See the *Enterprise Message API C++ Edition Developers Guide* for details on default libssl names per platform and OpenSSL version.
tunnelingProxyHostName(const EmaString&)	Optional. Specifies the host name of an HTTP Proxy for any Socket, Websocket, or Encrypted connections.
tunnelingProxyPort(const EmaString&)	Optional. Specifies the port number of the proxy server to connect to for an HTTP connection.
tunnelingSecurityProtocol(int)	Optional. Specifies the cryptographic protocols to be used for an Encrypted connection. The highest value of TLS will be selected by the EMA API first, then it will roll back if the encryption handshake fails. TLS 1.2 and TLS 1.3 are supported.
tunnelingSecurityProtocol(int securityProtocol)	Specifies the cryptographic protocols to be used for an Encrypted connection on a Linux operating system. The protocol supports TLS v1.2 and TLS v1.3. Use **OmmConsumerConfig::EncryptedProtocolTypes** flags to set allowed protocols. **@param[in]** securityProtocol specifies a cryptopgraphic protocol. **@return** reference to this object.
updateTypeFilter(const UInt64)	Specifies the type of update messages that the client is expected to receive from the server. The value is represented as a bitmask, where each bit corresponds to a specific update type. Possible bit values are defined in the **EmaRdm.h** header file.
username(const EmaString&)	**Required** for Version 1 oAuthPasswordGrant authentication. Specifies the username used to obtain the authentication token from the Delivery Platform and provisioned with Machine Account. Optional otherwise. Sets username; if not specified, username is extracted from run-time environment.

Table 33: `OmmConsumerConfig` Class Function Calls(Continued)

4.4.1.2   OMMIProviderConfig Class Function Calls

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

FUNCTION	DESCRIPTION
addAdminMsg(RefreshMsg)	Optional. Populates the entirety of the initial directory refresh message according to the specification discussed in the *Enterprise Message API C++ Edition RDM Usage Guide*. Supports Directory domain only.
adminControlDictionary(AdminControl)	Optional. Specifies whether API or user controls responding to Dictionary requests. Default is **ApiControlEnum**.
adminControlDirectory(AdminControl)	Specifies whether the API or the user controls the sending of Directory refresh messages. Default is **ApiControlEnum**.

Table 34: `OmmIProviderConfig` Class Function Calls

FUNCTION	DESCRIPTION
apiThreadBind(const EmaString& cpuString)	Optional. Specifies the CPU core to bind for internal EMA thread that dispatches messages. Application may call multiple times prior to initialization. Specify the CPU core in string format (CPU core ID or "P:X C:Y T:Z" format). Works in conjunction with **shouldInitializeCPUIDlib**.
cipherSuite	Optional. Specifies the cipher suites used by the provider. This is an OpenSSL formatted string.
clear()	Clears existing content from the `OmmIProviderConfig` object.
config(const Data&)	Passes in the provider's programmatic configuration.
dhParams(const EmaString&)	Optional. Specifies an optional DH Parameter file to be used with the Provider.
getProviderRole()	Retrieves Provider's role.
libCryptoName(const EmaString&)	Optional. Specifies the name of the **libcrypto** library for encrypted connections. See the *Enterprise Message API C++ Edition Developers Guide* for more details on default library name specific to each OpenSSL version and platform.
libCurlName (cons EmaString&)	Optional. Specifies the name of the **libcurl** library. This is not used for server-side connections, but this is required on first initialization for both REST and proxy connections.
libSslName (const EmaString&)	Optional. Specifies the name of the **libSsl** shared library for encrypted connections. See the *Enterprise Message API C++ Edition Developers Guide* for more details on default library names specific to each OpenSSL version and platform.
operationModel(OperationModel)	Optional. Specifies the operation model. Default is **API Dispatch: ApiDispatchEnum**.
port(const EmaSttring &)	Optional. Specifies a port. Default is **14002**.
providerName(const EmaString &)	This name identifies configuration section to be used by an **OmmIProvider** instance.
serverCert(const EmaString&)	Specifies the location of the server certificate file for encrypted providers.
serverPrivateKey(const EmaString&)	Specifies the location of the private key file for encrypted providers.

