Enterprise Message API C# Edition 3.2.0.L1

ENTERPRISE MESSAGE API CONFIGURATION GUIDE

Document Version: 3.2.0
Date of issue: April 2024
Document ID: EMACSharp320CG.240

REFINITIV™

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 Enterprise Message API product and other Refinitiv products which leverage it.

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

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

Configuration works in the same fashion across all platforms.

1.2 Audience

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

1.3 Acronyms and Abbreviations

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

Table 1: Acronyms and Abbreviations

ACRONYM / TERM	MEANING
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	Refinitiv Domain Model
RDP	Refinitiv Data 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 Refinitiv Real-Time — Optimized endpoints using RDP.
Refinitiv Real-Time Distribution System	Refinitiv Real-Time Distribution System is Refinitiv's financial market data distribution platform. It consists of the Refinitiv Real-Time Advanced Distribution Server and Refinitiv Real-Time Advanced Data Hub. Applications written to the Refinitiv Real-Time SDK can connect to this distribution system.
Reactor	The Reactor is a low-level, open-source, easy-to-use layer above the Enterprise Transport API. It offers heartbeat management, connection and item recovery, and many other features to help simplify application code for users.
RMTES	A multi-lingual text encoding standard
RSSL	Refinitiv Source Sink Library
RTT	Round Trip Time, this definition is used for round trip latency monitoring feature.
RWF	Refinitiv Wire Format, a Refinitiv proprietary binary format for data representation.

Table 1: Acronyms and Abbreviations

1.4 References

- Enterprise Message API C# Edition *Refinitiv Domain Model Usage Guide*
- *API Concepts Guide*
- Enterprise Message API C# Edition *Developers Guide*
- The Refinitiv Developer Community

1.5 Documentation Feedback

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

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

1.6 Document Conventions

This document uses the following types of conventions:

- Typographic
- Field and Text Values
- Boolean Values

1.6.1 Typographic

This document uses the following types of conventions:

- 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***.

1.6.2 Field and Text Values

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

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

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

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

1.6.3 Boolean Values

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

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

2 Enterprise Message API Configuration Overview

2.1 About Message API Configuration

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

Message API configuration data is divided into the following groups:

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

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

This manual discusses the 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.

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 Consumer Group

A **ConsumerGroup** contains two elements:

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

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

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

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

3.1.1 Generic XML Schema for ConsumerGroup

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

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

3.1.2 Setting a Default Consumer

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

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

3.1.3 Configuring Consumers in a ConsumerGroup

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

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

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

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

3.1.4 Consumer Entry Parameters

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

PARAMETER	TYPE	DEFAULT	DESCRIPTION
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.
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 3.4.6 **NOTE:** If both **Channel** and **ChannelSet** are configured, then the Enterprise Message API uses the **<ChannelSet>** parameter.
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 3.7.
DictionaryRequestTimeOut	ulong	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.

Table 2: Consumer Group Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
DirectoryRequestTimeOut	ulong	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.
DispatchTimeoutApiThread	long	0	Specifies the duration (in microseconds) for which the internal Enterprise Message API thread is inactive before going active to check whether a message was received. If set to zero, the Enterprise Message API internal thread goes active only if it gets notified about a received message.
EnableRtt	ulong	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	uint	100,000	Specifies the number of items the application expects to request. If set to **0**, the Enterprise Message API resets it to **1024**. For better performance, the application can set this to the approximate number of item requests it expects.
LoginRequestTimeOut	ulong	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	uint	100	Specifies the maximum number of messages the Enterprise Message API dispatches before taking a real-time break.
MaxDispatchCountUserThread	uint	100	Specifies the maximum number of messages the Enterprise Message API can dispatch in a single call to the `OmmConsumer.Dispatch()`.
MaxOutstandingPosts	uint	100,000	Specifies the maximum allowable number of on-stream posts waiting for an acknowledgment before the **OmmConsumer** disconnects.

Table 2: Consumer Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
MsgKeyInUpdates	ulong	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**.
ObeyOpenWindow	ulong	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)
PostAckTimUInteout	uint	15,000	Specifies the length of time (in milliseconds) a stream waits to receive an ACK for an outstanding post before forwarding a negative acknowledgment to the application. If set to **0**, the Enterprise Message API will wait for a response indefinitely.
ReconnectAttemptLimit	Int	-1	Specifies the maximum number of times the consumer and non-interactive provider attempt to reconnect to a channel when it fails. If set to **-1**, the consumer and non-interactive provider continually attempt to reconnect.
ReconnectMaxDelay	Int	5000	Sets the maximum amount of time the consumer and non-interactive provider wait (in milliseconds) before attempting to reconnect a failed channel. Refer also to the **ReconnectMinDelay** parameter.
ReconnectMinDelay	Int	1000	Specifies the minimum amount of time the consumer and non-interactive provider wait (in milliseconds) before attempting to reconnect a failed channel. This wait time increases with each connection attempt, from **ReconnectMinDelay** to **ReconnectMaxDelay**.
RequestTimeout	uint	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.
RestProxyHostName	EmaString	N/A	Specifies the proxy host name or IP address to be used for REST requests. When this parameter and **RestProxyPort** are not specified, the **ProxyHost** from Channel group is used.
RestProxyPort	EmaString	N/A	Specifies the port of the proxy server to be used for REST requests. When this parameter and **RestProxyHostName** are not specified, **ProxyPort** from Channel group is used.
RestRequestTimeOut	uint	45000	Specifies the timeout (in milliseconds) for token service and service discovery request. If the request times out, the OMMConsumer resends the token reissue and the timeout restarts. If the request times out, the OMMConsumer does not retry. If set to **0**, there is no timeout.

