We are now Refinitiv, formerly the Financial and Risk business of Thomson Reuters. We've set a bold course for the future – both ours and yours – and are introducing our new brand to the world.

As our brand migration will be gradual, you will see traces of our past through documentation, videos, and digital platforms.

Thank you for joining us on our brand journey.

REFINITIV™

Elektron Message API C++ Edition V3.2.X

ELEKTRON MESSAGE API CONFIGURATION GUIDE

Document Version: 3.2.1
Date of issue: 18 August 2018
Document ID: EMAC321CG.180

THOMSON REUTERS

Contents

Chapter 1 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 that 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 C++ Edition API Developer's Guide*.

1.3 Definitions

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

Table 1: Definitions

1.4 Acronyms and Abbreviations

ACRONYM	MEANING
ADH	Advanced Data Hub is the horizontally scalable service component within Thomson Reuters Enterprise Platform (TREP) providing high availability for publication and contribution messaging, subscription management with optional persistence, conflation and delay capabilities.
ADS	Advanced Distribution Server is the horizontally scalable distribution component within Thomson Reuters Enterprise Platform (TREP) 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
EED	Elektron Edge Device
EMA	Elektron Message API, referred to simply as the Message API
ETA	Elektron Transport API, referred to simply as the Transport API. Formerly referred to as UPA.
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol (Secure)
OMM	Open Message Model
QoS	Quality of Service
RDM	Reuters Domain Model
Reactor	The Reactor is a low-level, open-source, easy-to-use layer above ETA. It offers heartbeat management, connection and item recovery, and many other features to help simplify application code for users.
RMTES	Reuters Multi-Lingual Text Encoding Standard
RSSL	Reuters Source Sink Library
RWF	Reuters Wire Format, a Thomson Reuters proprietary format.
TR-DFD	Thomson Reuters Data Feed Direct
TREP	Thomson Reuters Enterprise Platform
UML	Unified Modeling Language
UTF-8	8-bit Unicode Transformation Format

Table 2: Acronyms and Abbreviations

1.5 References

1. Elektron Message API C++ Edition *RDM Usage Guide*
2. *API Concepts Guide*
3. Elektron Message API C++ Edition *Developers Guide*
4. Transport API C Edition *Value Added Components Developers Guide*
5. *Transport API C Edition Developers Guide*
6. The Thomson Reuters Professional Developer Community

1.6 Documentation Feedback

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

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

1.7 Document Conventions

This document uses the following types of conventions:

- Typographic
- Data Types
- Field and Text Values

1.7.1 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();
```

1.7.2 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

1.7.3 Field and Text Values

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

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

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

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

1.7.4 Boolean Values

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

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

Chapter 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 is the highest-level description of the application. Such settings typically select entries from the channel, logger, and dictionary groups.
- ***NiProvider***: Non-interactive provider configuration data is the highest-level description of the application. Such settings typically select entries from the channel, logger, and directory 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.
- ***Directory***: Directory configuration data configures source directory refresh information.

The Consumer and NiProvider groups are top-level configuration groups. Specific consumer and non-interactive provider applications select their configurations using the consumer and non-interactive provider names, which are passed in using the `consumerName()` and `providerName()` methods (for details on these methods, refer to Section 4.4.1).

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

2.2 Parameter Overview

Many default behaviors are hard-coded into the 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 2.3.

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

2.3 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 (for consumers) 14003 (for non-interactive providers)	Specifies the port number on the server to which the application connects.
DefaultConsumer	EmaString	EmaConsumer	If consumer components are configured, this parameter is ignored.
DefaultNiProvider	EmaString	EmaNiProvider	If non-interactive provider 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 3.5.2.
LoggerType	Enumeration	File	Specifies the destination for output messages. The parameter value can be either `File` or `Stdout`. For details on selecting a loggerType and its enumerations, refer to Section 3.5.2.
FileName	EmaString	“emaLog_*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

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 **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 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 **Consumer** parameters, refer to Section 3.1.4).

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

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

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

3.1.4 Consumer Entry Parameters

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

PARAMETER	TYPE	DEFAULT	DESCRIPTION
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.
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 3.3.
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 3.3.6. **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>**.

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 3.6.
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 will wait for a response indefinitely.
			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 will wait for a response indefinitely.
			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.
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.
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.
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 3.5.

Table 5: Consumer Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
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 will wait for a response indefinitely. **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.
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()`.
MaxOutstandingPosts	UInt64	100,000	Specifies the maximum allowable number of on-stream posts waiting for an acknowledgment before the **OmmConsumer** disconnects.
MsgKeyInUpdates	UInt64	1	Specifies whether EMA fills in message key values on updates using the message key provided with the request. Available values include: • **0** (false): Do not fill in the message's key values (values received from the wire are preserved). • **1** (true): Fill in the message's key values (values received from the wire are overridden).
Name	EmaString	N/A	Specifies the name of this **Consumer** component. **Name** is required when creating a **Consumer** component. You can use any value for **Name**.
ObeyOpenWindow	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)
PipePort	Int64	9001	Specifies the internal communication port. You might need to adjust this port if it conflicts with other processes on the machine.
PostAckTimeout	UInt64	15,000	Specifies the length of time (in milliseconds) a stream waits to receive an ACK for an outstanding post before forwarding a negative acknowledgment to the application. If set to **0**, EMA will wait for a response indefinitely.
ReconnectAttemptLimit	Int64	-1	Specifies the maximum number of times the consumer and non-interactive provider attempt to reconnect to a channel when it fails. If set to **-1**, the consumer and non-interactive provider continually attempt to reconnect.

Table 5: Consumer Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ReconnectMaxDelay	Int64	5000	Sets the maximum amount of time the consumer and non-interactive provider wait (in milliseconds) before attempting to reconnect a failed channel. Refer also to the **ReconnectMinDelay** parameter.
ReconnectMinDelay	Int64	1000	Specifies the minimum amount of time the consumer and non-interactive provider wait (in milliseconds) before attempting to reconnect a failed channel. This wait time increases with each connection attempt, from **reconnectMinDelay** to **reconnectMaxDelay**.
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 will wait for a response indefinitely.
ServiceCountHint	UInt64	513	Sets the size of directory structures for managing services. If the application specifies **0**, EMA resets it to **513**.
XmlTraceFileName	EmaString	EmaTrace	Sets the name of the file to which to write XML trace output if tracing is selected.
XmlTraceHex	UInt64	0	Sets whether to print incoming and outgoing messages in hexadecimal format. Possible values are: • **0** (false): Do not print messages in hexadecimal format. • **1** (true): Print messages in hexadecimal format.
XmlTraceMaxFileSize	UInt64	100000000	Specifies the maximum size (in bytes) for the trace file.
XmlTracePing	UInt64	0	Sets the EMA to trace incoming and outgoing ping messages. Possible values are: • **0** (false): Do not trace ping messages. • **1** (true): Trace ping messages.
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
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.
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**.

Table 5: Consumer Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
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.
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.

Table 5: Consumer Group Parameters (Continued)

3.2 Provider Groups

The EMA supports both interactive and non-interactive provider groups. The type of provider you configure will determine the group names and parameters that you use. To simplify the content in this guide, parameters and names use the variable ***Provider***. Throughout this section (and the remainder of the manual), the value for ***Provider*** is dependent on the type of provider which you configure:

- For non-interactive providers, ***Provider*** is **NiProvider**.
- For interactive providers, ***Provider*** is **IProvider**.

A ***Provider*Group** contains two elements:

- A **Default*Provider*** element, which you can use to specify a default **NiProvider** component. If a default ***Provider*** is not specified in the ***Provider*Group**, EMA uses the first non-interactive provider listed in the ***Provider*List**. For details on configuring a default ***Provider***, refer to Section 3.2.2.
- A ***Provider*List** element, which contains one or more ***Provider*** components (each should be uniquely identified by a **<Name …/>** entry). The non-interactive provider component is the highest-level abstraction within an application and typically refers to **Channel** (used by non-interactive providers), **Server** (used by interactive providers), **Logger**, and/or **Directory** components which specify provider capabilities.

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

For details on configuring a ***Provider*Group**, refer to Section 3.2.3.

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

3.2.1 Generic XML Schema for Provider Group

The generic XML schema for ***Provider*Group** is as follows:

