Elektron Message API C++ Edition V3.0.1

ELEKTRON MESSAGE API CONFIGURATION GUIDE

Document Version: 1.0
Date of issue: 13 November 2015
Document ID: EMAC301CG.150

THOMSON REUTERS

Contents

Chapter 1 Guide Introduction

1.1 About this Manual

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

This guide documents the functionality and capabilities of the Elektron Message API C++ Edition . The Elektron Message API can also connect to and leverage many different Thomson Reuters and customer components. If you want the Elektron Message API to interact with other components, consult specific component's documentation to determine the best way to configure and interact with these other devices.

This document explains the configuration parameters for the Elektron 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 Elektron Message API configuration parameters. You can obtain further information from the *Elektron Message API Developer's Guide*.

1.3 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 an XML file (**EmaConfig.xml**) to specify configuration parameters. Additionally, programmatic interfaces can change parameter settings.

Message API configuration data is divided into four types:

- ***Consumer***: Consumer configuration data is 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 provides configuration alternatives for those connections.
- ***Logger***: Logger configuration data specify logging alternatives and associated parameters.
- ***Dictionary***: Dictionary configuration data sets the location information for dictionary alternatives.

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

1.4 Definitions

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

Table 1: Definitions

1.5 Acronyms and Abbreviations

ACRONYM	MEANING
ADH	Advanced Data Hub
ADS	Advanced Distribution Server
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
EED	Elektron Edge Device
EMA	Elektron Message API, referred to simply as the Message API
EOA	Elektron Object API, referred to simply as the Object API.
ETA	Elektron Transport API, referred to simply as the Transport API
EWA	Elektron Web API
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol (Secure)
OMM	Open Message Model
QoS	Quality of Service
EDF	Elektron Data Feeds
EDF Direct	Elektron Data Feed Direct
RDM	Reuters Domain Model
RMTES	Reuters Multi-Lingual Text Encoding Standard

Table 2: Acronyms and Abbreviations

ACRONYM	MEANING
RSSL	Reuters Source Sink Library
RWF	Reuters Wire Format
TREP	Thomson Reuters Enterprise Platform
UML	Unified Modeling Language
UTF-8	8-bit Unicode Transformation Format

Table 2: Acronyms and Abbreviations

1.6 References

1. Elektron Message API C++ Edition *RDM Usage Guide*
2. *API Concepts Guide*
3. *Elektron Message API Developers Guide*
4. *Transport API C Edition Developers Guide*

1.7 Documentation Feedback

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

- Send us your comments via email at apidocumentation@thomsonreuters.com.
- Mark up the PDF using the **Comment** feature in Adobe Reader. After adding your comments, you can submit the entire PDF to Thomson Reuters by clicking **Send File** in the **File** menu. Use the apidocumentation@thomsonreuters.com address.

1.8 Document Conventions

This document uses the following types of conventions:

- Typographic
- Data Types
- Field and Text Values

1.8.1 Typographic

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

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

1.8.2 Data Types

Data types within the configuration repository are as follows:

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

Table 3: Data Type Conventions

1.8.3 Field and Text Values

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

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

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

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

Chapter 2 EMA Global Configuration Parameters

2.1 Parameter Overview

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

For example:

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

For a list of default behaviors (and the parameters that you can use to change these behaviors) refer to Section 2.2.

For details on editing **EmaConfig.xml** and its XML schema, refer to Chapter 2, EMA Global Configuration Parameters.

2.2 Default Behaviors

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

PARAMETER	TYPE	DEFAULT BEHAVIOR	NOTES
Host	EmaString	localhost	Specifies the host name of the server to which the application connects. The parameter value can be a remote host name or IP address.
Port	EmaString	14002	Specifies the port number on the server to which the application connects.
DefaultConsumer	EmaString	EmaConsumer	If consumer components are configured, this parameter is ignored.
LoggerSeverity	Enumeration	Success	Sets the level at which the EMA logs events. For details on logging severity levels and their enumerations, refer to Section 3.3.2.
LoggerType	Enumeration	File	Specifies the destination for output messages. The parameter value can be either `File` or `Stdout`. For details on selecting a loggerType and its enumerations, refer to Section 3.3.2.

Table 4: Global Configuration

PARAMETER	TYPE	DEFAULT BEHAVIOR	NOTES
FileName	EmaString	"emaLog_*pid*.log"	Specifies the base name of log file (used when **`LoggerType value="File"`**); the EMA automatically appends ***_pid*.log** to the base name, where ***pid*** is the logger's process id number.
RdmFieldDictionaryFileName	EmaString	./RDMFieldDictionary	Specifies the path and name of the **RdmFieldDictionary** file.
EnumTypeDefFileName	EmaString	./enumtype.def	Specifies the path and name of the **enumtypeDef** dictionary file.

