# Enterprise Message API C# Edition 3.1.0.L1

## ENTERPRISE MESSAGE API CONFIGURATION GUIDE

Document Version: 3.1.0
Date of issue: November 2023
Document ID: EMACSharp310CG.230

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 the 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*.

This document provides detailed yet supplemental information for application developers writing to the Enterprise Message API.

## 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 |

**Table 1: Acronyms and Abbreviations**

| ACRONYM / TERM | MEANING |
|---|---|
| HTTPS | Hypertext Transfer Protocol (Secure) |
| 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 Configuration 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:

- Methods, in-line code snippets, and types are shown in **`Courier New`** font.
- Parameters, filenames, tools, utilities, and directories are shown in **Bold** font.
- Document titles and variable values are shown in *italics*.
- When initially introduced, concepts are shown in ***Bold, Italics***.

### 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 EMA Configuration General Overview

## 2.1 About Message API Configuration

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

Message API configuration data is divided into the following groups:

- ***Consumer***: Consumer configuration data are the highest-level description of the application. Such settings typically select entries from the channel, logger, and dictionary 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.

The Consumer group is the top-level configuration group. Specific consumer applications select their configurations according to the name specified in the **`ConsumerName()`** 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 | String | N/A | Specifies the channel that the **Consumer** component should use. This channel must match the **Name** parameter from the appropriate **<Channel>** entry in the **ChannelGroup** configuration.<br>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 3.2. |
| ChannelSet | String | N/A | Specifies a comma-separated set of channels names. Each listed channel name should have an appropriate **<Channel>** entry in the **ChannelGroup**. Channels in the set will be tried with each reconnection attempt until a successful connection is made.For further details refer to Section 3.2.6<br>**NOTE:** If both **Channel** and **ChannelSet** are configured, then the Enterprise Message API uses the **<ChannelSet>** parameter. |
| Dictionary | String | N/A | Specifies how the consumer should access its dictionaries (it must match the **Name** parameter from the appropriate **<Dictionary>** entry in the **DictionaryGroup** configuration).<br>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 3.4. |
| 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.<br>If set to **0**, the Enterprise Message API will wait for a response indefinitely.<br>**NOTE:** If **ChannelSet** is configured:<br>• The Enterprise Message API honors **DictionaryRequestTimeOut** only on its first connection.<br>• 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.<br>If set to **0**, the Enterprise Message API will wait for a response indefinitely.<br>**NOTE:** If **ChannelSet** is configured:<br>• The Enterprise Message API honors **DirectoryRequestTimeOut** only on its first connection.<br>• 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.<br>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.<br>Available values include:<br>• 0 (false)<br>• 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**.<br>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.<br>If set to **0**, the Enterprise Message API will wait for a response indefinitely.<br>**NOTE:** If **ChannelSet** is configured:<br>• The Enterprise Message API honors **LoginRequestTimeOut** only on its first connection.<br>• 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:<br>• **0** (false): Do not fill in the message's key values (values received from the wire are preserved).<br>• **1** (true): Fill in the message's key values (values received from the wire are overridden). |
| Name | String | N/A | Specifies the name of this **Consumer** component. **Name** is required when creating a **Consumer** component.<br>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:<br>• **0** (false)<br>• **1** (true) |
| PostAckTimlongeout | 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.<br>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.<br>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.<br>If set to **0**, the Enterprise Message API will wait for a response indefinitely. |
| 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.<br>If set to **0**, there is no timeout. |
| 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**. |
| XmlTraceToStdout | ulong | 0 | Specifies whether the Enterprise Message API traces its messages in XML format to stdout. Possible values are:<br>• **0** (false): Turns off tracing.<br>• **1** (true): Turns on tracing to stdout. |

**Table 2: Consumer Group Parameters (Continued)**

## 3.2 Channel Group

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

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** section.

- For details on the parameters you can use to configure the **Consumer** component, refer to Section 3.1.4.
- For a generic **ChannelGroup** XML schema, refer to Section 3.2.1.
- For a list of universal parameters you can use in configuring any type of **Channel** regardless of the channel type, refer to Section 3.2.2.
- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_SOCKET**, refer to Section 3.2.4.
- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_ENCRYPTED**, refer to Section 3.2.5.