Table 2: Consumer Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ServiceCountHint	int	513	Sets the size of directory structures for managing services. If the application specifies **0**, the Enterprise Message API resets it to **513**.
XmlTraceFileName	ascii	EmaTrace	Sets the name of the file to which to write XML trace output if tracing is selected.
XmlTraceMaxFileSize	ulong	100000000	Specifies the maximum size (in bytes) for the trace file.
XmlTracePing	ulong	1	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.
XmlTraceRead	ulong	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	ulong	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	ulong	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	ulong	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	ulong	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 2: 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 IProvider 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 channel schemas, you specify the desired channel schema by adding the following parameter inside the appropriate **IProvider** section:

```
<Logger value=”VALUE”/>
```

If your provider component needs more than one loggerchannel 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	UInt	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	UInt	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	UInt	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	UInt	0	Sets the **IProvider** to accept incoming request messages even though the source directory is not accepting requests.
AcceptMessageWithoutBeingLogin	UInt	0	Sets the **IProvider** to accept incoming request messages even though the interactive provider has not accepted a login request.
AcceptMessageWithoutQosInRange	UInt	0	Sets the **IProvider** to accept incoming request messages even though the requesting QoS is not in the QoS range of the source directory.
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 3.8.

Table 3: IProviderGroup Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
DispatchTimeoutApiThread	Int	0	Specifies the duration (in microseconds) for which the internal Enterprise Message API thread is inactive before going active to check whether a message was received. If set to zero, the thread goes active only if notified about a received message.
EnforceAckIDValidation	UInt	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	UInt	128000	Sets the maximum fragmentation size (in bytes) of enumerated types dictionary multi-part refresh messages.
FieldDictionaryFragmentSize	UInt	8192	Sets the maximum fragmentation size (in bytes) of field dictionary multi-part refresh messages.
ItemCountHint	UInt	100,000	Specifies the number of items the application expects to maintain. If set to **0**, the Enterprise Message API resets it to **1024**. For better performance, the application can set this to the approximate number of items it maintains.
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 3.6.
MaxDispatchCountApiThread	UInt	100	Specifies the maximum number of messages the Enterprise Message API dispatches before taking a real-time break.
MaxDispatchCountUserThread	UInt	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**.
RefreshFirstRequired	UInt	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 3: IProviderGroup Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
RequestTimeout	UInt	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.
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 3.5.
ServiceCountHint	UInt	513	Sets the size of directory structures for managing services. If the application specifies **0**, the Enterprise Message API resets it to **513**.
XmlTraceFileName	ascii	EmaTrace	Sets the name of the file to which to write XML trace output if tracing is selected.
XmlTraceMaxFileSize	ulong	100000000	Specifies the maximum size (in bytes) for the trace file.
XmlTracePing	ulong	1	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.
XmlTraceRead	ulong	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	ulong	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	ulong	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**.

Table 3: IProviderGroup Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
XmlTraceToStdout	ulong	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	ulong	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 3: 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 toSection 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 NiProvider Group

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 channel schemas, you specify the desired channel schema by adding the following parameter inside the appropriate **NiProvider** section:

```
<Channel 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
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.
ChannelSet	EmaString	N/A	**NOTE:** If both **<Channel>** and **<ChannelSet>** are configured, the Enterprise Message API uses the **<ChannelSet>** parameter.
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 3.8.

Table 4: NiProviderGroup Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
DispatchTimeoutApiThread	Int	0	Specifies the duration (in microseconds) for which the internal Enterprise Message API thread is inactive before going active to check whether a message was received. If set to zero, the thread goes active only if notified about a received message.
ItemCountHint	UInt	100,000	Specifies the number of items the application expects to maintain. If set to **0**, the Enterprise Message API resets it to **1024**. For better performance, the application can set this to the approximate number of items it maintains.
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 3.6.
LoginRequestTimeOut	UInt	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	UInt	100	Specifies the maximum number of messages the Enterprise Message API dispatches before taking a real-time break.
MaxDispatchCountUserThread	UInt	100	Specifies the maximum number of messages the Enterprise Message API can dispatch in a single call to the `OmmNiProvider::dispatch()`.
MergeSourceDirectoryStreams	UInt	1	Specifies whether the Enterprise Message API merges all source directory streams (configured and user-submitted) into one stream: • **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**.
ReconnectAttemptLimit	Int	-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	Int	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 4: NiProviderGroup Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ReconnectMinDelay	Int	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	Int	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	UInt	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	UInt	1	Specifies whether the Enterprise Message API removes items from its internal hash table whenever it disconnects from the Refinitiv Real-Time Advanced Data Hub. Possible values include: • **1** (true) • **0** (false)
RequestTimeout	UInt	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.
ServiceCountHint	UInt	513	Sets the size of directory structures for managing services. If the application specifies **0**, the Enterprise Message API resets it to **513**.
XmlTraceFileName	ascii	EmaTrace	Sets the name of the file to which to write XML trace output if tracing is selected.
XmlTraceMaxFileSize	ulong	100000000	Specifies the maximum size (in bytes) for the trace file.
XmlTracePing	ulong	1	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.
XmlTraceRead	ulong	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

Table 4: NiProviderGroup Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
XmlTraceToFile	ulong	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	ulong	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	ulong	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	ulong	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: 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 3.4.5.

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	String	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 strings with Enterprise Message API's programmatic configuration (for further details, refer to Section 4.5). Available values include: • **RSSL_SOCKET** • **RSSL_ENCRYPTED**
ConnectionPingTimeout	UInt	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	ulong	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	UInt	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	UInt	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	UInt	6144	Specifies the upper buffer-usage threshold for the channel. Must be set explicitly in either file or programmatic configuration.
InitializationTimeout	UInt	5 (10 when used with **RSSL_ENCRYPTED** ChannelType)	Specifies the time (in seconds) to wait for the successful initialization of a channel.

Table 5: Universal <Channel> 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.
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	UInt	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	UInt	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	UInt	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	UInt	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.

Table 5: 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_SOCKET"/>
<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
Unencrypted Socket	RSSL_SOCKET	Not used
Encrypted Socket	RSSL_ENCRYPTED	RSSL_SOCKET

Table 6: Channel Settings for Socket 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	UInt	30	Sets the message size threshold (in bytes, the allowed value is 30-Integer.MAX_VALUE), above which all messages are compressed (thus individual messages might not be compressed). Different compression types have different behaviors and compression efficiency can vary depending on message size.
CompressionType	String	None	Specifies the Enterprise Message API's preferred type of compression. Compression is negotiated between the client and server: if the server supports the preferred compression type, the server will compress data at that level. Use with Enterprise Message API's programmatic configuration (for further details, refer to Section 4.5). Available values include: • **None** • **ZLib** • **LZ4** **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.
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.

Table 7: Parameters for Channel Type: RSSL_SOCKET

PARAMETER NAME	TYPE	DEFAULT	NOTES
ProxyPort	EmaString	""	Specifies the port on the proxy to which the Enterprise Message API connects. Any value provided by a function call overrides the setting in configuration file.
TcpNodelay	UInt	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 7: Parameters for Channel Type: RSSL_SOCKET (Continued)

3.4.5 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), use the following parameters to configure a channel whose type is **ENCRYPTED**.

PARAMETER NAME	TYPE	DEFAULT	NOTES
AuthenticationTimeout	Uint		Specifies the timeout in millisecond for client to complete the authentication with the server. Defaults to 10 seconds.
EncryptedProtocolType	String	RSSL_SOCKET	Specifies the type of protocol used for this encrypted connection. • **RSSL_SOCKET** (0)
SecurityProtocol	Uint	4	Specifies the combination of flag values that set the version(s) of the TLS encryption protocol used for this connection: Valid values are TLS 1.2 and TLS 1.3; if unspecified, default is to support both.

Table 8: Parameters for Channel Types: ENCRYPTED

3.4.6 Example XML Schema for Configuring ChannelSet

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

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

3.4.7 Example ChannelSet XML Configuration

The following XML example illustrates a specific ChannelSet configuration using the XML schema introduced in Section 3.4.6:

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

3.4.8 Example Programmatic Configuration for ChannelSet

The following XML example illustrates a programmatic ChannelSet configuration in C#:

```
Map innerMap = new Map();
Map configMap = new Map();

ElementList elementList = new ElementList();
ElementList innerElementList = new ElementList();

elementList.AddAscii("DefaultConsumer", "Consumer_1");
innerElementList.AddAscii("ChannelSet", "Channel_1, Channel_2");
innerElementList.AddAscii"Dictionary", "Dictionary_1");
innerMap.AddKeyAscii( "Consumer_1", MapAction.ADD, innerElementList.Complete() );
innerElementList.Clear();
```

```
elementList.AddMap( "ConsumerList", innerMap ());
innerMap.Clear();

configMap.AddAscii( "ConsumerGroup", MapAction.ADD, elementList.Complete() );
elementList.Clear();

innerElementList.AddEnum("ChannelType", ConnectionTypeEnum.SOCKET);
innerElementList.AddAscii("Host", "localhost");
innerElementList.AddAscii("Port", "14002");
innerMap.AddKeyAscii( "Channel_1", MapAction.ADD, innerElementList.Complete() );
innerElementList.Clear();

innerElementList.AddEnum("ChannelType", ConnectionTypeEnum.SOCKET);
innerElementList.AddAscii("Host", "121.1.1.100");
innerElementList.AddAscii("Port", "14008"));
innerMap.AddKeyAscii("Channel_2", MapAction.ADD, innerElementList.Complete());
innerElementList.Clear();

elementList.AddMap( "ChannelList", innerMap.Complete() );
innerMap.Clear();

configMap.AddKeyAscii("ChannelGroup", MapAction.ADD, elementList.Complete());
elementList.Clear();

innerElementList.AddEnum("DictionaryType", DictionaryTypeEnum.FILE);
innerElementList.AddAscii("RdmFieldDictionaryFileName", "./RDMFieldDictionary");
innerElementList.AddAscii("EnumTypeDefFileName", "./enumtype.def");

innerMap.AddKeyAscii( "Dictionary_1", MapAction.ADD, innerElementList.Complete() );
innerElementList.Clear();

elementList.AddMap( "DictionaryList", innerMap.Complete() );
configMap.AddKeyAscii( "DictionaryGroup", MapAction.ADD,elementList.Complete() );
elementList.Clear();
configMap.Complete()
```

3.5 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 3.5.1.
- For a list of parameters you can use in configuring **Server**, refer to Section 3.5.2.

3.5.1 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>
```

3.5.2 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 3.5.3.

PARAMETER NAME	TYPE	DEFAULT	NOTES
ConnectionMinPingTimeout	UInt	20000	Configures the minimum length of time (in milliseconds) to use as a timeout for a connected channel.
ConnectionPingTimeout	UInt	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	UInt	30	Sets the message size threshold (in bytes, the allowed value is 30-Integer.MAX_VALUE), above which all messages are compressed (thus individual messages might not be compressed). Different compression types have different behaviors and compression efficiency can vary depending on message size.