Table 4: Global Configuration (Continued)

Chapter 3 Configuration Groups

3.1 ConsumerGroup

A **ConsumerGroup** contains two elements:

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

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

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

For a list of parameters you can use in configuring a **ConsumerGroup**, 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 EMA 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** (for an example, refer to Appendix A).

```
<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 **ConsumerGroup** parameters, refer to Section 3.1.4).

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

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

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

3.1.4 Consumer Entry Parameters

Use the following parameters when configuring a **ConsumerGroup** in EMA.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
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**.
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 EMA resorts to default channel behavior when needed. For further details on the **<Channel>** entry and default behaviors, refer to Section 3.2
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 3.2.6. **Note:** If both **Channel** and **ChannelSet** are configured, then EMA uses the parameter that is configured last in the file. For example, if **<Channel>** is configured after **<ChannelSet>** then EMA uses **<Channel>**, but if **<ChannelSet>** is configured after **<Channel>** then EMA uses **<ChannelSet>**.
Logger	EmaString	N/A	Specifies a set of logging behavior the **Consumer** should exhibit (it must match the **Name** parameter from the appropriate **<Logger>** entry in the **LoggerGroup** configuration). If **Logger** is not specified, the EMA uses a set of logger default behaviors. For further details on the **<Logger>** entry and default settings, refer to Section 3.3.

Table 5: Consumer Group Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
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 EMA uses the channel's dictionary when needed. For further details on this default behavior, refer to Section 3.4.
DictionaryRequestTimeOut	UInt64	45,000	Specifies the amount of time (in milliseconds) the application has to download dictionaries from a provider before the **OmmConsumer** throws an exception. If set to **0**, EMA does not create a timeout.
			Note: If **ChannelSet** is configured: • EMA honors **DictionaryRequestTimeOut** only on its first connection. • If the channel supporting the first connection goes down, EMA does not use **DictionaryRequestTimeOut** on subsequent connections.
DirectoryRequestTimeOut	UInt64	45,000	Specifies the amount of time (in milliseconds) the provider has to respond with a source directory refresh message before the **OmmConsumer** throws an exception. If set to **0**, EMA does not create a timeout.
			Note: If **ChannelSet** is configured: • EMA honors **DirectoryRequestTimeOut** only on its first connection. • If the channel supporting the first connection goes down, EMA does not use **DirectoryRequestTimeOut** on subsequent connections.
LoginRequestTimeOut	UInt64	45,000	Specifies the amount of time (in milliseconds) the provider has to respond with a login refresh message before the **OmmConsumer** throws an exception. If set to **0**, EMA does not create a timeout.
			Note: If **ChannelSet** is configured: • EMA honors **LoginRequestTimeOut** only on its first connection. • If the channel supporting the first connection goes down, EMA does not use **LoginRequestTimeOut** on subsequent connections.
ItemCountHint	UInt64	100,000	Specifies the number of items the application expects to request. If set to **0**, EMA resets it to **513**. For better performance, the application can set this to the approximate number of item requests it expects.
ServiceCountHint	UInt64	513	Sets the size of directory structures for managing services. If the application specifies **0**, EMA resets it to **513**.

Table 5: Consumer Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ObeyOpenWindow	UInt64	1	Specifies whether the **OmmConsumer** obeys the **OpenWindow** from services advertised in a provider's Source Directory response. Available values include: • 0 (false) • 1 (true)
PostAckTimeout	UInt64	15,000	Specifies the length of time (in milliseconds) a stream waits to receive an ACK for an outstanding post before forwarding a negative acknowledgment to the application. If set to **0**, EMA does not create a timeout.
RequestTimeout	UInt64	15,000	Specifies the amount of time (in milliseconds) the **OmmConsumer** waits for a response to a request before sending another request. If set to **0**, EMA does not create a timeout.
MaxOutstandingPosts	UInt64	100,000	Specifies the maximum allowable number of on-stream posts waiting for an acknowledgment before the **OmmConsumer** disconnects.
DispatchTimeoutApiThread	Int64	-1	Specifies the duration (in microseconds) for which the internal EMA thread is inactive before going active to check whether a message was received. If set to less than zero, the EMA internal thread goes active only if it gets notified about a received message.
CatchUnhandledException	UInt64	1	Specifies whether EMA catches unhandled exceptions thrown from methods executed on the EMA's thread or whether EMA lets the application handle them. Available values include: • **1** (true): Whenever the EMA catches unhandled exceptions in its thread, the EMA logs an error message and then terminates the thread. • **0** (false): the EMA passes unhandled exceptions to the operating system.
MaxDispatchCountApiThread	UInt64	100	Specifies the maximum number of messages the EMA dispatches before taking a real-time break.
MaxDispatchCountUserThread	UInt64	100	Specifies the maximum number of messages the EMA can dispatch in a single call to the `OmmConsumer::dispatch()`.
PipePort	Int64	9001	Specifies the internal communication port. You might need to adjust this port if it conflicts with other processes on the machine.

Table 5: Consumer Group Parameters (Continued)

3.2 Channel Group

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 group. If a consumer application needs a specific channel group, you should specify this in the appropriate **Consumer** section (for details on configuring the **Consumer** component, refer to Section 3.1.3).

- 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 3.2.3.
- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_ENCRYPTED**, refer to Section 3.2.4.
- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_HTTP**, refer to Section 3.2.4.
- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_RELIABLE_MCAST**, 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**.

PARAMETER NAME	TYPE	DEFAULT	NOTES
Name	EmaString		Specifies the **Channel**'s name.
ChannelType	Enumeration	RSSL_SOCKET	Specifies the type of channel or connection used to connect to the server. Calling the host function can change this field. For details on this event, refer to Section 4.3.2. Use enumeration values with EMA's programmatic configuration (for details, refer to in Section 4.4) Available values include: • **RSSL_SOCKET** (0) • **RSSL_ENCRYPTED** (1): Supported only on Windows OS. • **RSSL_HTTP** (2): Supported only on Windows OS • **RSSL_RELIABLE_MCAST** (4)
GuaranteedOutputBuffers	UInt64	100	Specifies the number of guaranteed buffers (allocated at initialization time) available for use by each RsslChannel when writing data. Each buffer is created to contain **maxFragmentSize** bytes. For details on RsslChannel and **maxFragmentSize**, refer to the *Transport API Developers Guide*.
NumInputBuffers	UInt64	10	Specifies the number of buffers used to read data. Buffers are sized according to **maxFragmentSize**. For details on RsslChannel and **maxFragmentSize**, refer to the *Transport API Developers Guide*.
ConnectionPingTimeout	UInt64	30000	Specifies the duration (in milliseconds) after which the EMA terminates the connection if it does not receive communication or pings from the server.
SysRecvBufSize	UInt64	0	Specifies the size (in KB) of the system's receive buffer for this channel.
SysSendBufSize	UInt64	0	Specifies the size (in KB) of the system's send buffer for this channel.
XmlTraceFileName	EmaString	EmaTrace	Sets the name of the file to which to write XML trace output if tracing is selected.
XmlTraceToFile	UInt64	0	Sets whether EMA 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.
XmlTraceMaxFileSize	UInt64	100000000	Specifies the maximum size (in bytes) for the trace file.
XmlTraceToStdout	UInt64	0	Specifies whether EMA traces its messages in XML format to stdout. Possible values are: • **0** (false): Turns off tracing. • **1** (true): Turns on tracing to stdout.

Table 6: Universal <Channel> Parameters

PARAMETER NAME	TYPE	DEFAULT	NOTES
XmlTraceToMultipleFiles	UInt64	0	Specifies whether to write the XML trace to multiple files. Possible values are: • **1** (true): EMA writes the XML trace to a new file if the current file size reaches the **XmlTraceMaxFileSize**. • **0** (false): EMA stops writing the XML trace if the current file reaches the **XmlTraceMaxFileSize**.
XmlTraceWrite	UInt64	1	Sets the EMA to trace outgoing data. Possible values are: • **0** (false): Do not trace outgoing data. • **1** (true): Trace outgoing data.
XmlTraceRead	UInt64	1	Sets the EMA to trace incoming data. Possible values are: • **0** (false): Do not trace incoming data. • **1** (true): Trace incoming data