### 3.2.1 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>
```

### 3.2.2 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 3.2.3.

| PARAMETER NAME | TYPE | DEFAULT | NOTES |
|---|---|---|---|
| ChannelType | String | RSSL_SOCKET | Specifies the type of channel or connection used to connect to the server.<br>Calling the host function can change this field. For details on this event, refer to Section 4.4.2.<br>Use strings with Enterprise Message API's programmatic configuration (for further details, refer to Section 4.5). Available values include:<br>• **RSSL_SOCKET**<br>• **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. |
| EnableSessionManagement | ulong | 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.<br>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.<br>For details on RsslChannel and **maxFragmentSize**, refer to the *Transport API 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 | uin | 5<br>(10 when used with **RSSL_ENCRYPTED** ChannelType) | Specifies the time (in seconds) to wait for the successful initialization of a channel. |
| InterfaceName | String | "" | 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.<br>**InterfaceName** is for use in systems that have multiple network interface cards. If unspecified, the default network interface is used. |

**Table 3: Universal <Channel> Parameters**

| PARAMETER NAME | TYPE | DEFAULT | NOTES |
|---|---|---|---|
| Location | String | us-east-1 | Used only when **host** and **port** are unspecified, **Location** specifies the cloud location of the service provider endpoint to which the RTSDK API establishes a connection. If **Location** is not specified, the default setting is **us-east-1**. In any particular cloud location, the Enterprise Message API connects to the endpoint that provides two available zones for the location (e.g., **[us-east-1a, us-east-1b]**).<br>You can use **Location** only on an **RSSL_ENCRYPTED** ChannelType. |
| Name | String | | Specifies the **Channel**'s name. |
| NumInputBuffers | uint | 100 | Specifies the number of buffers used to read data. Buffers are sized according to **maxFragmentSize**.<br>For details on RsslChannel and **maxFragmentSize**, refer to the *Transport API 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.<br>Used only when:<br>• Host and port are unspecified. Refer to Section 3.2.4.<br>• **EnableSessionManagement** is set to **1**.<br>For details on service discovery, refer to the *Enterprise Message API Developers Guide*.<br>**NOTE:** You can use this parameter only with Enterprise Message API consumers.<br>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 3: Universal <Channel> Parameters (Continued)**

### 3.2.3 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 4: Channel Settings for Socket Connection Types**

### 3.2.4 Parameters for Use with Channel Type: RSSL_SOCKET

In addition to the universal parameters listed in Section 3.2.2, you can use the following parameters to configure a channel whose type is **RSSL_SOCKET**.

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

**Table 5: Parameters for Channel Type: RSSL_SOCKET**

### 3.2.5 Parameters for Use with Channel Types: RSSL_ENCRYPTED

In addition to the universal parameters listed in Section 3.2.2, and the parameters listed in the section specific to the protocol type you use (i.e., Section 3.2.4 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.<br>• **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:<br>Currently, TLS versions are TLS 1.2 (4) and TLS 1.3(8). |

**Table 6: Parameters for Channel Types: ENCRYPTED**

### 3.2.6 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>
```

### 3.2.7 Example ChannelSet XML Configuration

The following XML example illustrates a specific ChannelSet configuration using the XML schema introduced in Section 3.2.6:

```
<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>
```