Table 9: Universal <Server> Parameters

PARAMETER NAME	TYPE	DEFAULT	NOTES
CompressionType	String	None	Specifies the Enterprise Message API's preferred type of compression. Compression is negotiated between the client and server: if the server supports the preferred compression type, the server will compress data at that level. Use strings with Enterprise Message API's programmatic configuration. For further details, refer toSection 4.5. Available values include: • **None** • **ZLib** • **LZ4** **NOTE:** You can configure a server to force a particular compression type, regardless of client settings.
DirectWrite	ulong	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.
GuaranteedOutputBuffers	UInt	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	UInt	6144	Specifies the upper buffer-usage threshold for the channel. Must be set explicitly in either in file or programmatic configuration.
InitializationTimeout	UInt	60	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.
Name	EmaString		Specifies the **Server**'s name.
NumInputBuffers	UInt	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.

Table 9: Universal <Server> Parameters (Continued)

PARAMETER NAME	TYPE	DEFAULT	NOTES
ServerSharedSocket	UInt	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. For further details on **ServerSharedSocket**, refer to the *Transport API C# Edition Developers Guide*.
ServerType	String	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 strings with Enterprise Message API's programmatic configuration. For further details, refer to Section 4.5. Available values include **RSSL_SOCKET (0)**, **RSSL_ENCRYPTED (1)**. **RSSL_ENCRYPTED** requires additional configuration provided via **OmmIProviderConfig** and programmatic configuration. For details, refer to Section 4.4.1.
SysRecvBufSize	UInt	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	UInt	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	UInt	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: Universal <Server> Parameters (Continued)

3.5.3 Enterprise Message API 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.

CHANNEL CONNECTION TYPE	CHANNELTYPE	ENCRYPTEDPROTOCOLTYPE
Unencrypted Socket	RSSL_SOCKET	Not used
Encrypted Socket	RSSL_ENCRYPTED	RSSL_SOCKET

Table 10: Server Settings for Socket Connection Types

3.5.4 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	UInt	0 (System default TLS settings)	This is an unsigned integer representing a combination of flags specified in **`EmaConfig::EncryptedTLSProtocolFlags`**. By default, this is 0, which will match .NET **`sslProtocols.None`**, indicating that this connection will use the system default TLS version(s).
ServerCert	EmaString	""	**Required**. Specifies the filename of the server's certificate.
ServerPrivateKey	EmaString	""	**Required**. Specifies the filename of the server's private key.

Table 11: RSSL_ENCRYPTED ServerType Parameters

3.6 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.

3.6.1 Generic XML Schema for LoggerGroup

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

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

3.6.2 Logger Entry Parameters

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

PARAMETER NAME	TYPE	DEFAULT	NOTES
FileName	String	“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	ulong	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	String		Sets a unique name for the Logger component in the **LoggerList**.
NumberOfLogFiles	ulong	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_<*pid*>.N.log**" is created. Where "**emaLog**" is configured by "**FileName**" parameter of this section, and "**N**" is a sequential number starting from 0 till "**NumberOfLogFiles**".
MaxLogFileSize	ulong	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	String	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 Section 4.5). Possible values are: • LoggerSeverity::Verbose (0) • LoggerSeverity::Success (2) • LoggerSeverity::Warning (3) • LoggerSeverity::Error (4) • LoggerSeverity::NoLogMsg (5)

Table 12: Logger Group Parameters

PARAMETER NAME	TYPE	DEFAULT	NOTES
LoggerType	String	File	Specifies the logging mechanism. Use enumeration values with the Enterprise Message API's programmatic configuration (for details, refer to Section 4.5). 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 12: Logger Group Parameters (Continued)

3.7 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.

3.7.1 Generic XML Schema for DictionaryGroup

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

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

3.7.2 Dictionary Entry Parameters

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

PARAMETER NAME	TYPE	DEFAULT	NOTES
DictionaryType	String	ChannelDictionary	Specifies the dictionary loading mode. Use strings with Enterprise Message API's programmatic configuration (for further details, refer to Section 4.5). Possible values are: • **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 13: Dictionary Group Parameters

3.8 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**).

3.8.1 Generic XML Schema for Directory Entry

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

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

3.8.2 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"/>
```

3.8.3 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 14: Directory Entry Parameters

3.8.4 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.
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 15: Service Entry Parameters

3.8.5 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	UInt	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	UInt	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.

Table 16: Source Directory Info Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
Capabilities	Component Name	N/A	A component that includes **CapabilitiesEntry** parameters, which define the message domain types that can be requested from the service. For details on the parameter used in this section, refer to Section 3.8.5.1.
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 3.8.5.2.
AcceptingConsumerStatus	UInt	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 3.8.5.3.
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 3.8.5.4.
SupportsQoSRange	UInt	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	UInt	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 16: Source Directory Info Parameters (Continued)

3.8.5.1 Capabilities Entry Parameter

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

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

Table 17: CapabilitiesEntry Parameter

3.8.5.2 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 18: DictionariesProvided Parameter

3.8.5.3 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 19: DictionariesUsedEntry Parameter

3.8.5.4 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	UInt 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.Timeliness` class.
Rate	UInt 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.Rate` class.

Table 20: QoSEntry Section and Associated Parameters

3.8.6 StateFilter Entry Parameters

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

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ServiceState	UInt	N/A	Specifies whether the service is up or down: • 0: Service is down • 1: Service is up
AcceptingRequests	UInt	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 21: StateFilter Parameters

3.8.7 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.StreamState`** class.
DataState	EmaString	DataState::Ok	Specifies the state of the item data. **NOTE:** Acceptable **DataState** values are listed in the **`OmmState.DataState`** class.
StatusCode	EmaString	StatusCode::None	Specifies the item status code. **NOTE:** Codes and their meanings are listed in the **`OmmState.StatusCode`** class.
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 22: Status Entry Parameters

3.8.8 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 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 **Channel** or **Dictionary** in their respective lists. Instead, if an application wants to use a specific channel or dictionary configuration, the application must explicitly configure it in the appropriate **Consumer** section of the XML file.

4.3 Processing Enterprise Message API's XML Configuration File

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

Except for the parameter **DefaultConsumer**, you must wrap all other elements defined in the Enterprise Message API's configuration file in a component definition (i.e., **Consumer**, **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**, which must be located in the current working directory. By precedence, the Enterprise Message API uses the **EmaConfig.xml** from the current working directory. If the Enterprise Message API does not find an **EmaConfig.xml**, the application uses the default configuration. Additionally, if the **EmaConfig.xml** file is empty or contains malformed XML, the application uses the default configuration. For details on the default configuration, refer to Section 4.2.

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