InterfaceName	EmaString	""	Specifies a character representation of the IP address or hostname of the local network interface over which the EMA sends and receives content. **InterfaceName** is for use in systems that have multiple network interface cards. If unspecified, the default network interface is used.
ReconnectAttemptLimit	Int64	-1	Specifies the maximum number of times the **OmmConsumer** attempts to reconnect to a channel when it fails. If set to **-1**, the **OmmConsumer** continually attempts to reconnect.
ReconnectMinDelay	Int64	1000	Specifies the minimum amount of time the **OmmConsumer** waits (in millseconds) before attempting to reconnect a failed channel. The time **OmmConsumer** waits between each connection attempt increases with each attempt, from **reconnectMinDelay** to **reconnectMaxDelay**.
ReconnectMaxDelay	Int64	5000	The maximum amount of time the **OmmConsumer** waits (in milliseconds) before attempting to reconnect a failed channel. Refer also to the preceding **ReconnectMinDelay** parameter.

Table 6: Universal <Channel> Parameters (Continued)

3.2.3 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	UInt64	30	Sets the message size threshold (in bytes, the allowed value is 30-UInt32 MAX), above which all messages are compressed (thus individual messages might not be compressed). Different compression types have different behaviors and compression efficiency can vary depending on message size.
CompressionType	Enumeration	None	Specifies the EMA's preferred type of compression. Compression is negotiated between the client and server: if the server supports the preferred compression type, the server will compress data at that level. Use enumeration values with EMA's programmatic configuration (for details, refer to in Section 4.4) Available values are: • **None** (0) • **ZLib** (1) • **LZ4** (2) **Note:** A server can be configured to force a particular compression type, regardless of client settings.
Host	EmaString	localhost	Specifies the host name of the server to which the EMA 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 EMA connects.
TcpNodelay	UInt64	1	Specifies whether to use Nagle's algorithm when sending data. Available values are: • **0**: Send data using Nagle's algorithm. • **1**: Send data without delay.

Table 7: Parameters for Channel Type: RSSL_SOCKET

3.2.4 Parameters for Use with Channel Types: RSSL_HTTP or RSSL_ENCRYPTED

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 either **RSSL_HTTP** or **RSSL_ENCRYPTED**.

PARAMETER NAME	TYPE	DEFAULT	NOTES
CompressionThreshold	UInt64	30	Sets the message size threshold (in bytes, the allowed value is 30-UInt32 MAX), above which all messages are compressed (thus individual messages might not be compressed). Different compression types have different behaviors and compression efficiency can vary depending on message size.
CompressionType	Enumeration	None	Specifies the EMA's preferred type of compression. Compression is negotiated between the client and server: if the server supports the preferred compression type, the server will compress data at that level. Use enumeration values with EMA's programmatic configuration (for details, refer to in Section 4.4) Available values are: • **None** (0) • **ZLib** (1) • **LZ4** (2) **Note:** A server can be configured to force a particular compression type, regardless of client settings.
Host	EmaString	localhost	Specifies the host name of the server to which the EMA connects. The parameter value can be a remote host name or IP address.
ObjectName	EmaString	""	Specifies the object name to pass along with the underlying URL in HTTP connection messages.
Port	EmaString	14002	Specifies the port on the remote server to which the EMA connects.
TcpNodelay	UInt64	1	Specifies whether to use Nagle's algorithm when sending data. Available values are: • **0**: Send data using Nagle's algorithm. • **1**: Send data without delay.

Table 8: Parameters for Channel Types: RSSL_HTTP or RSSL_ENCRYPTED

3.2.5 Parameters for Use with Channel Type: RSSL_RELIABLE_MCAST

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_RELIABLE_MCAST**.

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

Additionally several parameters are designed for use with a TREP infrastructure tool called **rrdump**. **rrdump** is a monitoring utility available in the TREP Infrastructure Tools package. For more information on **rrdump**, refer to either of the *ADS* and *ADH Software Installation Manual*s.

PARAMETER NAME	TYPE	DEFAULT	NOTES
RecvAddress	EmaString	""	Specifies the multicast address to which this channel connects for receiving data.
RecvPort	EmaString	""	Specifies the multicast port to which this channel connects for receiving data.
SendAddress	EmaString	""	Specifies the multicast address to which this channel connects for sending data.
SendPort	EmaString	""	Specifies the multicast port to which this channel connects for sending data.
UnicastPort	EmaString	""	Port to which this connection connects for unicast messages (i.e., ack/nak messages and any retransmit messages). This value also configures a TCP listening port for use with the **rrdump** tool.
PacketTTL	UInt64	5	Sets the lifespan (in hops) of the data packet through the multicast network, which can prevent the packet from circulating indefinitely. It has a range of **0** - **255**. • **0** means the message can be sent only to other applications on the same machine. • A value of **255** sets the message to travel through the network indefinitely.
HsmInterface	EmaString	""	Specifies the Host Status Message (HSM) interface. By default, **HsmInterface** is set to the host machine's default interface.
HsmInterval	UInt64		The interval (in seconds) over which HSM packets are sent. You can use **rrdump** to change the value of **hsmInterval**. Thus, after starting the application, you can stop and restart HSM publication as needed. The default interval is **0** (disabled) which suspends host status message publication.
HsmMultiAddress	EmaString	""	Specifies the multicast address over which this channel sends HSM packets. EMA configuration allows for the use of defined aliases.
HsmPort	EmaString	""	Specifies the multicast port to which this channel sends HSM packets.
tcpControlPort	EmaString	""	Specifies the port to use for the RRCP **tcpControlPort**. This port is used when troubleshooting RRCP using the **rrdump** tool. A setting of **-1** disables **tcpControlPort**.

Table 9: Parameters for Channel Type: RSSL_RELIABLE_MCAST

PARAMETER NAME	TYPE	DEFAULT	NOTES
DisconnectOnGap	UInt64	0	Specifies whether the underlying connection should be closed if a multicast gap situation is detected. • **0** (false): **0** is the default value which means the underlying connection is not closed if a multicast gap situation occurs. • **1** (true): Sets the underlying connection to close if a multicast gap situation occurs.
ndata	UInt64	7	Specifies the maximum number of retransmissions to attempt for an unacknowledged point-to-point packet.
nmissing	UInt64	128	Specifies the maximum number of missed consecutive multicast packets, from a particular node, from which RRCP requests retransmits.
nrreq	UInt64	3	Specifies the maximum number of retransmit requests that can be sent for a missing packet.
tdata	UInt64	1	Specifies the time that RRCP waits before retransmitting an unacknowledged point-to-point data message. **tdata** is specified in RRCP clock ticks of 100 milliseconds, thus a value of 2 means 200 milliseconds.
trreq	UInt64	4	Specifies the amount of time that RRCP waits before "resending" a retransmit request for a missed multicast packet. **trreq** is specified in RRCP clock ticks (100 milliseconds), so a value of 2 means 200 milliseconds.
twait	UInt64	3	Specifies the duration of time for which RRCP ignores additional retransmit requests for a data packet that it has already retransmitted. This time period starts with the receipt of the first request for retransmission. **twait** is specified in RRCP clock ticks (100 milliseconds), so a value of 2 means 200 milliseconds.
tbchold	UInt64	3	Specifies the maximum time that RRCP holds a transmitted broadcast packet in case the packet needs to be retransmitted. **tbchold** is specified in RRCP clock ticks (100 milliseconds), so a value of 2 means 200 milliseconds.
tpphold	UInt64	3	Specifies the maximum time that RRCP holds a transmitted point-to-point packet in case the packet needs to be retransmitted. **tpphold** is specified in RRCP clock ticks (100 milliseconds), so a value of 2 means 200 milliseconds.
pktPoolLimitHigh	UInt64	190000	Specifies the high-water mark for the RRCP packet pool. If this limit is reached, no further RRCP packets are allocated until usage falls below the low-water mark (as set by **pktPoolLimitLow**).

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

PARAMETER NAME	TYPE	DEFAULT	NOTES
pktPoolLimitLow	UInt64	180000	Specifies the low-water mark for the RRCP packet pool. If RRCP packet allocation gets frozen (due to **pktPoolLimitHigh** having been reached), additional RRCP packets are allocated only when usage falls below the **pktPoolLimitLow** setting. **pktPoolLimitLow** should be greater than **3 * userQLimit**.
userQLimit	UInt64	65535	Specifies the maximum backlog of messages allowed on an application's inbound message queue. If **userQLimit** is exceeded, the RRCP protocol engine begins to discard messages for that application until the backlog decreases.

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

3.2.6 Example XML Schema for Configuring ChannelSet

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