### 3.2.8 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()
```

## 3.3 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.

### 3.3.1 Generic XML Schema for LoggerGroup

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

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

### 3.3.2 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.<br>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:<br>• **0** (false): Include only the time, omitting the date.<br>• **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:<br>• 0: A file with unique pid name is created. Log file rolling is disabled by default.<br>• 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"`**).<br>Possible values are:<br>• 0: A log file grows indefinitely. Log file size limit is disabled.<br>• 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.<br>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**).<br>Use enumeration values with the Enterprise Message API’s programmatic configuration (for details, refer to Section 4.5).<br>Possible values are:<br>• LoggerSeverity::Verbose (0)<br>• LoggerSeverity::Success (1)<br>• LoggerSeverity::Warning (2)<br>• LoggerSeverity::Error (3)<br>• LoggerSeverity::NoLogMsg (4) |

**Table 7: Logger Group Parameters**

| PARAMETER NAME | TYPE | DEFAULT | NOTES |
|---|---|---|---|
| LoggerType | String | File | Specifies the logging mechanism.<br>Use enumeration values with the Enterprise Message API's programmatic configuration (for details, refer to Section 4.5).<br>Possible values are:<br>• **LoggerType::File**: The Enterprise Message API logs to the file specified in the parameter **FileName**.<br>• **LoggerType::Stdout**: The Enterprise Message API logs to stdout. |

**Table 7: Logger Group Parameters (Continued)**

## 3.4 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.

### 3.4.1 Generic XML Schema for DictionaryGroup

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

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

### 3.4.2 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.<br>Use strings with Enterprise Message API's programmatic configuration (for further details, refer to Section 4.5). Possible values are:<br>• **FileDictionary** (0): The Enterprise Message API loads the dictionaries from the files specified in the parameters **RdmFieldDictionaryFileName** and **EnumTypeDefFileName**.<br>• **ChannelDictionary** (1): The Enterprise Message API downloads dictionaries by requesting the dictionaries from the upstream provider. |
| EnumTypeDefFileName | String | ./enumtype.def | Sets the location of the **EnumTypeDef** file. |
| EnumTypeDefItemName | String | RWFEnum | Sets the name of the EnumTypeDef item specified in the source directory **InfoFilter.DictionariesProvided**, and **InfoFilter.DictionariesUsed** elements. |
| Name | String | | Sets a unique name for a Dictionary component in the **DictionaryList**. |
| RdmFieldDictionaryFileName | String | ./RDMFieldDictionary | Sets the location of the **RdmFieldDictionary**. |
| RdmFieldDictionaryItemName | String | RWFFld | Sets the name of the RdmFieldDictionary item specified in the source directory **InfoFilter.DictionariesProvided**, and **InfoFilter.DictionariesUsed** elements. |

**Table 8: Dictionary Group Parameters**

# 4 EMA Configuration Processing

## 4.1 Overview and Configuration Precedence

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

## 4.2 Default Configuration

### 4.2.1 Default Consumer Configuration

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

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

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

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

## 4.3 Processing EMA's XML Configuration File

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

Except for the 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 EMA Using Function 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 function calls on the **OmmConsumerConfig** objects.