```
OmmConsumerConfig config = new OmmConsumerConfig();

consumer = new OmmConsumer(config);
```

For complete details, refer to the example *100__MP__Streaming* included with the Refinitiv Real-Time SDK.

4.3.1.2 Using a Custom Filename and/or Directory

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

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

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

If you want to specify a custom path and filename, have the application create an **OmmConsumerConfig** object with the path and filename in the argument (for details on this object, refer to the *Enterprise Message API 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 = new OmmConsumerConfig(PATH);

consumer = new OmmConsumer(config);
```

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

4.3.2 Use of the Correct Order in the XML Schema

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

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

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

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

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

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

- Configuration 1:

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

- Configuration 2:

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

4.3.3 Processing the Consumer “Name”

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

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

4.4 Configuring the Enterprise Message API Using Method Calls

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

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

4.4.1 Enterprise Message API Configuration Method Calls

4.4.1.1 OmmConsumerConfig Class Method Calls

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

METHOD	DESCRIPTION
AddAdminMsg(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 Refinitiv 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.
ApplicationId(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(EmaString)	**Required**. Specifies an authentication parameter. • Version 2 Authentication: a unique ID provisioned as part of Service Account used to make an authentication request. For details, 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(EmaString)	Sets the client secret. **Required** for Version 2 oAuthClientCredential authentication and provisioned as part of Service Account.
Config(Data)	Passes in the consumer's programmatic configuration.
ConsumerName(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(DataDictionary, boolean)	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.

Table 23: `OmmConsumerConfig` Class Method Calls

METHOD	DESCRIPTION
EncryptedProtocolType(EncryptedProtocol Types)	Optional. Specifies the encrypted protocol type used by the current consumer. Use **`EmaConfig.EncryptedProtocolTypeEnum`** to set allowed encrypted protocol type.
Host(EmaString)	Sets the host and port parameters. For details, refer to . Sample value: “localhost:14002”.
OperationModel(OperationModel)	Optional. Sets the operation model to either of these: • **`OperationModel.API_DISPATCH`** (default) • **`OperationModel.USER_DISPATCH`**
Password(EmaString)	**Required** for Version 1 oAuthPasswordGrant authentication. Specifies the password used together with the username to obtain the access token.
Position(EmaString)	Sets the **position** variable. **position** has no default value.
RestProxyHost(String)	Optional. Specifies the address or host name of an HTTP proxy server for REST requests.
RestProxyPassword(String)	Optional. Specifies the password to authenticate to the proxy server for REST requests.
RestProxyPort(String)	Optional. Specifies the port number of the proxy server for REST requests.
RestProxyUserName(String)	Optional. Specifies the user name to authenticate to the proxy server for REST requests.
ServiceDiscoveryUrl(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/**.
TokenScope(EmaString)	Optional for 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**.
TokenServiceUrlV2(EmaString)	Optional. Specifies a URL to override the default for token service V2 oAuthClientCredentials to perform authentication to get access tokens. Default value is https://api.refinitiv.com/auth/oauth2/v2/token.
ProxyUserName	Optional. Specifies the username for an authenticated proxy.
ProxyPassword	Optional. Specifies the password for an authenticated proxy.
ProxyHost(EmaString)	Optional. Specifies the host name of an HTTP Proxy for any Socket or Encrypted connections.
ProxyPort(EmaString)	Optional. Specifies the port number of the proxy server to connect to for an HTTP connection.
TlsCipherSuites(IEnumerable<TlsCipherSuite > cipherSuites)	Optional. Specifies the collection of cipher suites allowed for TLS negotiation.
EncryptedProtocolFlags(uint protocolFlags)	Optional. Specifies the encrypted protocol flags defined in the **EmaConfig.EncryptedTLSProtocolFlags** class.

Table 23: `OmmConsumerConfig` Class Method Calls(Continued)

4.4.1.2 OmmIProviderConfig Class Method Calls

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

METHOD	DESCRIPTION
addAdminMsg(RefreshMsg)	Optional. Populates the entirety of the initial directory refresh message according to the specification discussed in the *Enterprise Message API C# EditionRDM Usage Guide*. Supports Directory domain only.
AdminControlDictionary(OmmIProviderConfig. AdminControlMode)	Optional. Specifies whether API or user controls responding to Dictionary requests. Default is **OmmIProviderConfig.AdminControlMode.API_CONTROL**.
AdminControlDirectory(OmmIProviderConfig. AdminControlMode)	Specifies whether the API or the user controls the sending of Directory refresh messages. Default is **OmmIProviderConfig.AdminControlMode.API_CONTROL**.
TlsCipherSuites(IEnumerable<TlsCipherSuites>)	Optional. Specifies the cipher suites used by the provider.This is an IEnumerable collection containing **System.Net.Security.TlsCipherSuite** enumerations.
Clear()	Clears existing content from the **`OmmIProviderConfig`** object.
Config(Data)	Passes in the provider's programmatic configuration.
OperationModel(OmmIProvider. OperationalModelMode)	Optional. Specifies the operation model. Default is **API Dispatch: OmmIProviderConfig.OperationalModelMode.API_DISPATCH**.
Port(String)	Optional. Specifies a port. Default is **14002**.
ProviderName(String)	This name identifies configuration section to be used by an **OmmIProvider** instance.
serverCert(String)	Specifies the location of the server certificate file for encrypted providers.
serverPrivateKey(String)	Specifies the location of the private key file for encrypted providers.