```
<ConsumerGroup>
   <ConsumerList>
      <Consumer>
         <Name value="Consumer_1"/>
         <!-- ChannelSet is optional  -->
         <ChannelSet value="Channel_1, Channel_2"/>
         <!-- Logger is optional: defaulted to "File + Success" -->
         <Logger value="Logger_1"/>
         <!-- Dictionary is optional: defaulted to "ChannelDictionary" -->
         <Dictionary value="Dictionary_1"/>
      </Consumer>
   </ConsumerList>
</ConsumerGroup>
<ChannelGroup>
   <ChannelList>
      <Channel>
         <Name value="Channel_1"/>
         <ChannelType value="ChannelType::RSSL_SOCKET"/>
         <Host value="localhost"/>
         <Port value="14002"/>
      </Channel>
      <Channel>
         <Name value="Channel_2"/>
         <ChannelType value="ChannelType::RSSL_SOCKET"/>
         <Host value="122.1.1.100"/>
         <Port value="14008"/>
      </Channel>
   </ChannelList>
</ChannelGroup>
```

3.2.7 Example Programmatic Configuration for ChannelSet

The following is an example programmatic **channelSet** configuration. In this example, the consumer uses the **Channel** parameters **reconnectAttemptLimit**, **reconnectMinDelay**, **reconnectMaxDelay**, **xmlTraceFileName**, **xmlTraceMaxFileSize**, **xmlTraceToFile**, **xmlTraceToStdout**, **xmlTraceToMultipleFiles**, **xmlTraceWrite**, **xmlTraceRead** and **msgKeyInUpdates** of the last channel. Though each channel in the **ChannelSet** can have different values for these parameters, EMA uses parameter values as set for the last channel in the set (even if the consumer successfully connects to a different channel in the **ChannelSet**).