```
<ProviderGroup>
    <DefaultProvider value="VALUE"/>
    <ProviderList>
        <Provider>
            <Name value="VALUE"/>
            ...
        </Provider>
    </ProviderList>
</ProviderGroup>
```

3.2.2 Setting a Default Provider

If a **Default*Provider*** is not specified, then the EMA uses the first ***Provider*** component in the ***Provider*Group**. However, you can specify a default provider by including the following parameter on a unique line inside ***Provider*Group** but outside ***Provider*List** (for an example, refer to Appendix A).

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

3.2.3 Configuring a *Provider* in a *Provider*Group

To configure a ***Provider*** component, add the appropriate parameters to the target provider in the XML schema, each on a unique line (for a list of available ***Provider*** parameters, refer to Section 3.2.4).

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

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

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

3.2.4 Provider Entry Parameters

Use the following parameters when configuring a ***Provider***. Certain parameters can only be used with a specific provider type (e.g., **Channel** can only be used with an **NiProvider**). The parameter's description will mention any provider-type restrictions.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
AcceptDirMessageWithoutMinFilters	UInt64	0	Used only with **IProvider**. Sets the **IProvider** to accept incoming directory request messages without the minimum required INFO and STATE directory filters.
AcceptMessageSameKeyButDiffStream	UInt64	0	Used only with **IProvider**. Sets the **IProvider** to accept incoming request messages even though they have a message key, domain, and private stream flag that match those of an existing request which uses a different stream ID.
AcceptMessageThatChangesService	UInt64	0	Used only with **IProvider**. Sets the **IProvider** to accept incoming request messages for reissuing the service name of an existing item stream.
AcceptMessageWithoutAcceptingRequests	UInt64	0	Used only with **IProvider**. Sets the **IProvider** to accept incoming request messages even though the source directory is not accepting requests.
AcceptMessageWithoutBeingLogin	UInt64	0	Used only with **IProvider**. Sets the **IProvider** to accept incoming request messages even though the interactive provider has not accepted a login request.
AcceptMessageWithoutQosInRange	UInt64	0	Used only with **IProvider**. Sets the **IProvider** to accept incoming request messages even though the requesting QoS is not in the QoS range of the source directory.

Table 6: *Provider*Group Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
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.
Channel	EmaString	N/A	Used only with **NiProvider**. Specifies the channel that the **NiProvider** component should use. This channel must match the **Name** parameter from the appropriate **<Channel>** entry in the **ChannelGroup** configuration. If **Channel** is not specified, the EMA resorts to default channel behavior when needed. For further details on the **<Channel>** entry and default behaviors, refer to Section 3.3.
ChannelSet	EmaString	N/A	Used only with **NiProvider**. Specifies a comma-separated set of channel names. Each channel name must have a corresponding **<Channel>** entry in the **ChannelGroup**. In the event of a reconnection, Channels in the set are tried until a successful connection is made. For further details, refer to Section 3.3.6. **Note:** If both **Channel** and **ChannelSet** are configured, EMA uses the parameter configured last (linearly) in the file. For example: • If **<Channel>** is configured after **<ChannelSet>**, EMA uses **<Channel>**. • If **<ChannelSet>** is configured after **<Channel>**, EMA uses **<ChannelSet>**.
EnumTypeFragmentSize	UInt64	128000	Used only with **IProvider**. Sets the maximum enumeration types fragmentation size (in bytes) for each multi-part refresh message.
FieldDictionaryFragmentSize	UInt64	8192	Used only with **IProvider**. Sets the maximum field dictionary fragmentation size (in bytes) for each multi-part refresh message.

Table 6: *Provider*Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
Server	EmaString	N/A	Used only with **IProvider**. Specifies the channel that the **IProvider** component should use. This channel must match the **Name** parameter from the appropriate **<Server>** entry in the **ServerGroup** configuration. If **Server** is not specified, the EMA resorts to default channel behavior when needed. For further details on the **<Server>** entry and default behaviors, refer to Section 3.4.
Directory	EmaString	N/A	Specifies source directory refresh information that the ***Provider*** sends after establishing a connection (this must match the **Name** parameter from the appropriate **<Directory>** entry in the **DirectoryGroup** configuration). If **Directory** is not specified, the EMA uses a hard coded configuration. For further details on the **<Logger>** entry and default settings, refer to Section 3.7.
DispatchTimeoutApiThread	Int64	-1	Specifies the duration (in microseconds) for which the internal 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.
ItemCountHint	UInt64	100,000	Specifies the number of items the application expects to maintain. If set to **0**, EMA resets it to **513**. For better performance, the application can set this to the approximate number of items it maintains.
Logger	EmaString	N/A	Specifies a set of logging behavior the ***Provider*** should exhibit (it must match the **Name** parameter from the appropriate **<Logger>** entry in the **LoggerGroup** configuration). If **Logger** is not specified, the EMA uses a set of logger default behaviors. For further details on the **<Logger>** entry and default settings, refer to Section 3.5.

Table 6: *Provider*Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
LoginRequestTimeOut	UInt64	45,000	Used only with **NiProvider**. Specifies the amount of time (in milliseconds) the provider has to respond with a login refresh message before the `OmmProvider` throws an exception. If set to **0**, EMA will wait for a response indefinitely. **Note:** When **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.
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 `OmmProvider::dispatch()`.
MergeSourceDirectoryStreams	UInt64	1	Used only with **NiProvider**. Specifies whether EMA merges all source directory streams (configured and user-submitted) into one stream: • **1** (true) • **0** (false)
Name	EmaString	N/A	Specifies the name of this ***Provider*** component. **Name** is required when creating a ***Provider*** component. You can use any value for **Name**.
PipePort	Int64	See Description	Specifies the internal communication port. You might need to adjust this port if it conflicts with other processes on the machine. • NiProvider uses a default of **9001**. • IProvider uses a default of **9002**.
ReconnectAttemptLimit	Int64	-1	Used only with **NiProvider**. Specifies the maximum number of times the consumer and non-interactive provider attempt to reconnect to a channel to a channel when it fails. If set to **-1**, the consumer and non-interactive provider continually attempt to reconnect.
ReconnectMaxDelay	Int64	5000	Used only with **NiProvider**. Sets the maximum amount of time the consumer and non-interactive provider wait (in milliseconds) before attempting to reconnect a failed channel. Refer also to the **ReconnectMinDelay** parameter.

Table 6: *Provider*Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ReconnectMinDelay	Int64	1000	Used only with **NiProvider**. Specifies the minimum amount of time the consumer and non-interactive provider wait (in milliseconds) before attempting to reconnect a failed channel. This wait time increases with each connection attempt, from **reconnectMinDelay** to **reconnectMaxDelay**.
RecoverUserSubmitSourceDirectory	UInt64	1	Used only with **NiProvider**. Specifies whether EMA recovers user-submitted source directories when recovering from a disconnect: • **1** (true) • **0** (false)
RefreshFirstRequired	UInt64	1	Specifies whether EMA requires the application to send a refresh message prior to sending update messages. Available values include: • **1** (true) • **0** (false)
RemoveItemsOnDisconnect	UInt64	1	Used only with **NiProvider**. Specifies whether EMA removes items from its internal hash table whenever it disconnects from the ADH: • **1** (true) • **0** (false)
RequestTimeout	UInt64	15000	Specifies the length of time (in milliseconds) the `OmmProvider` waits for a response to a request before sending another request (the DICTIONARY domain will not send another request). If set to **0**, EMA waits for a response indefinitely.
ServiceCountHint	UInt64	513	Sets the size of directory structures for managing services. If the application specifies **0**, EMA resets it to **513**.
XmlTraceFileName	EmaString	EmaTrace	Sets the name of the file to which to write XML trace output if tracing is selected.
XmlTraceHex	UInt64	0	Sets whether to print incoming and outgoing messages in hexadecimal format. Possible values are: • **0** (false): Do not print messages in hexadecimal format. • **1** (true): Print messages in hexadecimal format.
XmlTraceMaxFileSize	UInt64	100000000	Specifies the maximum size (in bytes) for the trace file.

Table 6: *Provider*Group Parameters (Continued)

PARAMETER	TYPE	DEFAULT	DESCRIPTION
XmlTracePing	UInt64	0	Sets the EMA to trace incoming and outgoing ping messages. Possible values are: • **0** (false): Do not trace ping messages. • **1** (true): Trace ping messages.
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
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.
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**.
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.
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.

Table 6: *Provider*Group Parameters (Continued)

3.3 Channel Group

ChannelGroup is used only with an **NiProvider**.

The **ChannelGroup** contains a **ChannelList**, which contains one or more **Channel** entries (each uniquely identified by a **<Name …/>** entry). Each channel includes a set of connection parameters for a specific connection or connection type.

There is no default channel. If an EMA application needs a specific channel, you must specify this in the appropriate **Consumer** or **NiProvider** section.

- For details on the parameters you can use to configure the **Consumer** component, refer to Section 3.1.4.
- For details on the parameters you can use to configure the **NiProvider** component, refer to Section 3.2.4
- For a generic **ChannelGroup** XML schema, refer to Section 3.3.1.
- For a list of universal parameters you can use in configuring any type of **Channel** regardless of the channel type, refer to Section 3.3.2.
- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_SOCKET**, refer to Section 3.3.3.
- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_ENCRYPTED**, refer to Section 3.3.4.
- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_HTTP**, refer to Section 3.3.4.
- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_RELIABLE_MCAST**, refer to Section 3.3.5.

3.3.1 Generic XML Schema for ChannelGroup

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

