

Enterprise Transport API

C Edition

3.8.2.L1

VALUE ADDED COMPONENTS

Document Version: 3.8.2.L1
Date of issue: September 2024
Document ID: ETAC382UMVAC.240



LSEG DATA &
ANALYTICS

© **LSEG 2015 - 2024**. All rights reserved.

Republication or redistribution of LSEG Data & Analytics content, including by framing or similar means, is prohibited without the prior written consent of LSEG Data & Analytics. 'LSEG Data & Analytics' and the LSEG Data & Analytics logo are registered trademarks and trademarks of LSEG Data & Analytics.

Any software, including but not limited to: the code, screen, structure, sequence, and organization thereof, and its documentation are protected by national copyright laws and international treaty provisions. This manual is subject to U.S. and other national export regulations.

LSEG Data & Analytics, by publishing this document, does not guarantee that any information contained herein is and will remain accurate or that use of the information will ensure correct and faultless operation of the relevant service or equipment. LSEG Data & Analytics, its agents, and its employees, shall not be held liable to or through any user for any loss or damage whatsoever resulting from reliance on the information contained herein.

Contents

1	Introduction	1
1.1	About this Manual	1
1.2	Audience	1
1.3	Programming Language.....	1
1.4	Acronyms and Abbreviations	1
1.5	References	3
1.6	Documentation Feedback	3
1.7	Document Conventions.....	3
1.7.1	<i>Optional, Conditional, and Required</i>	3
1.7.2	<i>Typographic</i>	3
1.7.3	<i>Document Structure</i>	4
1.7.4	<i>Diagrams</i>	5
2	Product Description and Overview	6
2.1	What is the Enterprise Transport API?	6
2.2	What are Enterprise Transport API Value Added Components?	6
2.3	Enterprise Transport API Reactor	7
2.4	Open Message Model Consumer Watchlist.....	8
2.4.1	<i>Data Stream Aggregation and Recovery</i>	8
2.4.2	<i>Additional Features</i>	8
2.4.3	<i>Usage Notes</i>	8
2.5	Administration Domain Model Representations	8
2.6	Value Added Utilities	9
2.7	Value Added Cache	9
3	Building an OMM Consumer	10
3.1	Overview	10
3.2	Leverage Existing or Create New RsslReactor	10
3.3	Implement Callbacks and Populate Role	10
3.4	Establish Connection using rsslReactorConnect	11
3.5	Issue Requests and/or Post Information	11
3.6	Log Out and Shut Down.....	11
3.7	Additional Consumer Details.....	12
4	Building an OMM Interactive Provider	13
4.1	Overview	13
4.2	Leverage Existing or Create New RsslReactor	13
4.3	Create an RsslServer.....	13
4.4	Implement Callbacks and Populate Role	14
4.5	Associate Incoming Connections Using rsslReactorAccept.....	14
4.6	Perform Login Process.....	14
4.7	Provide Source Directory Information	15
4.8	Provide or Download Necessary Dictionaries	16
4.9	Handle Requests and Post Messages	16
4.10	Dispatch Round Trip Time Messages	17
4.11	Disconnect Consumers and Shut Down	17
4.12	Additional Interactive Provider Details	17
5	Building an OMM Non-Interactive Provider	18
5.1	Overview	18