Table 24: `OmmIProviderConfig` Class Method Calls

4.4.1.3 OMMNiProviderConfig Class Method Calls

You can use the following method calls in an Enterprise Message API **NiProvider** application. For further details on variables, refer to the *Enterprise Message API C# EditionRDM Usage Guide*.

METHOD	DESCRIPTION
addAdminMsg(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# EditionRDM Usage Guide*. Supports Login domain only.
adminControlDirectory(OmmNiProviderConfig. AdminControlMode)	Optional. Specifies whether the API or the user controls the sending of Directory refresh messages. Available values include: • **OmmIProviderConfig.AdminControlMode.API_CONTROL** • **OmmNiProviderConfig.AdminControlMode.API_CONTROL** For details on control models, refer to **OmmNiProviderConfig**.

Table 25: `OmmNiProviderConfig` Class Method Calls

METHOD	DESCRIPTION
applicationId(String)	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.
config(Data)	Passes in the NiProvider's programmatic configuration.
getProviderRole()	Retrieves NiProvider's role.
host(String)	Optional. Specifies a hostname and port. By default, it is set to **localhost:14003**.
instanceId(String)	Optional. Specifies the instance identifier. Can be any ASCII string, e.g. "Instance1".
operationModel(OmmNiProvider. OperationalModelMode)	Optional. Specifies the operation model. Default is **OmmNiProvider.OperationalModelMode.API_DISPATCH**.
password(String)	Specifies password. Overrides a value specified in Login domain via the **`addAdminMsg( const ReqMsg& )`** method.
position(String)	Specifies position in login request attribute.
providerName(String)	This name identifies configuration section to be used by **OmmNiProvider** instance.
proxyPasswd(String)	Optional. Specifies the password to authenticate. Needed for all authentication protocols.
proxyUserName(String)	Optional. Specifies the user name to authenticate. Needed for all authentication protocols.
EncryptedProtocolFlags(String)	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:TlsCipherSuites** flags to set allowed protocols.
TlsCipherSuites(IEnumerable<TlsCipherSuite > cipherSuites)	Optional. Specifies the cipher suites to be used for an Encrypted connection. This is an IEnumerable collection of **System.Net.Security.TlsCipherSuite** enumerations.
username(String)	Specifies the name used in the login request. Overrides a value specified in the Login domain via the **`addAdminMsg( RequestMsg )`** method.

Table 25: `OmmNiProviderConfig` Class Method Calls

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

Host and **Port** parameters both have global default values. Thus, if either an **`OmmConsumerConfig`** 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**.

If needed, you can have the application reset both host and port values by calling the **`Host( 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( 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 Method 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**.

For more details on **ServiceEndpointDiscovery** and the classes it uses for functionality, refer to the reference manual.

4.5 Programmatic Configuration

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

4.5.1 OMM Data Structure

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

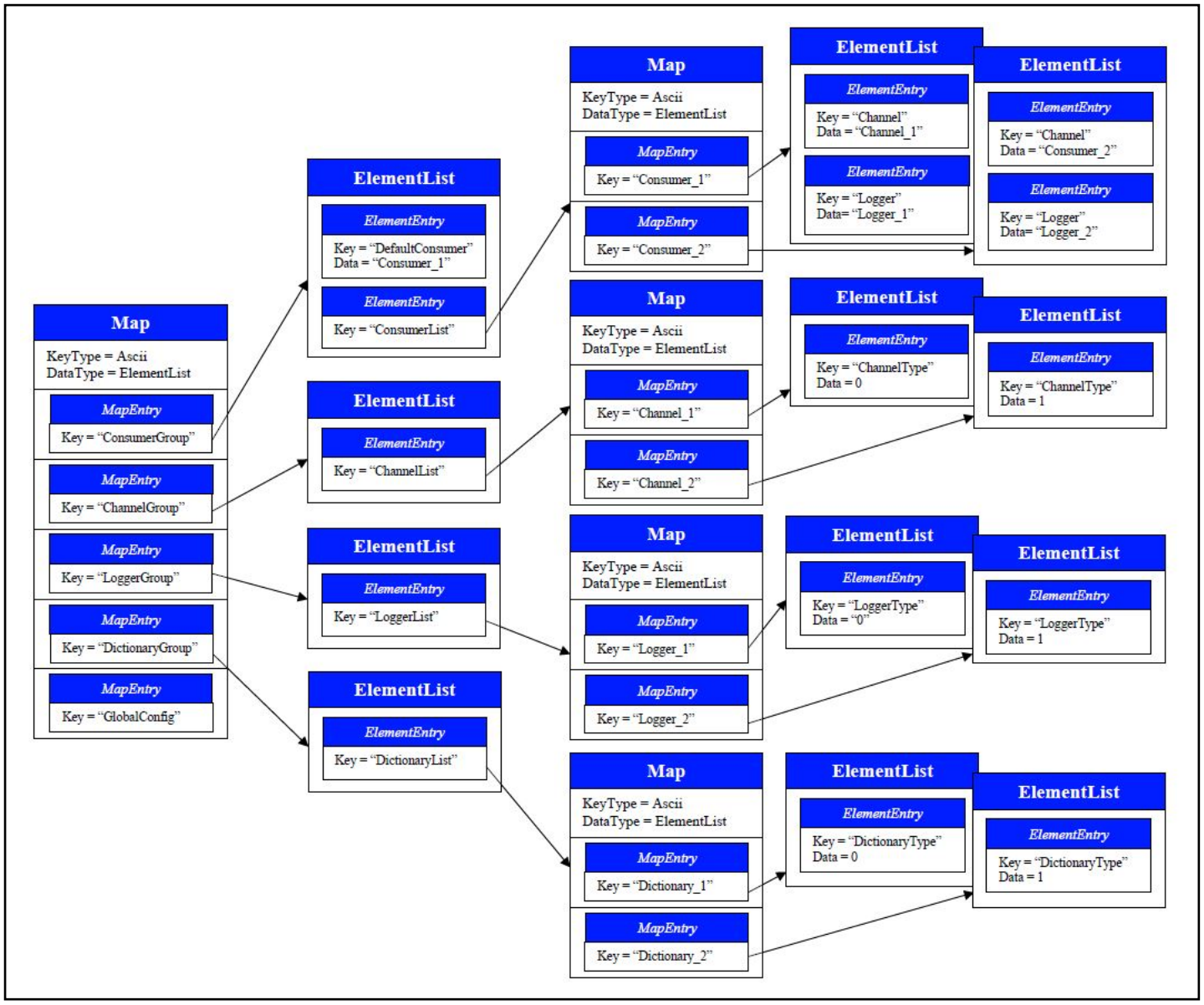
- The 1st tier lists the following Enterprise Message API's components each of which has its own list in the 2nd tier:
  - Consumer
  - IProvider
  - NiProvider
  - Channel
  - Logger
  - Directory
  - Dictionary
- The 2nd tier includes each component's list and the default consumers and providers for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