```
<ChannelGroup>
    <ChannelList>
        <Channel>
            <Name value="VALUE"/>
            …
        </Channel>
    </ChannelList>
</ChannelGroup>
```

3.3.2 Universal Channel Entry Parameters

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

PARAMETER NAME	TYPE	DEFAULT	NOTES
ChannelType	Enumeration	RSSL_SOCKET	Specifies the type of channel or connection used to connect to the server. Calling the host function can change this field. For details on this event, refer to Section 4.4.2. Use enumeration values with EMA's programmatic configuration (for further details, refer to Section 4.5). Available values include: • **RSSL_SOCKET** (0) • **RSSL_ENCRYPTED** (1): Supported on Windows OS and Linux. • **RSSL_HTTP** (2): Supported only on Windows OS • **RSSL_RELIABLE_MCAST** (4)
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.
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*.
HighWaterMark	UInt64	6144	Specifies the upper buffer-usage threshold for the channel.
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.
Name	EmaString		Specifies the **Channel**'s name.
NumInputBuffers	UInt64	10	Specifies the number of buffers used to read data. Buffers are sized according to **maxFragmentSize**. For details on RsslChannel and **maxFragmentSize**, refer to the *Transport API Developers Guide*.
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.

Table 7: Universal <Channel> Parameters

3.3.3 Parameters for Use with Channel Type: RSSL_SOCKET

In addition to the universal parameters listed in Section 3.3.2, you can use the following parameters to configure a channel whose type is **RSSL_SOCKET**.

PARAMETER NAME	TYPE	DEFAULT	NOTES
CompressionThreshold	UInt64	30	Sets the message size threshold (in bytes, the allowed value is 30-UInt32 MAX), above which all messages are compressed (thus individual messages might not be compressed). Different compression types have different behaviors and compression efficiency can vary depending on message size.
CompressionType	Enumeration	None	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 further details, refer to Section 4.5). Available values include: • **None** (0) • **ZLib** (1) • **LZ4** (2) **Note:** A server can be configured to force a particular compression type, regardless of client settings.
Host	EmaString	localhost	Specifies the host name of the server to which the 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 8: Parameters for Channel Type: RSSL_SOCKET

3.3.4 Parameters for Use with Channel Types: RSSL_HTTP or RSSL_ENCRYPTED

In addition to the universal parameters listed in Section 3.3.2, you can use the following parameters to configure a channel whose type is 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 further details, refer to Section 4.5). Available values include: • **None** (0) • **ZLib** (1) • **LZ4** (2) **Note:** A server can be configured to force a particular compression type, regardless of client settings.
Host	EmaString	localhost	Specifies the host name of the server to which the 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 and HTTPS connection messages.
Port	EmaString	14002	Specifies the port on the remote server to which the EMA connects.
ProxyHost	EmaString	""	Specifies the host name of the proxy to which the EMA connects. The parameter value can be a host name or an IP address. Any value provided by a function call overrides the setting in configuration file.
ProxyPort	EmaString	""	Specifies the port on the proxy to which the EMA connects. Any value provided by a function call overrides the setting in configuration file.
TcpNodelay	UInt64	1	Specifies whether to use Nagle's algorithm when sending data. Available values are: • **0**: Send data using Nagle's algorithm. • **1**: Send data without delay.

Table 9: Parameters for Channel Types: RSSL_HTTP or RSSL_ENCRYPTED

3.3.5 Parameters for Use with Channel Type: RSSL_RELIABLE_MCAST

In addition to the universal parameters listed in Section 3.3.2, you can use the following parameters to configure a channel whose type is **RSSL_RELIABLE_MCAST**.

Several of these parameters configure how the channel sends a Host Status Messages on the network, while others configure how the channel manages RRCP packet transmission. For further details on the Host Status Message (HSM) concept, on configuring HSMs, and on RRCP packet transmission, refer to the *ADS* or *AHD Software Installation 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
DisconnectOnGap	UInt64	0	Specifies whether the underlying connection should be closed if a multicast gap situation is detected. • **0** (false): **0** is the default value which means the underlying connection is not closed if a multicast gap situation occurs. • **1** (true): Sets the underlying connection to close if a multicast gap situation occurs.
HsmInterface	EmaString	“”	Specifies the Host Status Message (HSM) interface. By default, **HsmInterface** is set to the host machine’s default interface.
HsmInterval	UInt64		The interval (in seconds) over which HSM packets are sent. You can use **rrdump** to change the value of **hsmInterval**. Thus, after starting the application, you can stop and restart HSM publication as needed. The default interval is **0** (disabled) which suspends host status message publication.
HsmMultAddress	EmaString	“”	Specifies the multicast address over which this channel sends HSM packets. EMA configuration allows for the use of defined aliases.
HsmPort	EmaString	“”	Specifies the multicast port to which this channel sends HSM packets.
ndata	UInt64	7	Specifies the maximum number of retransmissions to attempt for an unacknowledged point-to-point packet.
nmissing	UInt64	128	Specifies the maximum number of missed consecutive multicast packets, from a particular node, from which RRCP requests retransmits.
nrreq	UInt64	3	Specifies the maximum number of retransmit requests that can be sent for a missing packet.
PacketTTL	UInt64	5	Sets the lifespan (in hops) of the data packet through the multicast network, which can prevent the packet from circulating indefinitely. It has a range of **0** - **255**. • **0** means the message can be sent only to other applications on the same machine. • A value of **255** sets the message to travel through the network indefinitely.

Table 10: Parameters for Channel Type: RSSL_RELIABLE_MCAST

PARAMETER NAME	TYPE	DEFAULT	NOTES
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**).
pktPoolLimitLow	UInt64	180000	Specifies the low-water mark for the RRCP packet pool. If RRCP packet allocation gets frozen (due to **pktPoolLimitHigh** having been reached), additional RRCP packets are allocated only when usage falls below the **pktPoolLimitLow** setting. **pktPoolLimitLow** should be greater than **3 * userQLimit**.
RecvAddress	EmaString	“”	Specifies the multicast address to which this channel connects for receiving data.
RecvPort	EmaString	“”	Specifies the multicast port to which this channel connects for receiving data.
SendAddress	EmaString	“”	Specifies the multicast address to which this channel connects for sending data.
SendPort	EmaString	“”	Specifies the multicast port to which this channel connects for sending data.
tbchold	UInt64	3	Specifies the maximum time that RRCP holds a transmitted broadcast packet in case the packet needs to be retransmitted. **tbchold** is specified in RRCP clock ticks (100 milliseconds), so a value of 2 means 200 milliseconds.
tcpControlPort	EmaString	“”	Specifies the port to use for the RRCP **tcpControlPort**. This port is used when troubleshooting RRCP using the **rrdump** tool. A setting of **-1** disables **tcpControlPort**.
tdata	UInt64	1	Specifies the time that RRCP waits before retransmitting an unacknowledged point-to-point data message. **tdata** is specified in RRCP clock ticks of 100 milliseconds, thus a value of 2 means 200 milliseconds.
tpphold	UInt64	3	Specifies the maximum time that RRCP holds a transmitted point-to-point packet in case the packet needs to be retransmitted. **tpphold** is specified in RRCP clock ticks (100 milliseconds), so a value of 2 means 200 milliseconds.
trreq	UInt64	4	Specifies the amount of time that RRCP waits before "resending" a retransmit request for a missed multicast packet. **trreq** is specified in RRCP clock ticks (100 milliseconds), so a value of 2 means 200 milliseconds.
twait	UInt64	3	Specifies the duration of time for which RRCP ignores additional retransmit requests for a data packet that it has already retransmitted. This time period starts with the receipt of the first request for retransmission. **twait** is specified in RRCP clock ticks (100 milliseconds), so a value of 2 means 200 milliseconds.

Table 10: Parameters for Channel Type: RSSL_RELIABLE_MCAST (Continued)

PARAMETER NAME	TYPE	DEFAULT	NOTES
UnicastPort	EmaString	“”	Port to which this connection connects for unicast messages (i.e., ack/nak messages and any retransmit messages). This value also configures a TCP listening port for use with the **rrdump** tool.
userQLimit	UInt64	65535	Specifies the maximum backlog of messages allowed on an application's inbound message queue. If **userQLimit** is exceeded, the RRCP protocol engine begins to discard messages for that application until the backlog decreases.

Table 10: Parameters for Channel Type: RSSL_RELIABLE_MCAST (Continued)

3.3.6 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>
```