Table 34: `OmmIProviderConfig` Class Function Calls (Continued)

4.4.1.3   OMMNIProviderConfig Class Function Calls

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

FUNCTION	DESCRIPTION
addAdminMsg(const RefreshMsg&)	Optional. Specifies an administrative refresh message to override the default administrative refresh. Supports Directory domain only.
addAdminMsg(RequestMsg)	Used only with **NiProvider**. Optional. Populates part of or all of the login request message according to the specification discussed in the *Enterprise Message API C++ Edition RDM Usage Guide*. Supports Login domain only.

Table 35: `OmmNiProviderConfig` Class Function Calls

FUNCTION	DESCRIPTION
adminControlDirectory(AdminControl)	Optional. Specifies whether the API or the user controls the sending of Directory refresh messages. Available values include: • **OmmNiProviderConfig::ApiControlEnum** (which is the default) • **OmmNiProviderConfig::UserControlEnum** For details on control models, refer to **OmmNiProviderConfig.h**.
apiThreadBind(const EmaString&)	Optional. Specifies the CPU core to bind for internal EMA thread that dispatch messages. Application may call multiple times prior to initialization. Specify the CPU core in string format (CPU core ID or "P:X C:Y T:Z" format). Works in conjunction with **shouldInitializeCPUIDlib**.
applicationId(const EmaString&)	Optional. Specifies the authorization application identifier set in login request attribute. Must be unique for each application. No default value. Range 257 to 65535 is available for site-specific use. Range 1 to 256 is reserved.
clear()	Clears existing content from the **OmmProviderConfig** object.
channelType(ConnectionTypes)	Optional. Specifies the channel type used by the current provider. Use **EmaConfig::ConnectionTypeEnum** to set allowed connection type.
config(const Data&)	Passes in the NiProvider's programmatic configuration.
encryptedProtocolType(EncryptedProtocol Types)	Optional. Specifies the encrypted protocol type used by the current provider. Use **EmaConfig::EncryptedProtocolTypeEnum** to set allowed encrypted protocol type.
getProviderRole()	Retrieves NiProvider's role.
host(const EmaString&)	Optional. Specifies a hostname and port. By default, it is set to **localhost:14003**.
instanceId(const EmaString&)	Optional. Specifies the instance identifier. Can be any ASCII string, e.g. "Instance1".
libcurlName(const EmaString&)	Optional. Specifies the name of the **libcurl** library for connecting through HTTP proxies. This is supported on Socket connections and Encrypted connections with Socket encrypted protocol.
operationModel(OperationModel)	Optional. Specifies the operation model. Default is **API Dipatch: ApiDispatchEnum**.
password(const EmaString&)	Specifies password. Overrides a value specified in Login domain via the **addAdminMsg(const ReqMsg&)** method.
position(const EmaString&)	Specifies position in login request attribute.
providerName(const EmaString&)	This name identifies configuration section to be used by **OmmNiProvider** instance.
proxyDomain(const EmaString&)	Optional. Specifies the domain of the user to authenticate. Needed for NTLM or for Negotiate/Kerberos or for Kerberos authentication protocols. For Negotiate/Kerberos or for Kerberos authentication protocols, **proxyDomain** should be the same as the domain in the 'realms' and 'domain_realm' sections of the Kerberos configuration file.
proxyPasswd(const EmaString&)	Optional. Specifies the password to authenticate. Needed for all authentication protocols.
proxyUserName(const EmaString&)	Optional. Specifies the user name to authenticate. Needed for all authentication protocols.
shouldInitializeCPUIDlib(bool)	Optional. Must be set to **True** for thread binding to occur. Lets Enterprise Transport API initialize CpuID library for thread binding to take effect.
sslCAStore(const EmaString&)	Optional. Specifies the path to an OpenSSL Certificate Authority store.