4.5.2 Creating a Programmatic Configuration for a Consumer

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

▶ To programmatically configure an Enterprise Message API Consumer:

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



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

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

4.5.3 Example: Programmatic Configuration of a Consumer

The following example illustrates programmatically configuring a Consumer:

```
Map innerMap = new Map();
Map configMap = new Map();
ElementList elementList = new ElementList();
ElementList innerElementList = new ElementList();

elementList.AddAscii("DefaultConsumer", "Consumer_1");
innerElementList.AddAscii("ChannelSet", "Channel_1, Channel_2");
innerElementList.AddAscii("Dictionary", "Dictionary_1");
innerMap.AddKeyAscii( "Consumer_1", MapAction.ADD, innerElementList.Complete() );
innerElementList.Clear();

elementList.AddMap( "ConsumerList", innerMap.Complete() );
innerMap.Clear();
configMap.AddKeyAscii( "ConsumerGroup", MapAction.ADD, elementList.Complete() );
elementList.Clear();

innerElementList.AddEnum("ChannelType", ConnectionTypeEnum.SOCKET);
innerElementList.AddAscii("Host", "localhost");
innerElementList.AddAscii("Port", "14002");
innerMap.AddKeyAscii( "Channel_1", MapAction.ADD, innerElementList.Complete() );
innerElementList.Clear();

innerElementList.AddEnum("ChannelType", ConnectionTypeEnum.SOCKET);
innerElementList.AddAscii("Host", "121.1.1.100");
innerElementList.AddAscii("Port", "14008");
innerMap.AddKeyAscii( "Channel_2", MapAction.ADD, innerElementList.Complete() );
innerElementList.Clear();

elementList.AddMap( "ChannelList", innerMap.Complete() );
innerMap.Clear();
configMap.AddKeyAscii( "ChannelGroup", MapAction.ADD, elementList.Complete() );
elementList.Clear();

innerElementList.AddEnum("DictionaryType", DictionaryTypeEnum.CHANNEL);
innerElementList.AddAscii("RdmFieldDictionaryFileName", "./RDMFieldDictionary");
innerElementList.AddAscii("EnumTypeDefFileName", "./enumtype.def");
innerMap.AddKeyAscii( "Dictionary_1", MapAction.ADD, innerElementList.Complete() );
innerElementList.Clear();

elementList.AddMap( "DictionaryList", innerMap.Complete() );
configMap.AddKeyAscii( "DictionaryGroup", MapAction.ADD, elementList.Complete() );
elementList.Clear();
configMap.Complete();

...
```

```
consumer = new OmmConsumer(new OmmConsumerConfig().Config(configMap));
```

4.5.4 Creating a Programmatic Configuration for a Provider

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

▶ To programmatically configure an Enterprise Message API 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