3.3.7 Example Programmatic Configuration for ChannelSet

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

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

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

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

3.4 Server Group

ServerGroup is used only with an **IProvider**.

The **ServerGroup** contains a **ServerList**, which contains one or more **Server** entries (each uniquely identified by a **<Name …/>** entry). Each channel includes a set of connection parameters for a specific connection or connection type.

There is no default server. If an EMA application needs a specific server, you need to specify this in the appropriate **Consumer** or **IProvider** section.

- For details on the parameters you can use to configure the **Consumer** component, refer to Section 3.1.4.
- For details on the parameters you can use to configure the **IProvider** component, refer to Section 3.2.4.
- For a generic **ServerGroup** XML schema, refer to Section 3.4.1.
- For a list of parameters you can use in configuring **Server**, refer to Section 3.4.2.

3.4.1 Generic XML Schema for ServerGroup

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

```
<ServerGroup>
    <ServerList>
        <Server>
            <Name value="VALUE"/>
            …
        </Server>
    </ServerList>
</ServerGroup>
```

3.4.2 Server Entry Parameters

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

PARAMETER NAME	TYPE	DEFAULT	NOTES
ConnectionMinPingTimeout	UInt64	20000	Configures the minimum length of time (in milliseconds) to use as a timeout for a connected channel.
ConnectionPingTimeout	UInt64	60000	Specifies the duration (in milliseconds) after which the EMA terminates the connection if it does not receive communication or pings from the server.
CompressionThreshold	UInt64	30	Sets the message size threshold (in bytes, the allowed value is 30-UInt32 MAX), above which all messages are compressed (thus individual messages might not be compressed). Different compression types have different behaviors and compression efficiency can vary depending on message size.

Table 11: Universal <Server> Parameters

PARAMETER NAME	TYPE	DEFAULT	NOTES
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 further details, refer to Section 4.5). Available values include: • **None** (0) • **ZLib** (1) • **LZ4** (2) **Note:** A server can be configured to force a particular compression type, regardless of client settings.
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*.
HighWaterMark	UInt64	6144	Specifies the upper buffer-usage threshold for the channel.
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.
Name	EmaString		Specifies the **Server**'s name.
NumInputBuffers	UInt64	10	Specifies the number of buffers used to read data. Buffers are sized according to **maxFragmentSize**. For details on RsslChannel and **maxFragmentSize**, refer to the *Transport API Developers Guide*.
Port	EmaString	14002	Specifies the port on the remote server to which the EMA connects.
ServerType	Enumeration	RSSL_SOCKET	Specifies the type of channel or connection used to connect to the server. Calling the host function can change this field. For details on this event, refer to Section 4.4.2. Use enumeration values with EMA's programmatic configuration (for further details, refer to Section 4.5). Currently **RSSL_SOCKET (0)** is the only available value.
SysRecvBufSize	UInt64	0	Specifies the size (in KB) of the system's receive buffer for this channel.

Table 11: Universal <Server> Parameters (Continued)

PARAMETER NAME	TYPE	DEFAULT	NOTES
SysSendBufSize	UInt64	0	Specifies the size (in KB) of the system's send buffer for this channel.
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 11: Universal <Server> Parameters (Continued)

3.5 Logger Group

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

3.5.1 Generic XML Schema for LoggerGroup

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

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

3.5.2 Logger Entry Parameters

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

PARAMETER NAME	TYPE	DEFAULT	NOTES
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.
Name	EmaString		Sets a unique name for the Logger component in the **LoggerList**.
LoggerSeverity	Enumeration	Success	Severity levels aggregate messages so that a severity level includes all messages from higher levels (e.g., a setting of **1** includes any messages normally printed at levels **2** and **3**). Use enumeration values with EMA's programmatic configuration (for details, refer to in Section 4.5). Possible values are: • LoggerSeverity::Verbose (0) • LoggerSeverity::Success (1) • LoggerSeverity::Warning (2) • LoggerSeverity::Error (3) • LoggerSeverity::NoLogMsg (4)
LoggerType	Enumeration	File	Specifies the logging mechanism. Use enumeration values with EMA's programmatic configuration (for details, refer to in Section 4.5). Possible values are: • **LoggerType::File**: EMA logs to the file specified in the parameter **FileName**. • **LoggerType::Stdout**: EMA logs to stdout.