Table 35: `OmmNiProviderConfig` **Class Function Calls**

FUNCTION	DESCRIPTION
tunnelingLibCryptoName(const EmaString&)	Optional. Specifies the name of the **libcrypto** library for Encrypted connections. See the *Enterprise Message API C++ Edition Developers Guide* for more details on default library name specific to each OpenSSL version and platform.
tunnelingLibSslName(const EmaString&)	Optional. Specifies the name of the **libssl** library for Encrypted connections. See the *Enterprise Message API C++ Edition Developers Guide* for more details on default library name specific to each OpenSSL version and platform.
tunnelingObjectName(const EmaString&)	Specifies the object name to pass along with the underlying URL in HTTP and HTTPS connection messages.
tunnelingProxyHostName(const EmaString&)	Optional. Specifies the address or host name of the proxy server to connect to for an HTTP or HTTPS connection.
tunnelingProxyPort(const EmaString&)	Optional. Specifies the port number of the proxy server to connect to for an HTTP or HTTPS connection.
tunnelingSecurityProtocol(int)	Optional. Specifies the cryptographic protocols to be used for an Encrypted connection. The highest TLS version supported by API will be selected by the Rssl API first, then it will roll back if the encryption handshake fails. The protocol supports TLS v1.2 and TLS v.1.3. Use **OmmNiProviderConfig::EncryptedProtocolTypes** flags to set allowed protocols.
username(const EmaString&)	Specifies the name used in the login request. Overrides a value specified in the Login domain via the **addAdminMsg(const ReqMsg&)** method.
workerThreadBind(const EmaString&)	Optional. Specifies the CPU core to bind for Reactor Worker thread. Application may call multiple times prior to initialization. Specify the CPU core in string format (CPU core ID or "P:X C:Y T:Z" format). Works in conjunction with **shouldInitializeCPUIDlib**.

Table 35: `OmmNiProviderConfig` **Class Function Calls**

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 3.4.2) 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*.

4.4.3   Service Discovery Configuration Using Function Calls

4.4.3.1   ServiceEndpointDiscovery

ServiceEndpointDiscovery class provides the functionality to query endpoints from RDP service discovery.

The application interacts with service discovery through the ServiceEndpointDiscovery interface methods.

The results of these interactions are communicated back to application through ServiceEndpointDiscoveryClient.

Refer to the reference manual manual for more details on ServiceEndpointDiscovery and the classes it uses for functionality.

4.4.3.1.1   ServiceEndpointDiscoveryConfig

When creating a ServiceEndpointDiscovery object, configuration options can be set programmatically by passing a ServiceEndpointDiscoveryConfig object into the constructor.

The following table provides a list of configurations that can be specified in a ServiceEndpointDiscoveryConfig object.

FUNCTION	DESCRIPTION
libCurlName(const EmaString&)	Optional. Specifies a name to override the default for the **curl** shared library.
libCryptoName(const EmaString&)	Optional. Specifies a name to override the default for the OpenSSL **libcrypto** shared library.
libSslName(const EmaString&)	Optional. Specifies a name to override the default for the OpenSSL **libssl** shared library.
restEnableLog(bool)	Optional. Specifies whether to enable or disable REST logging. If set to true, REST logging is enabled. Defaults to false.
restLogOutputStream(FILE*)	Optional. Specifies file output location for REST Logs. If NULL, sends logs to stdout.
restVerboseMode(bool)	Optional. Specifies whether to enable or disable verbose REST logging. Defaults to false.
shouldInitializeCPUIDlib(bool)	Optional. Specifies whether to enable or disable the CpuId library initialization. Defaults to true.
serviceDiscoveryUrl(const EmaString&)	Optional. Specifies a URL to override the default for the RDP service discovery to get global endpoints
tokenServiceUrlV1(const EmaString&)	Optional. Specifies a URL to override the default for token service V1 Password Credentials to perform authentication to get access and refresh tokens.
tokenServiceUrlV2(const EmaString&)	Optional. Specifies a URL to override the default for token service V2 Password Credentials to perform authentication to get access and refresh tokens.

Table 36: ServiceEndpointDiscoveryConfig Functions

4.4.4   Multi-Credential Configuration

OmmConsumers can have associate login request messages and/or oAuth credentials with channels configured in the **ChannelSet** or **Warm Standby** configuration. Using the **addOAuth2Credential** and **addLoginMsgCredential** function calls allow for multiple credentials and login request options to be used with channels configured in the **ChannelGroup**.

oAuth2 credentials can be configured with the **OAuth2Credential** class.

Login credentials will be configured with a Login Domain **ReqMsg** and associated **channelList**. The **LoginReq** class can be used to assist with populating this **ReqMsg**.

For an example implementation of this functionality, see example Consumer 480_MP_MultiCredentials.