4.5.5 Example: Programmatic Configuration of a Provider

The following example illustrates programmatically configuring a Provider:

```
Map innerMap = new Map();
Map configMap = new Map();
ElementList elementList = new ElementList();
ElementList innerElementList = new ElementList();

elementList.AddAscii("DefaultIProvider", "Provider_1");

innerElementList.AddAscii("Server", "Server_1")
    .AddAscii("Logger", "Logger_1")
    .AddAscii("Directory", "Directory_1")
    .AddUint("ItemCountHint", 5000)
    .AddUint("ServiceCountHint", 5000)
    .AddUint("RequestTimeout", 5000)
    .addInt("DispatchTimeoutApiThread", 1)
    .AddUint("CatchUnhandledException", 0)
    .AddUint("MaxDispatchCountApiThread", 500)
    .AddUint("MaxDispatchCountUserThread", 500)
    .AddAscii("XmlTraceFileName", " MyXMLTrace")
    .addInt("XmlTraceMaxFileSize", 70000000)
    .AddUint("XmlTraceToFile", 0)
    .AddUint("XmlTraceToStdout", 1)
    .AddUint("XmlTraceToMultipleFiles", 0)
```

```
    .AddUint("XmlTraceWrite", 0)
    .AddUint("XmlTraceRead", 0)
    .AddUint("XmlTracePing", 1)
    .AddUint("XmlTraceHex", 1)
    .addInt("PipePort", 9696)
    .AddUint("RefreshFirstRequired", 1)
    .AddUint("AcceptDirMessageWithoutMinFilters", 0)
    .AddUint("AcceptMessageSameKeyButDiffStream", 0)
    .Complete();

innerMap.AddKeyAscii("Provider_1", MapEntry::AddEnum, elementList)
    .Complete()

elementList.addMap("IProviderList", innerMap)
    .Complete();

configMap.AddKeyAscii("IProviderGroup", MapEntry::AddEnum, elementList);
elementList.clear();
innerMap.clear();
innerElementList.Clear();

innerElementList.AddEnum("ServerType", 0)
    .AddEnum("CompressionType", 1)
    .AddUint("GuaranteedOutputBuffers", 5000)
    .AddUint("NumInputBuffers", 5000)
    .AddUint("ConnectionPingTimeout", 70000)
    .AddAscii("Port", "14002")
    .AddUint("TcpNodelay", 1)
    .Complete();

innerMap.AddKeyAscii("Server_1", MapEntry::AddEnum, innerElementList)
    .Complete();
elementList.addMap("ServerList", innerMap)
    .Complete();

configMap.AddKeyAscii("ServerGroup", MapEntry::AddEnum, elementList);
elementList.clear();
innerMap.clear();
innerElementList.Clear();

innerElementList.AddEnum("LoggerType", 0)
    .AddAscii("FileName", "logFileProv")
    .AddEnum("LoggerSeverity", 1).Complete())
    .Complete();

innerMap.AddKeyAscii("Logger_1", MapEntry::AddEnum, elementList)
    .Complete();

elementList.addMap("LoggerList", innerMap
    .Complete();
```

```
outermostMap.AddKeyAscii("LoggerGroup", MapEntry::AddEnum, elementList);
elementList.clear();
innerMap.clear();
innerElementList.Clear();



    .AddEnum("DictionaryType", 0)
    .AddAscii("RdmFieldDictionaryItemName", "RWFFld")
    .AddAscii("EnumTypeDefItemName", "RWFEnum")
    .AddAscii("RdmFieldDictionaryFileName", "./RDMFieldDictionary")
    .AddAscii("EnumTypeDefFileName",
    "./enumtype.def")
    .Complete();

innerMap.AddKeyAscii("Dictionary_1", MapEntry::AddEnum,ElementList)
    .Complete();
elementList.addMap("DictionaryList", innerMap)
    .Complete();
innerMap.clear();

configMap.AddKeyAscii("DictionaryGroup", MapEntry::AddEnum, elementList);
elementList.clear();
innerElementList.Clear();

Map serviceMap = new Map();
ElementList serviceFilterElementList = new ElementList();
ElementList serviceElementList = new ElementList();
Series qosSeries = new Series();
Array serviceArray = new Array();

serviceElementList.AddUint("ServiceId", 1)
    .AddAscii("Vendor", "Vendor")
    .AddUint("IsSource", 1)
    .AddUint("AcceptingConsumerStatus", 1)
    .AddUint("SupportsQoSRange", 1)
    .AddUint("SupportsOutOfBandSnapshots", 1)
    .AddAscii("ItemList", "#.itemlist")

    serviceArray.AddAscii("MMT_MARKET_PRICE")
        .AddAscii("MMT_MARKET_BY_PRICE")
        .AddAscii("MMT_MARKET_BY_ORDER")
        .AddAscii("130")
        .Complete();

serviceElementList.AddArray("Capabilities", serviceArray);
serviceArray.Clear();

serviceArray.AddAscii("Dictionary_1")
    .Complete();
```

```
serviceElementList.addArray("DictionariesProvided", serviceArray);
serviceArray.Clear();

serviceArray.AddAscii("Dictionary_1")
    .Complete();

serviceElementList.addArray("DictionariesUsed", serviceArray);
serviceArray.Clear();
innerElementList.Clear();

innerElementList.AddAscii("Timeliness", "Timeliness::RealTime")
    .AddAscii("Rate", "Rate::TickByTick")
    .Complete();

qosSeries.AddEntry(innerElementList)
    .Complete();

serviceElementList.addSeries("QoS", qosSeries)
    .Complete();

serviceFilterElementList.addElementList("InfoFilter", serviceElementList);
serviceElementList.Clear();
innerElementList.Clear();


serviceElementList.AddUint("ServiceState", 1)
    .AddUint("AcceptingRequests", 1);

innerElementList.AddAscii("StreamState", "StreamState::Open")
    .AddAscii("DataState", "DataState::Suspect")
    .AddAscii("StatusCode", "StatusCode::DacsDown")
    .AddAscii("StatusText", "dacsDown")
    .Complete();

serviceElementList.addElementList("Status", innerElementList)
    .Complete();

serviceFilterElementList.addElementList("StateFilter", serviceElementList)
    .Complete();

serviceMap.AddKeyAscii("DIRECT_FEED", MapEntry::AddEnum, serviceFilterElementList)
    .Complete();

innerMap.AddKeyAscii("Directory_1", MapEntry::AddEnum, serviceMap)
    .Complete();
elementList.clear();
elementList.addMap("DirectoryList", innerMap)
    .Complete();
configMap.AddKeyAscii("DirectoryGroup", MapEntry::AddEnum, elementList)
    .Complete();
```

```
...

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

Document ID: EMACSharp320CG.240
Date of issue: April 2024