Table 12: Logger Group Parameters

3.6 Dictionary Group

The **DictionaryGroup** contains a **DictionaryList**, which contains one or more **Dictionary** components (each uniquely identified by a **<Name …/>** entry). Each **Dictionary** component defines parameters relating to how the dictionary is accessed.

3.6.1 Generic XML Schema for DictionaryGroup

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

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

3.6.2 Dictionary Entry Parameters

Use the following parameters when configuring a **Dictionary** entry in the EMA.

PARAMETER NAME	TYPE	DEFAULT	NOTES
DictionaryType	Enumeration	ChannelDictionary	Specifies the dictionary loading mode. Use enumeration values with EMA's programmatic configuration (for further details, refer to Section 4.5). 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.
EnumTypeDefFileName	EmaString		Sets the location of the **EnumTypeDef** file.
EnumTypeDefItemName	EmaString	RWFEnum	Sets the name of the EnumTypeDef item specified in the source directory **InfoFilter.DictionariesProvided**, and **InfoFilter.DictionariesUsed** elements.
Name	EmaString		Sets a unique name for a Dictionary component in the **DictionaryList**.
RdmFieldDictionaryFileName	EmaString		Sets the location of the **RdmFieldDictionary**.
RdmFieldDictionaryItemName	EmaString	RWFFld	Sets the name of the RdmFieldDictionary item specified in the source directory **InfoFilter.DictionariesProvided**, and **InfoFilter.DictionariesUsed** elements.

Table 13: Dictionary Group Parameters

3.7 Directory Group

The **DirectoryGroup** contains a **DirectoryList**, which contains one or more **Directory** components (each uniquely identified by a **<Name …/>** entry). Each **Directory** component defines a list of **Service** components (which in turn define parameters that relate to the Service **InfoFilter** and **StateFilter**).

3.7.1 Generic XML Schema for Directory Entry

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

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

3.7.2 Setting Default Directory

If you do not specify a **DefaultDirectory**, then the EMA uses the first **Directory** component in the **DirectoryGroup**. However, you can specify a default directory by including the following parameter on a unique line inside **DirectoryGroup** but outside **DirectoryList** (for an example, refer to Appendix A).

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

3.7.3 Configuring a Directory in a DirectoryGroup

To configure a **Directory** component, add the following parameters (as appropriate) to the target directory in the XML Schema, each on a separate line:

PARAMETER	TYPE	DEFAULT	DESCRIPTION
Name	EmaString	N/A	Specifies the name of this **Directory** component. Name is required when creating a **Directory** component. You can use any value for **Name**.
Service	Component Name	N/A	Specifies **InfoFilter** and **StateFilter** values for the given **Service**.
			Note: A **Directory** may contain several Service components.

Table 14: Directory Entry Parameters

3.7.4 Service Entry Parameters

The Service Entry resembles the RDM's Source Directory Domain payload. For further details, refer to the *EMA C++ Edition RDM Usage Guide*. The EMA supports only the RDM entries **InfoFilter** and **StateFilter**. Use the following parameters when configuring a Service in EMA:

PARAMETER	TYPE	DEFAULT	DESCRIPTION
Name	EmaString	N/A	Specifies the name of this **Service** component. You can use any value for **Name**.
InfoFilter	Component Name	N/A	Specifies **InfoFilter** values for the given **Service**. **InfoFilter** values set a filter on the types of information EMA sends out.
StateFilter	Component Name	N/A	Specifies **StateFilter** values for the given **Service**. EMA sends **StateFilter** values to describe the service's state.

Table 15: Service Entry Parameters

3.7.5 InfoFilter Entry Parameters

EMA uses the following **InfoFilter** parameters to set filters on the types of information EMA sends out overs its services (as specified in the **EmaConfig.xml**).

For an example of structuring sections (e.g., **InfoFilter**) and components (e.g., **Capabilities** or **DictionariesProvided**) in **EmaConfig.xml** refer to Appendix A.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ServiceId	UInt64	N/A	Specifies the **Service**'s unique identifier. Available values include 0 - 65535.
Vendor	EmaString	N/A	Specifies the name of the vendor that provides the service.
IsSource	UInt64	0	Specifies whether the source of data sent on this service is its original publisher: • **1**: The service's data is provided directly by an original publisher • **0**: The service's data is a consolidation of multiple sources into a single service.

Table 16: Source Directory Info Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
Capabilities	Component Name	N/A	A component that includes **CapabilitiesEntry** parameters, which define the message domain types that can be requested from the service. For details on the parameter used in this section, refer to Section 3.7.5.1.
ItemList	EmaString	N/A	Specifies the name of the **SymbolList** that includes all items provided by this service.
DictionariesProvided	Component Name	N/A	A component that includes **DictionariesProvidedEntry** parameters, which define the dictionaries that the provider makes available. When specifying a dictionary, use the **Dictionary**'s component name whose ***ItemName** entries are used in this Service's RDM **DictionariesProvided** entry. For details on the parameter used in this section, refer to Section 3.7.5.2.
AcceptingConsumerStatus	UInt64	1	Indicates whether a service can accept and process messages related to Source Mirroring. • 0: The provider does not accept consumer status • 1: The provider accept consumer status
DictionariesUsed	Component Name	N/A	A component that includes **DictionariesUsedEntry** parameters, which define the dictionaries that the provider uses. When specifying a dictionary, use the **Dictionary**'s component name whose ***ItemName** entries are used in this Service's RDM **DictionariesUsed** entry. For details on the parameter used in this section, refer to Section 3.7.5.3.
QoS	Component Name	Includes a single QoSEntry	A component that includes **QoSEntry** sections, with each **QoSEntry** section defining a QoS **Timeliness** and **Rate** supported by this Service. For details on the parameter used in this section, refer to Section 3.7.5.4.
SupportsQoSRange	UInt64	0	Indicates whether the provider supports a QoS range when requesting an item. • 0: The provider does not support a QoS Range. • 1: The provider supports a QoS Range. For further details on using QoS ranges, refer to the *RDM C++ Edition Usage Guide*.
SupportsOutOfBandSnapshots	UInt64	For non-interactive provider: 0	Indicates whether the provider supports Snapshot requests after the OpenLimit has been reached: • 0: The provider does not support snapshot requests. • 1: The providers supports snapshot requests. For details on **OpenLimit**, refer to the *RDM C++ Edition Usage Guide*.

Table 16: Source Directory Info Parameters (Continued)

3.7.5.1 Capabilities Entry Parameter

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

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

Table 17: CapabilitiesEntry Parameter

3.7.5.2 DictionariesProvided Entry Parameter

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

PARAMETER	TYPE	DEFAULT	DESCRIPTION
DictionariesProvidedEntry	EmaString	**RWFFld** for RdmFieldDictionaryItemName **RWFEnum** for enumTypeDefItemName	Specifies the name of a **Dictionary** component from the **DictionaryGroup** section whose **RdmFieldDictionaryItemName** and **enumTypeDefItemName** parameters are used in this **Service**'s RDM **DictionariesProvided** entry.