4.4.4.1 OAuth2Credential Class Functions

You can use the following class functions:

FUNCTION	DESCRIPTION
channelList(const EmaString&)	Comma separated list of channel names that should be used with these credentials. If this is set to a blank EMAString, these credentials will be used by any channels that have Session Management turned on and are not explicitly specified by other credentials **channelList**.
clear()	Zeroes out any sensitive data strings and clears out the members in the class.
clientId(const EmaString&)	Required. Specifies an authentication parameter. • Version 2 Authentication: a unique ID provisioned as part of Service Account used to make an authentication request. • Version 1 Authentication: a unique ID for application used to make an authentication request.
clientSecret(const EmaString&)	Sets the client secret. Optional for Version 1 oAuthPasswordGrant authentication. Required for Version 2 oAuthClientCredential authentication and provisioned as part of Service Account.
const EmaString& getChannelList()	Gets the channelList. This is optionally used for login V1 and login V2.
const EmaString& getClientId()	Gets the clientId. This is used for login V1 and login V2.
const EmaString& getClientSecret()	Gets the clientSecret. This is used for login V2.
const EmaString& getPassword()	Gets the password. This is only used for login V1.
const bool getTakeExclusiveSignOnControl()	Specifies the **takeExclusiveSignOnControl** feature. This is optionally used for login V1.
const EmaString& getTokenScope()	Gets the tokenScope. This is optionally used for login V1 and login V2.
const EmaString& getUserName()	Gets the user name. This is only used for login V1.
password(const EmaString&)	Required for Version 1 oAuthPasswordGrant authentication. Specifies the password used together with the username to obtain the access token.
tokenScope(const EmaString&)	Optional for Version 1 and Version 2 authentication. Specifies token scope to override the default for limiting the scope of generated token from the token service. Defaults to "trapi.streaming.pricing.read".
takeExclusiveSignOnControl(bool)	Optional. Only used for Version 1 authentication. Specifies exclusive sign-on control to force sign-out of other applications using the same credentials. Default value is "True".
userName(const EmaString&)	Required for Version 1 oAuthPasswordGrant authentication. Specifies the username used to obtain the authentication token from the Delivery Platform and provisioned with Machine Account.

Table 37: `OAuth2Credential` Class Functions

4.5   Helper Function Calls

4.5.1   OmmConsumer Helper Function Calls

When the application directly calls **OmmConsumer::fallbackPreferredHost()**, the library will immediately start a Preferred Host operation. This will return an error if Preferred Host is not enabled on the channel. If a Preferred Host operation is currently in progress, this will immediately return success. After completion of shifting to preferred host, the Channel/WarmStandby Group library sends Login StatusMsg with dataState ==RSSL_DATA_OK, streamState == RSSL_STREAM_OPEN, text == "Preferred host complete" and code == OmmState.SocketPHComplete.

4.5.1.0.1   FallBack to Preferred Host Function Call

EMA Application may trigger a fallback to preferred host in a ChannelSet configuration or fallback to a preferred group in WarmStandby configuration.

4.5.1.0.2   *Method Call*

When the application directly calls OmmConsumer::fallbackPreferredHost(), the library will immediately start a Preferred Host operation. This will return an error if Preferred Host is not enabled on the channel. If a Preferred Host operation is currently in progress, this will immediately return success. After completion of shifting to preferred host Channel/Warm Standby Group library sends Login StatusMsg with dataState ==RSSL_DATA_OK, streamState == RSSL_STREAM_OPEN, text == "Preferred host complete" and code == OmmState.SocketPHComplete.

4.5.1.1   Dynamically Set Configuration Using IOCtl: Preferred Host Options

All Preferred Host options can be modified during runtime via **OmmConsumer::modifyReactorChannelIOCtl()** calls. **PreferredHostOptions** class must be used to alter Preferred Host configuration. If IOCtl method is called while a preferred host operation is in progress, the change will be applied after the operation completes. For details on each of the following preferred host parameters, refer to Section 3.1.4.

METHOD	DESCRIPTION
enablePreferredHostOptions(bool)	Specifies the value of Preferred Host parameter **EnablePreferredHostOptions**.
phDetectionTimeSchedule(String)	Specifies the value of Preferred Host parameter **PHDetectionTimeSchedule**.
phDetectionTimeInterval(int)	Specifies the value of Preferred Host parameter **PHDetectionTimeInterval**.
preferredChannelName(String)	Specifies the value of Preferred Host parameter **PreferredChannelName**.
preferredWSBChannelName(String)	Specifies name of WarmStandby Group to set as preferred WarmStandby group.
fallBackWithInWSBGroup(bool)	Specifies value of Preferred Host parameter **PHFallBackWithInWSBGroup**.