```
Map configMap;
Map innerMap;
ElementList elementList;
elementList.addAscii( "DefaultConsumer", "Consumer_1" );
innerMap.addKeyAscii( "Consumer_1", MapEntry::AddEnum, ElementList()
.addAscii( "ChannelSet", "Channel_1, Channel_2" )
```

```
.addAscii( "Logger", "Logger_1" )
.addAscii( "Dictionary", "Dictionary_1" ).complete() ).complete();
elementList.addMap( "ConsumerList", innerMap );
elementList.complete();
innerMap.clear();
configMap.addKeyAscii( "ConsumerGroup", MapEntry::AddEnum, elementList );
elementList.clear();
innerMap.addKeyAscii( "Channel_1", MapEntry::AddEnum, ElementList()
.addEnum( "ChannelType", 0 )
.addAscii( "InterfaceName", "localhost" )
.addAscii( "Host", "localhost" )
.addAscii("Port", "14002" ).complete() )
innerMap.addKeyAscii( "Channel_2", MapEntry::AddEnum, ElementList()
.addEnum( "ChannelType", 0 )
.addAscii( "InterfaceName", "localhost" )
.addAscii( "Host", "121.1.1.100" )
.addAscii("Port", "14008" ).complete() ).complete();
elementList.addMap( "ChannelList", innerMap );
elementList.complete();
innerMap.clear();
configMap.addKeyAscii( "ChannelGroup", MapEntry::AddEnum, elementList );
elementList.clear();

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

3.3 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 EMA.

PARAMETER NAME	TYPE	DEFAULT	NOTES
Name	EmaString		Sets a unique name for the Logger component in the LoggerList.
LoggerType	Enumeration	File	Specifies the logging mechanism. Use enumeration values with EMA's programmatic configuration (for details, refer to in Section 4.4) Possible values are: • **0**: EMA logs to the file specified in the parameter **FileName**. • **1**: EMA logs to stdout.
LoggerSeverity	Enumeration	Success	Severity levels aggregate messages so that a severity level includes all messages from higher levels (e.g., a setting of **1** includes any messages normally printed at levels **2** and **3**). Use enumeration values with EMA's programmatic configuration (for details, refer to in Section 4.4) Possible values are: • Verbose (0) • Success (1) • Warning (2) • Error (3) • NoLogMsg (4)
FileName	EmaString	"emaLog_*pid*.log"	The EMA ignores this parameter if **LoggerType** is set to **Stdout** (1).
IncludeDateInLoggerOutput	UInt64	0	Sets whether to include the date in EMA's log messages. Possible values are: • **0** (false): Include only the time, omitting the date. • **1** (true): Include both date and time.

Table 10: Logger Group Parameters

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 EMA.

PARAMETER NAME	TYPE	DEFAULT	NOTES
Name	EmaString		Sets a unique name for a Dictionary component in the DictionaryList.
DictionaryType	Enumeration	ChannelDictionary	Specifies the dictionary loading mode. Use enumeration values with EMA's programmatic configuration (for details, refer to in Section 4.4) Possible values are: • **FileDictionary** (0): The EMA loads the dictionaries from the files specified in the parameters **RdmFieldDictionaryFileName** and **EnumTypeDefFileName**. • **ChannelDictionary** (1): The EMA downloads dictionaries by requesting the dictionaries from the upstream provider.
RdmFieldDictionaryFileName	EmaString		Sets the location of the **RdmFieldDictionary**.
EnumTypeDefFileName	EmaString		Sets the location of the **EnumTypeDef** file.

Table 11: Dictionary Group Parameters

Chapter 4 EMA Configuration Processing

4.1 Default Configuration

The EMA configuration is determined by hard-coded behaviors, any customized beahviors specified in **EmaConfig.xml**, programmatic changes, and other internal processing. All of these items affect the configuration used by application components. This chapter discusses how the application configuration is derived.

Each application must eventually instantiate an ***OmmConsumer*** object. Constructors for OmmConsumer require a ***OmmConsumerConfig*** object. The ***OmmConsumerConfig*** constructor takes no arguments, but it does read and process an optional XML file (**EmaConfig.xml**), which applications can use to modify EMA's default behavior.

EMA provides a hard-coded configuration for use whenever an **OmmConsumerConfig** object is instantiated without an **EmaConfig.xml** file in the run-time environment. The resulting EMA 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"
- **XmlTraceToFile** value="0"

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

For specifics on EMA's default configuration, refer to Section 2.2.