Table 18: DictionariesProvided Parameter

3.7.5.3 DictionariesUsed Entry Parameter

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

PARAMETER	TYPE	DEFAULT	DESCRIPTION
DictionariesUsedEntry	EmaString	RWFFld for RdmFieldDictionaryItemName RWFEnum for enumTypeDefItemName	Specifies the name of a **Dictionary** component from the **DictionaryGroup** section whose **RdmFieldDictionaryItemName** and **enumTypeDefItemName** are used in this **Service**'s RDM **DictionariesUsed** entry.

Table 19: DictionariesUsedEntry Parameter

3.7.5.4 QoSEntry Section and Associated Parameters

Use a **QoSEntry** section to configure a specific QoS supported by the **Service**'s **InfoFilter**. You can include multiple QoSEntry sections in a parent **QoS** section. For an example of how to structure QoS entries in the **EmaConfig.xml**, refer to Appendix A.

PARAMETER	TYPE	DEFAULT	DESCRIPTION
QoSEntry		N/A	**QoSEntry** is the name of a section that contains parameters specifying the **Timeliness** and **Rate** parameters for a given QoS. You can use multiple **QoSEntry** sections for a **Service**'s **InfoFilter**.

Table 20: QoSEntry Section and Associated Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
Timeliness	UInt64 or EmaString	Timeliness::Realtime	Specifies the QoS timeliness, which describes the age of the data (e.g., real time).
			Note: You can use numbers or names. Accepted names are listed in the **OmmQos.h** file.
Rate	UInt64 or EmaString	Rate::tickByTick	Specifies the QoS rate, which is the rate of change for data sent over the **Service**.
			Note: You can use numbers or names. Accepted names are listed in the **OmmQos.h** file.

Table 20: QoSEntry Section and Associated Parameters (Continued)

3.7.6 StateFilter Entry Parameters

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

PARAMETER	TYPE	DEFAULT	DESCRIPTION
ServiceState	UInt64	N/A	Specifies whether the service is up or down: • 0: Service is down • 1: Service is up
AcceptingRequests	UInt64	For non-interactive provider: 0	Specifies whether the service accepts request messages: • 0: The provider does not accept request messages. • 1: The provider accepts request messages.
Status		Open / Ok / None / ""	Specifies a change in status to apply to all items provided by this service. The status only applies to items that received an **OPEN/OK** in a refresh or status message.

Table 21: StateFilter Parameters

3.7.7 Status Entry Parameters

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

PARAMETER	TYPE	DEFAULT	DESCRIPTION
StreamState	EmaString	StreamState::Open	Specifies the state of the item stream.
			Note: Acceptable **StreamState** values are listed in the **OmmState.h** file.
DataState	EmaString	DataState::Ok	Specifies the state of the item data.
			Note: Acceptable **DataState** values are listed in the **OmmState.h** file.

Table 22: Service Entry Parameters

PARAMETER	TYPE	DEFAULT	DESCRIPTION
StatusCode	EmaString	StatusCode::None	Specifies the item status code. **Note:** Codes and their meanings are listed in the **OmmState.h** file.
StatusText	EmaString	""	Specific **StatusText** regarding the current data and stream state. Typically used for informational purposes. **StatusText** has an encoded text with a maximum allowed length of 32,767 bytes.t

Table 22: Service Entry Parameters (Continued)

Chapter 4 EMA Configuration Processing

4.1 Overview

The EMA 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 EMA's configuration as used by application components.

4.2 Default Configuration

4.2.1 Default *Consumer* Configuration

Each EMA 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 EMA'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.

EMA 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 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"

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 2.3.

4.2.2 Default *Provider* Configurations

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

EMA provides a hard-coded configuration for use whenever an **`Omm*Provider*Config`** 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.

4.2.2.1 Example: Default Channel Behavior (NiProvider)

The default (hard-coded) behavior for a **Channel** adheres to the following configuration:

- **ChannelType** value="RSSL_SOCKET"
- **CompressionType** value="None"
- **TcpNoDelay** value="1"
- **Host** value="localhost"
- **Port** value= "14003"

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

4.2.2.2 Example: Default Server Behavior (IProvider)

The default (hard-coded) behavior for a **Server** adheres to the following configuration:

- **ServerType** value="RSSL_SOCKET"
- **CompressionType** value="None"
- **TcpNoDelay** value="1"
- **Port** value= "14002"

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

4.3 Processing EMA's XML Configuration File

The Elektron SDK package installs into EMA's working directory a default configuration file named **EmaConfig.xml**. By default, EMA 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 EMA determines its configuration, refer to Section 4.3.1.

Except for the parameters **DefaultConsumer** and **DefaultNiProvider**, you must wrap all other elements defined in the EMA's configuration file in a component definition (i.e., **Consumer**, **NiProvider**, **Logger**, **Channel**, **Directory**, or **Dictionary**) otherwise EMA ignores the element. This section includes some examples that illustrate this requirement. Appendix A illustrates the proper placement of **DefaultConsumer** and **DefaultNiProvider** in the configuration file.

4.3.1 Reading the Configuration File

Note: The following section uses EMA Consumer objects (i.e., `OmmConsumer` and `OmmConsumerConfig`) to illustrate how EMA checks for a configuration file, and if one exists, how EMA starts to process it. For details on interactive and non-interactive providers (instead of consumers) and their OmmProvider-type objects, refer to the *EMA C++ Developers Guide*.

The `OmmConsumer` constructor expects an `OmmConsumerConfig` object. By default, `OmmConsumerConfig` searches its working directory for a configuration file by the name of **EmaConfig.xml**. However, if you store your configuration file elsewhere on the system, or use a custom filename, you can include an argument with the configuration object to specify the alternate path and/or name of your configuration file.

4.3.1.1 Using EmaConfig.xml in the Working Directory

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

- If **EmaConfig.xml** exists and contains valid XML, EMA uses the XML to modify its configuration.
- If **EmaConfig.xml** exists, but is empty or contains malformed XML, the application uses the default configuration (for details on the default configuration, refer to Section 4.2).
- If **EmaConfig.xml** does not exist, the application uses the default configuration (for details on the default configuration, refer to Section 4.2).

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

```
OmmConsumerConfig config();

OmmConsumer consumer(config);
```

For complete details, you can refer to the example *100__MarketPrice__Streaming* included with the Elektron 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, EMA appends **EmaConfig.xml** to the argument and verifies whether **EmaConfig.xml** exists in the specified directory.
- If the argument represents a directory and filename, EMA verifies whether the specified file exists.
- If the specified file does not exist, the application throws an IceException, which indicates the specified path and the current working directory.
- If the argument represents neither a file nor a directory, an IceException is thrown.

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

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

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

For example, 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 *EMA C++ Developers Guide*) and pass it to the **OmmConsumer** object as follows (where ***PATH*** is the alternate path and/or filename you want to use for your configuration file):

```
OmmConsumerConfig config(PATH);

OmmConsumer consumer(config);
```

For complete details, you can refer to the example *111__MarketPrice__UserSpecifiedFileConfig* included with the Elektron SDK.

4.3.2 Use of the Correct Order in the XML Schema

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

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

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

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

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

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

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

- Configuration 1:

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

- Configuration 2:

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

4.3.3 Processing the Consumer “Name”

The EMA 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 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 configuration file has no **Consumer** components, the EMA application uses **EmaConsumer**.

4.3.4 Processing the *Provider* "Name"

The EMA is hard-coded to use a default non-interactive provider of **Ema*Provider***. However, you can change this by using the configuration file (e.g., **EmaConfig.xml**). When you use the XML file, the default ***Provider* Name** is either specified by the **Default*Provider*** element, or if this parameter is not set, then the EMA application will default to the name of the first non-interactive provider component.