Table 38: Preferred Host Options

4.5.1.2   GetChannelInfo Function Call: Preferred Host Parameters

EMA Application may request information regarding configured preferred host parameters. The following table describes the values contained in the **PreferredHostInfo** class. This information is available through the **OmmConsumer.getChannelInfo().getPreferredHostInfo()** method and is returned as a part of the **ChannelInformation** structure.

METHOD	DESCRIPTION
getEnablePreferredHostOptions()	Return the value of Preferred Host parameter **EnablePreferredHostOptions**.
getPHDetectionTimeSchedule()	Return the value of Preferred Host parameter **PHDetectionTimeSchedule**.
getPHDetectionTimeInterval()	Return the value of Preferred Host parameter **PHDetectionTimeInterval**.
getPreferredChannelName()	Return the value of Preferred Host parameter **PreferredChannelName**.
getPreferredWSBChannelName()	Return the value of Preferred Host parameter **PreferredWSBChannelName**.

Table 39: Preferred Host Parameters

METHOD	DESCRIPTION
getPHFallBackWithInWSBGroup()	Return value of Preferred Host parameter **PHFallBackWithInWSBGroup**.
getRemainingDetectionTime()	Return the remaining time, in seconds, until the library will perform the next Preferred Host operation.

Table 39: Preferred Host Parameters

4.6   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.

4.6.1   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:
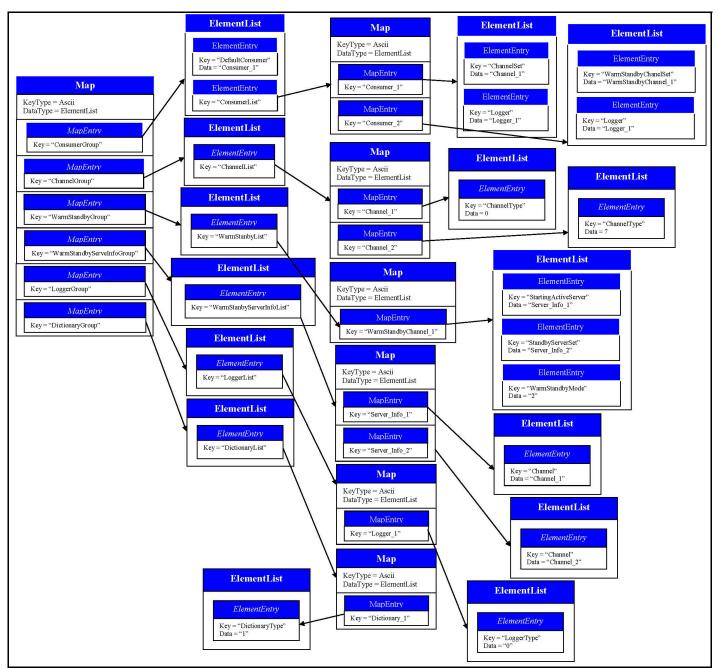
- The 1st tier lists the following Enterprise Message API's components each of which has its own list in the 2nd tier:
  - Consumer
  - IProvider
  - NiProvider
  - Channel
  - WarmStandby
  - WarmStandbyServerInfo
  - SessionChannel
  - Server
  - Logger
  - Dictionary
  - Directory

- 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.

4.6.2   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:

Map
KeyType = Ascii
DataType = ElementList
- MapEntry — Key = "ConsumerGroup"
- MapEntry — Key = "ChannelGroup"
- MapEntry — Key = "WarmStandbyGroup"
- MapEntry — Key = "WarmStandbyServeInfoGroup"
- MapEntry — Key = "LoggerGroup"
- MapEntry — Key = "DictionaryGroup"

ElementList
- ElementEntry — Key = "DefaultConsumer" Data = "Consumer_1"
- ElementEntry — Key = "ConsumerList"

ElementList
- ElementEntry — Key = "ChannelList"

ElementList
- ElementEntry — Key = "WarmStanbyList"

ElementList
- ElementEntry — Key = "WarmStanbyServerInfoList"

ElementList
- ElementEntry — Key = "LoggerList"

ElementList
- ElementEntry — Key = "DictionaryList"

Map
KeyType = Ascii
DataType = ElementList
- MapEntry — Key = "Consumer_1"
- MapEntry — Key = "Consumer_2"