4.2 Processing EmaConfig.xml

Except for the parameter **DefaultConsumer**, all configuration elements defined in the**EmaConfig.xml** file must be wrapped within a component definition (i.e., **Consumer**, **Logger**, **Channel**, or **Dictionary**) or they will be ignored. This section includes some examples that illustrate this requirement. Appendix A illustrates the proper placement of **DefaultConsumer** within **EmaConfig.xml**.

4.2.1 Use of the Correct Order in the XML Scema

Consider the following snippet from an **EmaConfig.xml** (only those parts needed for the example are included). In this snippet, the application creates a consumer with **Name Consumer_1** which logs to a file named **emaLogfile**.

```
<ConsumerGroup>
   <ConsumerList>
      <Consumer>
         <Name value="Consumer_1"/>
         <Logger value="Logger_2"/>
      </Consumer>
```

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

Now assume that the following was not included in **EmaConfig.xml**:

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

In this case, the EMA application relies on its hard-coded behavior and uses the filename **emaLog_*pid*.log**.

However, if the snippet were configured in either of the following configurations, the EMA application would revert to its default behaviors because its parameters are not in the correct order (i.e., the **FileName** parameter needs to be contained in a **Logger** component entry):

- Configuration 1:

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

- Configuration 2:

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

4.2.2 Processing the Consumer “Name”

The EMA is hard-coded to use a default consumer of **EmaConsumer**. However, you can change this by using **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 EMA 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 EMA throws an exception indicating that **DefaultConsumer** is invalid.
- If the **EmaConfig.xml** has no **Consumer** components, the EMA application uses **EmaConsumer**.

4.3 Configuring EMA Using Function Calls

From an application standpoint, instantiating an OmmConsumerConfig object creates the initial configuration from the **DefaultXML.h** and **EmaConfig.xml** files. Certain variables can then be altered via function calls on the OmmConsumerConfig object.

Note: Function calls override any settings in the **EmaConfig.xml** file.

4.3.1 EMA Function Calls

You can use the following function calls in an EMA application:

FUNCTION	DESCRIPTION
username(const EmaString &)	Sets the **username** variable. If **username** is not set, the application extracts a username from the run-time environment.
password(const EmaString &)	Sets the **password** variable. **password** has no default value.
position(const EmaString &)	Sets the **position** variable. **position** has no default value.
applicationId(const EmaString &)	Sets the **applicationId** variable. **applicationId** has no default value.
host(const EmaString &)	Sets the host and port parameters. For details, refer to Section 4.3.2.
operationModel(OperationModel)	Sets the operation model to either `OmmConsumerConfig::ApiDispatchEnum` (which is the default) or `OmmConsumerConfig::UserDispatchEnum`.
consumerName(const EmaString &)	Sets the consumer name. If a consumer does not exist with that name, the application throws an exception.
addAdminMsg(const ReqMsg&)	Populates part of or all of the login request message, directory request message, or dictionary request message according to the specification discussed in the *EMA C++ Edition Reuters Domain Models (RDM) Usage Guide*.

Table 12:

4.3.2 Using the `host( )` Function: How "Host" and "Port" are Processed

Because the **Host** and **Port** parameters both have global default values (**localhost** and **14002** respectively), if an OmmConsumerConfig object exists, its **Host** and **Port** will always have values (either the default value or some value specified within **EmaConfig.xml**).

If you wish, you can have the application reset both host and port values by calling the **`host( const EmaString& )`** method on the OmmConsumerConfig object using the syntax: ***HostValue*:*PortValue***.

Note: Calling the **`host()`** function results in the **channelType** (refer to Section 3.2.2) being set to **RSSL_SOCKET**, regardless of any previous setting for that configuration element.

Host and **Port** values observe the following rules when updating due to the **`host( const EmaString& )`** method:

- If the host parameter is missing or empty, then host and port reset to their global default values (**localhost:14002**)
- If the host parameter is set to the string **":"**, then host and port reset to their global default values (**localhost:14002**)
- If the host parameter is a string (not containing a **:**), then host is set to that string and port resets to its default value (**14002**).
- If the parameter begins with a **:** and is followed by some text, then host is set to its global default value (**localhost**) 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 Programmatic Configuration

In addition to changing EMA's configuration via **EmaConfig.xml** or function calls, you can do so programmatically via an OMM data structure.

4.4.1 OMM Data Structure

Programmatic configuration of EMA provides a way of configuring all parameters and overriding parameters configured in **EmaConfig.xml** using an OMM data structure, which is divided into four tiers:

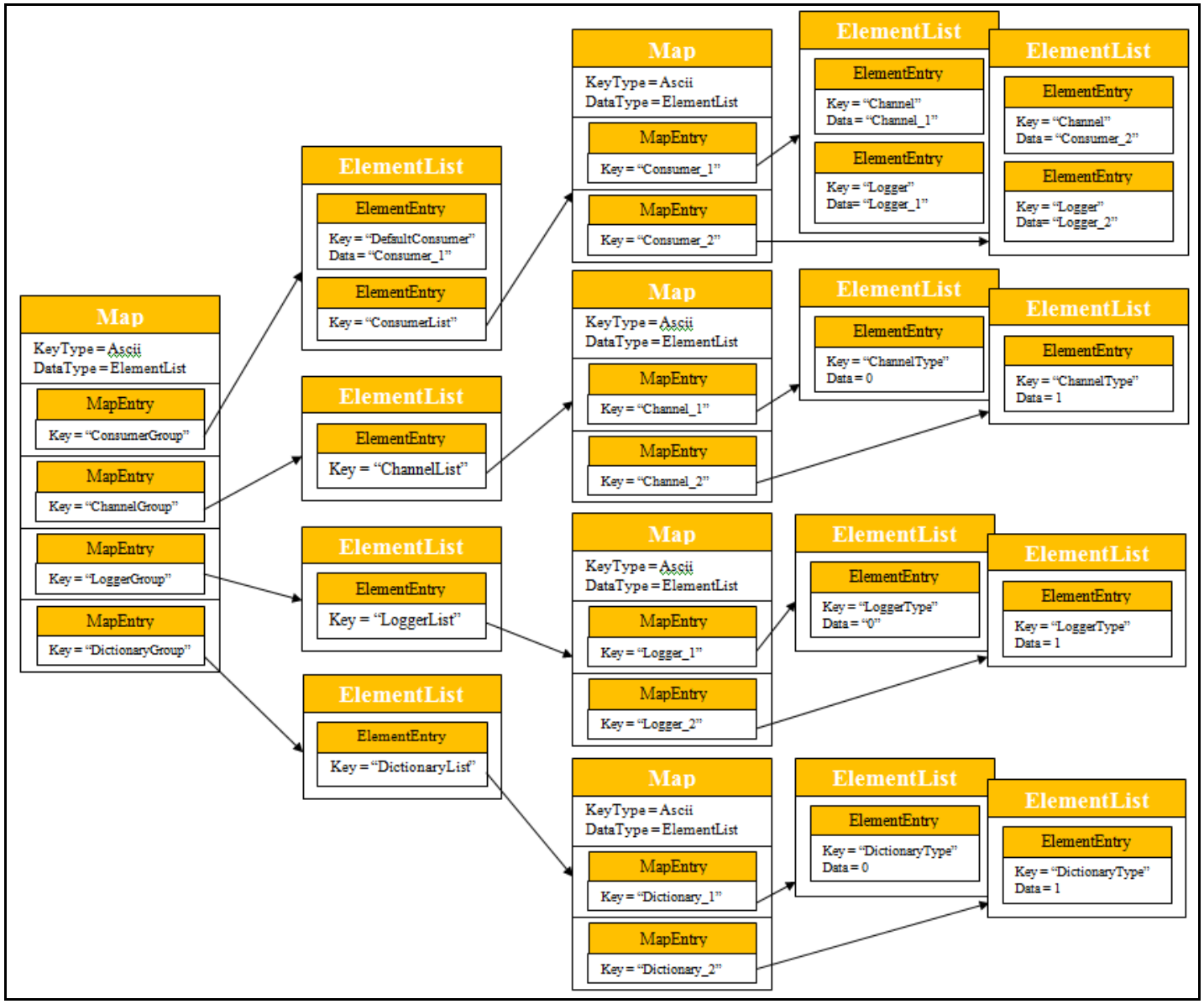
- The 1st tier lists EMA's Consumer, Channel, Logger, and Dictionary components; each of which has it owns list in the 2nd tier.
- The 2nd tier includes each component's list and the default consumer to use when loading configuration parameters.
- The 3rd tier defines individual names for these components,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

4.4.2 Creating the Configuration

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

To programmatically configure EMA:

1. Create a map with the following hierarchy to configure EMA 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.

4.4.3 Example: Programmatic Configuration

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

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

innerMap.addKeyAscii( "Consumer_1", MapEntry::AddEnum, ElementList()
    .addAscii( "Channel", "Channel_1" )
    .addAscii( "Logger", "Logger_1" )
    .addAscii( "Dictionary", "Dictionary_1" )
    .addUInt( "ItemCountHint", 5000 )
    .addUInt( "ServiceCountHint", 5000 )
    .addUInt( "ObeyOpenWindow", 0 )
    .addUInt( "PostAckTimeout", 5000 )
    .addUInt( "RequestTimeout", 5000 )
    .addUInt( "MaxOutstandingPosts", 5000 )
    .addInt( "DispatchTimeoutApiThread", 1 )
    .addUInt( "HandleException", 0 )
    .addUInt( "MaxDispatchCountApiThread", 500 )
    .addUInt( "MaxDispatchCountUserThread", 500 )
    .addInt( "ReactorEventFdPort", 45000 )
    .addInt( "PipePort", 4001 ).complete() ).complete();

elementList.addMap( "ConsumerList", innerMap );

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

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

innerMap.addKeyAscii( "Channel_1", MapEntry::AddEnum, ElementList()
    .addEnum( "ChannelType", 0 )
    .addAscii( "InterfaceName", "localhost" )
    .addEnum("CompressionType", 1)
    .addUInt( "GuaranteedOutputBuffers", 5000 )
    .addUInt( "ConnectionPingTimeout", 50000 )
    .addInt("ReconnectAttemptLimit", 10)
    .addInt( "ReconnectMinDelay", 2000 )
    .addInt( "ReconnectMaxDelay", 6000 )
    .addAscii( "Host", "localhost" )
    .addAscii("Port", "14002" )
    .addUInt( "TcpNodelay", 0 )
    .addAscii( "XmlTraceFileName", "MyXMLTrace" )
    .addInt( "XmlTraceMaxFileSize", 50000000 )
    .addUInt( "XmlTraceToFile", 1 )
    .addUInt( "XmlTraceToStdout", 0 )
```

```
    .addUInt( "XmlTraceToMultipleFiles", 1 )
    .addUInt( "XmlTraceWrite", 1 )
    .addUInt( "XmlTraceRead", 1 )
    .addUInt( "MsgKeyInUpdates", 1 ).complete() ).complete();

elementList.addMap( "ChannelList", innerMap );