- If **Default*Provider*** uses an invalid name (i.e., no ***Provider*** components in the XML file use that name), the EMA throws an exception indicating that **Default*Provider*** is invalid.
- If the **EmaConfig.xml** has no ***Provider*** components, the EMA application uses **Ema*Provider***.

4.4 Configuring EMA Using Function Calls

From an application standpoint, instantiating `OmmConsumerConfig` and `OmmNiProviderConfig` objects creates the initial configuration from the **DefaultXML.h** and an EMA XML configuration file (if one exists). Certain variables can then be altered via function calls on the `OmmConsumerConfig` and `Omm*Provider*Config` objects.

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

4.4.1 EMA Configuration Function Calls

4.4.1.1 OmmConsumerConfig Class Function Calls

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

FUNCTION	DESCRIPTION
addAdminMsg(const ReqMsg&)	Populates part of or all of the login request message, directory request message, or dictionary request message according to the specification discussed in the *EMA Reuters Domain Models (RDM) Usage Guide* specific to the programming language you use.
applicationId(const EmaString &)	Sets the **applicationId** variable. **applicationId** has no default value.
clear()	Clears existing content from the `OmmConsumerConfig` object.
config(const Data&)	Passes in the consumer's programmatic configuration.
consumerName(const EmaString &)	Sets the consumer name, which is used to select a specific consumer as defined in EMA's configuration. If a consumer does not exist with that name, the application throws an exception.
host(const EmaString &)	Sets the host and port parameters. For details, refer to Section 4.4.2.
operationModel(OperationModel)	Sets the operation model to either `OmmConsumerConfig::ApiDispatchEnum` (which is the default) or `OmmConsumerConfig::UserDispatchEnum`.
password(const EmaString &)	Sets the **password** variable. **password** has no default value.
position(const EmaString &)	Sets the **position** variable. **position** has no default value.
username(const EmaString &)	Sets the **username** variable. If **username** is not set, the application extracts a username from the run-time environment.

Table 23: `OmmConsumerConfig` Class Function Calls

4.4.1.2 Class Function Calls

You can use the following function calls in an EMA ***Provider*** application. For further details on variables, refer to the *EMA C++ RDM Usage Guide*. Certain function calls can only be used with a specific provider type (e.g., **`addAdminMsg( const ReqMsg& )`** can only be used with an **NiProvider**). The parameter's description will mention any provider-type restrictions.

FUNCTION	DESCRIPTION
addAdminMsg(const ReqMsg&)	Used only with **NiProvider**. Populates part of or all of the login request message according to the specification discussed in the *EMA C++ RDM Usage Guide*.
addAdminMsg(const RefreshMsg&)	Populates part of or all of the initial directory refresh message according to the specification discussed in the *EMA C++ RDM Usage Guide*.
adminControlDirectory(AdminControl)	Specifies whether the API or the user controls the sending of Directory refresh messages. Available values include: • **`OmmProviderConfig::ApiControlEnum`** (which is the default) • **`OmmProviderConfig::UserControlEnum`** For details on control models, refer to **Omm*Provider*Config.h**.
applicationId(const EmaString &)	Used only with **NiProvider**. Sets the ***applicationId*** variable. ***applicationId*** has no default value.
clear()	Clears existing content from the **`OmmProviderConfig`** object.
config(const Data&)	Passes in the provider's programmatic configuration.
host(const EmaString &)	Used only with **NiProvider**. Sets the **host** and **port** parameters. For details, refer to Section 4.4.2.
instanceId(const EmaString&)	Used only with **NiProvider**. Sets the ***instanceId*** variable. ***instanceId*** has no default value.
operationModel(OperationModel)	Specifies whether the API or the user controls the thead (i.e., the operation model). Available values include: • **`OmmProviderConfig::ApiDispatchEnum`**(which is the default) • **`OmmProviderConfig::UserDispatchEnum`** For details on operation models, refer to **Omm*Provider*Config.h**.
password(const EmaString &)	Used only with **NiProvider**. Sets the ***password*** variable. ***password*** has no default value.
port()	Sets the **port** parameters.

Table 24: `OmmProviderConfig` Class Function Calls

FUNCTION	DESCRIPTION
position(const EmaString &)	Used only with **NiProvider**. Sets the ***position*** variable. ***position*** has no default value.
providerName(const EmaString &)	Sets the provider's name, which is used to select a specific provider as defined in EMA's configuration. If a provider does not exist with that name, the application throws an exception.
username(const EmaString &)	Used only with **NiProvider**. Sets the ***username*** variable. If ***username*** is not set, the application extracts a username from the run-time environment.

Table 24: `Omm`*`Provider`*`Config` Class Function Calls (Continued)

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

Host and **Port** parameters both have global default values. Thus, if either an `OmmConsumerConfig` or `OmmNiProviderConfig` object exists, its **Host** and **Port** will always have values (either the default value or some other value as specified in a configuration XML file such as **EmaConfig.xml**).

- The default ***Host*:*Port*** value for `OmmConsumerConfig` is **localhost:14002**.
- The default ***Host*:*Port*** value for `OmmNiProviderConfig` is **localhost:14003**.

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

Note: Calling the `host()` function sets **channelType** (refer to Section 3.3.2) to **RSSL_SOCKET**, regardless of how it was previously configured.

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

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

4.5 Programmatic Configuration

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

4.5.1 OMM Data Structure

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

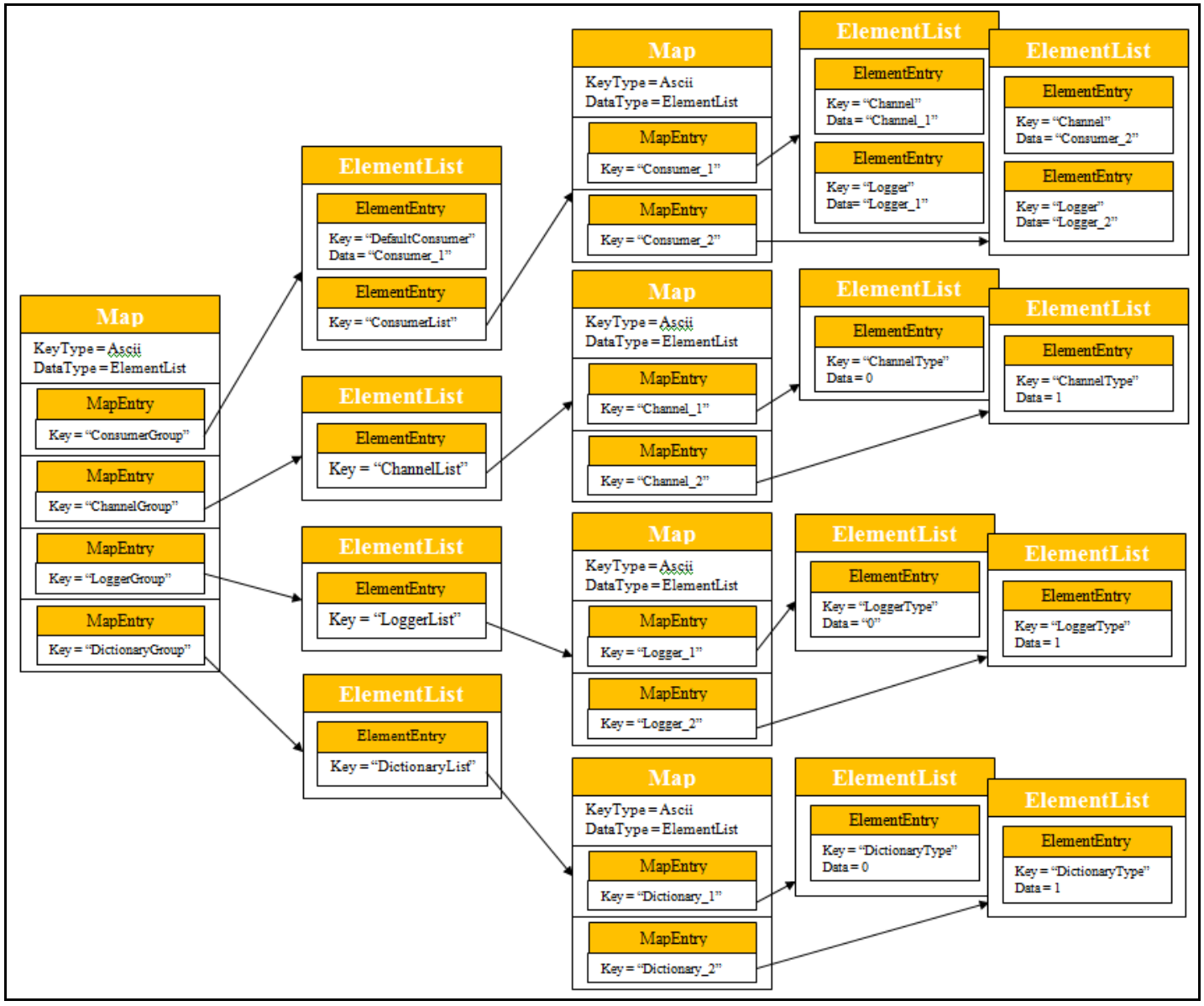
- The 1st tier lists EMA's Consumer, IProvider, NiProvider, Channel, Logger, Directory, and Dictionary components; each of which has it owns list in the 2nd tier.
- The 2nd tier includes each component's list and the default consumers and providers for use when loading configuration parameters.
- The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