Map
KeyType = Ascii
DataType = ElementList
- MapEntry — Key = "Channel_1"
- MapEntry — Key = "Channel_2"

Map
KeyType = Ascii
DataType = ElementList
- MapEntry — Key = "WarmStandbyChannel_1"

Map
KeyType = Ascii
DataType = ElementList
- MapEntry — Key = "Server_Info_1"
- MapEntry — Key = "Server_Info_2"

Map
KeyType = Ascii
DataType = ElementList
- MapEntry — Key = "Logger_1"

Map
KeyType = Ascii
DataType = ElementList
- MapEntry — Key = "Dictionary_1"

ElementList
- ElementEntry — Key = "ChannelSet" Data = "Channel_1"
- ElementEntry — Key = "Logger" Data = "Logger_1"

ElementList
- ElementEntry — Key = "WarmStandbyChanelSet" Data = "WarmStandbyChannel_1"
- ElementEntry — Key = "Logger" Data = "Logger_1"

ElementList
- ElementEntry — Key = "ChannelType" Data = 0

ElementList
- ElementEntry — Key = "ChannelType" Data = 7

ElementList
- ElementEntry — Key = "StartingActiveServer" Data = "Server_Info_1"
- ElementEntry — Key = "StandbyServerSet" Data = "Server_Info_2"
- ElementEntry — Key = "WarmStandbyMode" Data = "2"

ElementList
- ElementEntry — Key = "Channel" Data = "Channel_1"

ElementList
- ElementEntry — Key = "Channel" Data = "Channel_2"

ElementList
- ElementEntry — Key = "DictionaryType" Data = "1"

ElementList
- ElementEntry — Key = "LoggerType" Data = "0"

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`.

4.6.3   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 )
    .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)
```
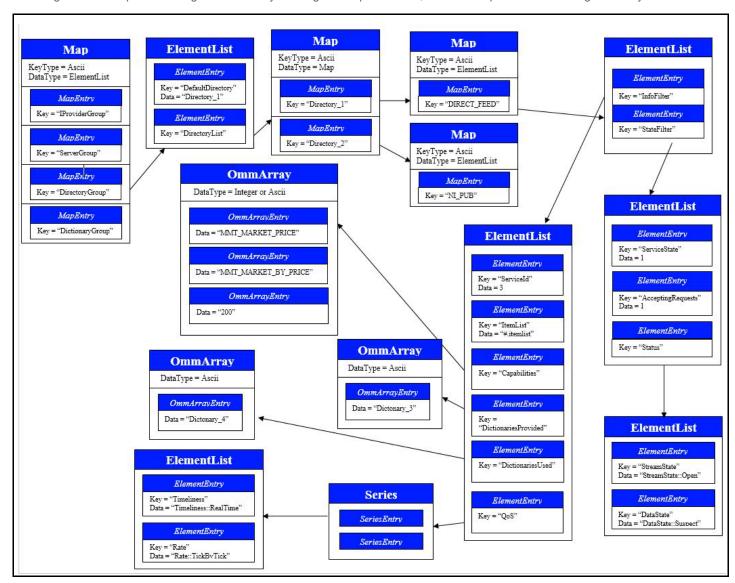
```
    .addUInt( "GuaranteedOutputBuffers", 5000 )
    .addUInt( "ConnectionPingTimeout", 50000 )
    .addAscii( "Host", "localhost" )
    .addAscii("Port", "14002" )
    .addUInt( "TcpNodelay", 0 ).complete() ).complete();

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();
```

4.6.4   Creating a Programmatic Configuration for an IProvider

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 IProvider:**

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 `OmmIProviderConfig` 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 an IProvider, or
   - A Dictionary Refresh message published by an IProvider

4.6.5   Example: Programmatic Configuration of an IProvider

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

```
Map outermostMap, innerMap;
ElementList elementList;

elementList.addAscii("DefaultIProvider", "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)
        .addUInt("RefreshFirstRequired", 1)
        .addUInt("AcceptDirMessageWithoutMinFilters", 0)
        .addUInt("AcceptMessageSameKeyButDiffStream", 0)
        .complete())
        .complete();

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

outermostMap.addKeyAscii("IProviderGroup", 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("TcpNodelay", 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 );
```

Document ID: EMAC390L3CG.250
Date of issue: June 2025

LSEG DATA & ANALYTICS