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

### 4.4.1 EMA Configuration Method Calls

#### 4.4.1.1 OmmConsumerConfig Class Method Calls

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

| METHOD | DESCRIPTION |
|---|---|
| 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 Reuters Domain Models (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( String ) | Sets the **applicationId** variable. **applicationId** has no default value. |
| Clear() | Clears existing content from the **OmmConsumerConfig** object. |
| ClientId( String ) | **Required**. Specifies an authentication parameter.<br>• Version 2 Authentication: a unique ID provisioned as part of Service Account used to make an authentication request.<br>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( String) | Sets the client secret.<br>**Required** for Version 2 oAuthClientCredential authentication and provisioned as part of Service Account. |
| Config( Data ) | Passes in the consumer's programmatic configuration. |
| ConsumerName( String) | 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.<br>Overrides DataDictionary object provided via **EmaConfig.xml** or programmatic configuration. |
| Host( String) | Sets the host and port parameters. For details, refer to. Sample value: "localhost:14002". |
| OperationModel( OperationModel ) | Optional.<br>Sets the operation model to either of these:<br>• **OperationModel.API_DISPATCH** (default)<br>• **OperationModel.USER_DISPATCH** |

**Table 9: OmmConsumerConfig Class Method Calls**

| METHOD | DESCRIPTION |
|---|---|
| Password( String ) | Specifies the password used in the OMM Login message. |
| Position( String ) | Sets the **position** variable. **position** has no default value. |
| ServiceDiscoveryUrl( String ) | 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( String ) | 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( String ) | 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( String ) | Optional. Specifies the host name of an HTTP Proxy for any Socket or Encrypted connections. |
| ProxyPort( String ) | Optional. Specifies the port number of the proxy server to connect to for an HTTP connection. |
| Username( String ) | Optional. Sets username; if not specified, username is extracted from run-time environment. |
| 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 9: `OmmConsumerConfig` Class Method Calls(Continued)**

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

**Host** and **Port** parameters both have global default values. Thus, if either an **`OmmConsumerConfig`** 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( String )`** method on the object using the syntax: ***HostValue*:*PortValue***.

---

**NOTE:** Calling the **`Host()`** function sets **channelType** (refer to Section 3.2.2) to **RSSL_SOCKET**, regardless of how it was previously configured.

---

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

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

### 4.4.3 Service Discovery Configuration Using Function Calls

#### 4.4.3.1 ServiceEndpointDiscovery

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

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

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

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:

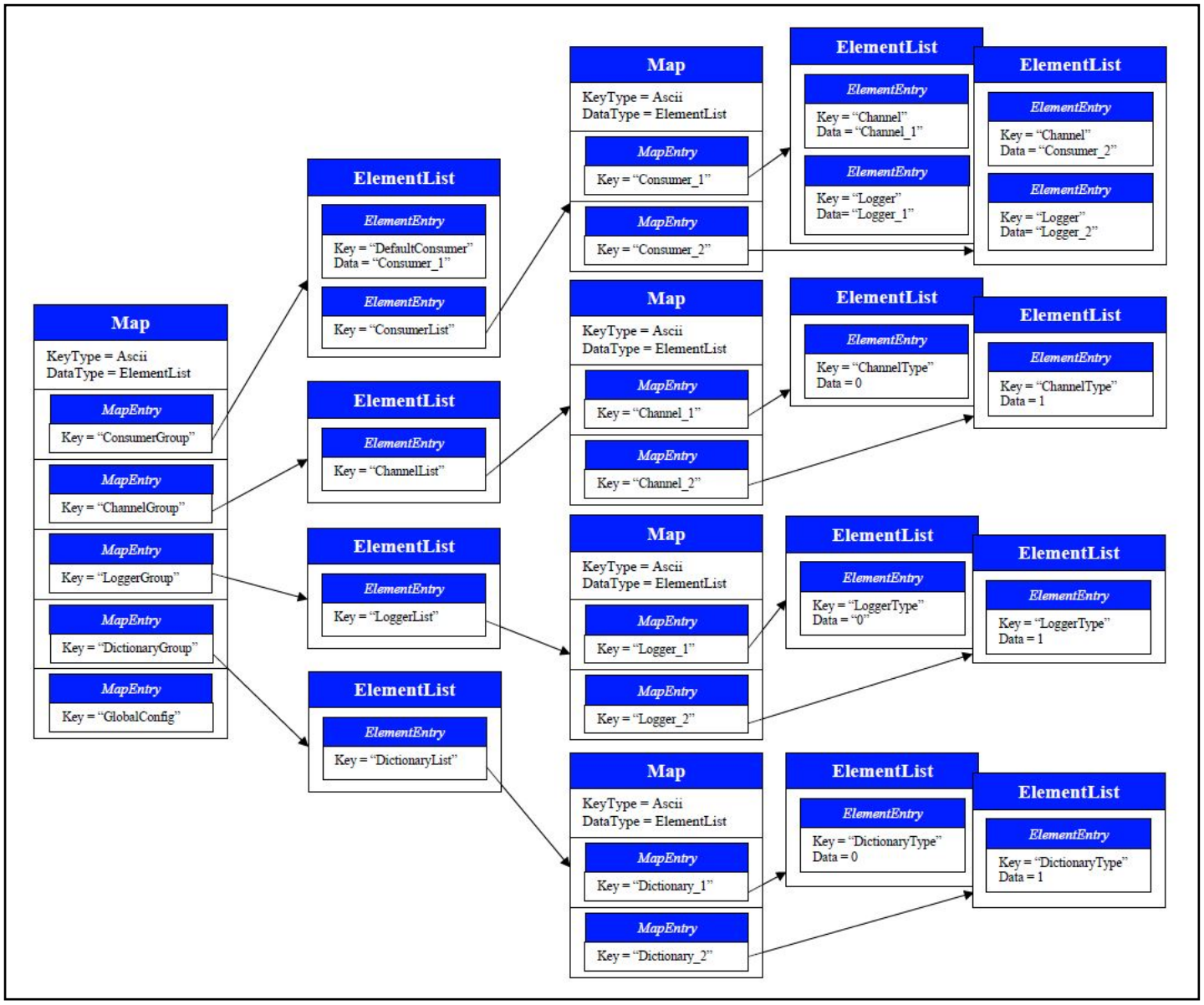
- The 1st tier lists the following Enterprise Message API's components each of which has its own list in the 2nd tier:
  - Consumer
  - Channel
  - Logger
  - Dictionary
- The 2nd tier includes each component's list and the default consumers for use when loading configuration parameters.
- The 3rd tier 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`**.

### 4.5.3 Example: Programmatic Configuration of the 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));
```


Document ID: EMACSharp310CG.230
Date of issue: November 2023