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

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

elementList.addMap( "LoggerList", innerMap );

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

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

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

elementList.addMap( "DictionaryList", innerMap );

elementList.complete();

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

configMap.complete();
```

Appendix A EmaConfig.xml Configuration File

This is the current version of the EmaConfig.xml file distributed with the training examples:

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

<ConsumerGroup>
   <!-- DefaultConsumer parameter defines which consumer configuration is used by OmmConsumer -->
   <!-- if application does not specify it through OmmConsumerConfig::consumerName() -->
   <!-- first consumer on the ConsumerList is a default consumer if this parameter is not specified -->
   <DefaultConsumer value="Consumer_1"/>
   <ConsumerList>
      <Consumer>
         <Name value="Consumer_1"/>

         <!-- Channel is optional: defaulted to "RSSL_SOCKET + localhost + 14002" -->
         <Channel value="Channel_1"/>
         <!-- Logger is optional: defaulted to "File + Success" -->
         <Logger value="Logger_1"/>

         <!-- Dictionary is optional: defaulted to "ChannelDictionary" -->
         <Dictionary value="Dictionary_1"/>
      </Consumer>
      <Consumer>
         <Name value="Consumer_2"/>
         <Channel value="Channel_2"/>
         <Logger value="Logger_2"/>
         <Dictionary value="Dictionary_2"/>
      </Consumer>
   </ConsumerList>
</ConsumerGroup>
<ChannelGroup>
   <ChannelList>
      <Channel>
         <Name value="Channel_1"/>

         <!-- ChannelType possible values are: -->
         <!-- ChannelType::RSSL_SOCKET    - TCP IP connection type -->
         <!-- ChannelType::RSSL_HTTP      - Http tunnel connection type -->
         <!-- ChannelType::RSSL_ENCRYPTED - Https tunnel connection type -->
         <ChannelType value="ChannelType::RSSL_SOCKET"/>

         <!-- CompressionType is optional: defaulted to None -->
         <!-- possible values: None, ZLib, LZ4 -->
         <CompressionType value="CompressionType::None"/>
         <GuaranteedOutputBuffers value="5000"/>
```

```
            <!-- ConnectionPingTimeout is optional: defaulted to 30000 -->
            <ConnectionPingTimeout value="30000"/>

            <!-- TcpNodelay is optional: defaulted to 1 -->
            <!-- possible values: 1 (tcp_nodelay option set), 0 (tcp_nodelay not set) -->
            <TcpNodelay value="1"/>
            <Host value="localhost"/>
            <Port value="14002"/>
        </Channel>
        <Channel>
            <Name value="Channel_2"/>
            <ChannelType value="ChannelType::RSSL_SOCKET"/>
            <CompressionType value="CompressionType::None"/>
            <GuaranteedOutputBuffers value="5000"/>
            <Host value="122.1.1.100"/>
            <Port value="14002"/>
        </Channel>
        <Channel>
            <Name value="Channel_3"/>
            <ChannelType value="ChannelType::RSSL_ENCRYPTED"/>
            <CompressionType value="CompressionType::None"/>
            <GuaranteedOutputBuffers value="5000"/>
            <Host value="122.1.1.100"/>
            <Port value="14002"/>
        </Channel>
        <Channel>
            <Name value="Channel_4"/>
            <ChannelType value="ChannelType::RSSL_HTTP"/>
            <CompressionType value="CompressionType::None"/>
            <GuaranteedOutputBuffers value="5000"/>
            <Host value="122.1.1.100"/>
            <Port value="14002"/>
        </Channel>
    </ChannelList>
</ChannelGroup>
<LoggerGroup>
    <LoggerList>
        <Logger>
            <Name value="Logger_1"/>

            <!-- LoggerType is optional:  defaulted to "File" -->
            <!-- possible values: Stdout, File -->
            <LoggerType value="LoggerType::Stdout"/>

            <!-- LoggerSeverity is optional: defaulted to "Success" -->
            <!-- possible values: Verbose, Success, Warning, Error, NoLogMsg -->
            <LoggerSeverity value="LoggerSeverity::Success"/>
        </Logger>
        <Logger>
            <Name value="Logger_2"/>
            <LoggerType value="LoggerType::File"/>
```

```
        <!-- FileName is optional: defaulted to "emaLog_ProcessId.log" -->
        <FileName value="emaLog"/>
        <LoggerSeverity value="LoggerSeverity::Success"/>
      </Logger>
   </LoggerList>
</LoggerGroup>
<DictionaryGroup>
   <DictionaryList>
      <Dictionary>
         <Name value="Dictionary_1"/>
         <!-- DictionaryType is optional: defaulted to ChannelDictionary" -->
         <!-- possible values: FileDictionary, ChannelDictionary -->
         <!-- if DictionaryType is set to ChannelDictionary, file names are ignored -->
         <DictionaryType value="DictionaryType::ChannelDictionary"/>
      </Dictionary>
      <Dictionary>
         <Name value="Dictionary_2"/>
         <DictionaryType value="DictionaryType::FileDictionary"/>

         <!-- dictionary names are optional: defaulted to RDMFieldDictionary and enumtype.def -->
         <RdmFieldDictionaryFileName value="./RDMFieldDictionary"/>
         <EnumTypeDefFileName value="./enumtype.def"/>
      </Dictionary>
   </DictionaryList>
</DictionaryGroup>
</EmaConfig>
```

Document ID: EMAC301CG.150
Date of issue: 13 November 2015