4.5.2 Creating a Programmatic Configuration for a Consumer

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

To programmatically configure an EMA consumer:

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` or `OmmProvider`.

4.5.3 Example: Programmatic Configuration of a Consumer

The following sample illustrates the EMA programmatic configuration of a consumer:

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

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

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

elementList.addMap( "ConsumerList", innerMap );

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

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

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

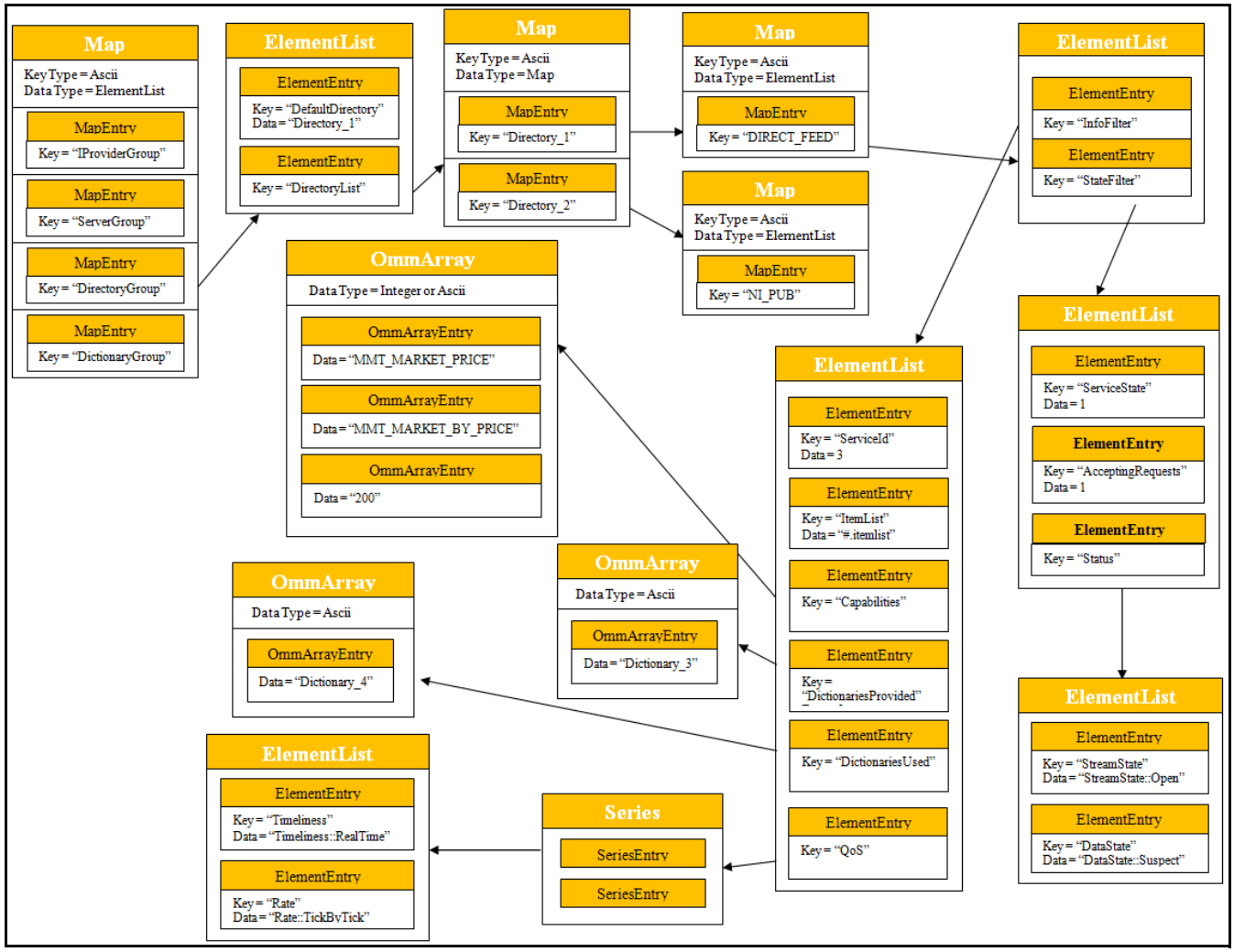
```
    .addEnum("CompressionType", 1)
    .addUInt( "GuaranteedOutputBuffers", 5000 )
    .addUInt( "ConnectionPingTimeout", 50000 )
    .addAscii( "Host", "localhost" )
    .addAscii("Port", "14002" )
    .addUInt( "TcpNodelay", 0 ).complete() ).complete();

elementList.addMap( "ChannelList", innerMap );

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

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

elementList.addMap( "LoggerList", innerMap );

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

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

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

elementList.addMap( "DictionaryList", innerMap );

elementList.complete();

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

configMap.complete();
```

4.5.4 Creating a Programmatic Configuration for a *Provider*

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

To programmatically configure an EMA *Provider*:

1. To configure an EMA directory's configuration parameters, create a map with the following hierarchy:



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

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

Note: You must set **adminControlDirectory** and **adminControlDictionary** to their default settings (`ApiControlEnum`) when programmatically configuring:

- A Directory Refresh message published by a *Provider*, or
- A Dictionary Refresh message published by an IProvider

4.5.5 Example: Programmatic Configuration of a Provider

The following sample illustrates the EMA programmatic configuration of a provider:

```
Map outermostMap, innerMap;
ElementList elementList;

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

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

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

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

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

```
        .addUInt("TcpNodelay", 1)
        .complete())
        .complete();

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

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

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

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

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

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

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

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

Map serviceMap;

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

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

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

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

...

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

Document ID: EMAC321CG.180
Date of issue: 18 August 2018