5.2	Leverage Existing or Create New RsslReactor	18
5.3	Implement Callbacks and Populate Role	18
5.4	Establish Connection using rsslReactorConnect	19
5.5	Perform Dictionary Download	19
5.6	Provide Content	19
5.7	Log Out and Shut Down	20
5.8	Additional Non-Interactive Provider Details	20
6	Reactor Detailed View	21
6.1	Concepts	21
6.1.1	<i>Functionality: Enterprise Transport API Versus Enterprise Transport API Reactor</i>	22
6.1.2	<i>Reactor Error Handling</i>	23
6.1.3	<i>Reactor Error Info Codes</i>	23
6.1.4	<i>Enterprise Transport API Reactor Application Lifecycle</i>	24
6.2	Reactor Use	25
6.2.1	<i>Creating a Reactor</i>	25
6.2.2	<i>Destroying a Reactor</i>	28
6.3	Reactor Channel Use	29
6.3.1	<i>Reactor Channel Roles</i>	30
6.3.2	<i>Reactor Channel Role: OMM Consumer</i>	31
6.3.3	<i>Reactor Channel Role: OMM Provider</i>	34
6.3.4	<i>Reactor Channel Role: OMM Non-Interactive Provider</i>	35
6.3.5	<i>Reactor Channel: Role Union</i>	36
6.4	Managing Reactor Channels	37
6.4.1	<i>Adding Reactor Channels</i>	37
6.4.2	<i>Modifying Reactor Channels</i>	42
6.4.3	<i>Reactor Channel Reconnection and Recovery Behaviors</i>	43
6.4.4	<i>Removing Reactor Channels</i>	50
6.5	OMM Consumer Multi-Credential Configuration	50
6.5.1	<i>Login Message Configuration and Callback</i>	50
6.6	Reporting on Channel Statistics	51
6.7	Dispatching Data	51
6.7.1	<i>rsslReactorDispatch Function</i>	51
6.7.2	<i>Reactor Callback Functions</i>	53
6.7.3	<i>Reactor Callback: Channel Event</i>	54
6.7.4	<i>Reactor Callback: Default Message</i>	57
6.7.5	<i>Reactor Callback: RDM Login Message</i>	58
6.7.6	<i>Reactor Callback: RDM Directory Message</i>	60
6.7.7	<i>Reactor Callback: RDM Dictionary Message</i>	61
6.8	Writing Data	63
6.8.1	<i>Writing Data using rsslReactorSubmitMsg()</i>	63
6.8.2	<i>Writing data using rsslReactorSubmit()</i>	66
6.9	Creating and Using Tunnel Streams	73
6.9.1	<i>Authenticating a Tunnel Stream</i>	74
6.9.2	<i>Opening a Tunnel Stream</i>	74
6.9.3	<i>Negotiating Stream Behaviors: Class of Service</i>	76
6.9.4	<i>Tunnel Stream Callback Functions and Event Types</i>	79
6.9.5	<i>Opening a Tunnel Stream Code Sample</i>	82
6.9.6	<i>Accepting Tunnel Streams</i>	83
6.9.7	<i>Receiving Content on a TunnelStream</i>	88
6.9.8	<i>Sending Content on a TunnelStream</i>	88
6.10	Cloud Connectivity	92
6.10.1	<i>rsslReactorQueryServiceDiscovery</i>	92
6.10.2	<i>OAuth Credential Management</i>	95
6.11	JSON to RWF Protocol Conversion for WebSocket Support	99

6.11.1	<i>Consumer and WebSocket Support</i>	99
6.11.2	<i>Interactive Provider and WebSocket Support</i>	101
6.11.3	<i>RsslReactorJsonConverterOptions Structure</i>	103
6.11.4	<i>RsslReactorJsonConversionEventCallback Function</i>	104
6.11.5	<i>RsslReactorServiceNameToldCallback Function</i>	104
6.11.6	<i>RsslReactorServiceNameToldEvent Structure</i>	105
6.11.7	<i>RsslReactorJsonConversionEvent Structure</i>	105
6.12	Reactor Utility Functions	106
6.12.1	<i>General Reactor Utility Functions</i>	106
6.12.2	<i>RsslReactorChannelInfo Structure Members</i>	106
6.12.3	<i>rsslReactorIoctl Option Values</i>	106
6.12.4	<i>Reactor Debug Functions</i>	106
7	Consuming Data from the Cloud	108
7.1	Overview	108
7.2	Encrypted Connections	108
7.3	Credential Management	108
7.4	Version 1 Authentication Using OAuth Password and Refresh_Token	109
7.4.1	<i>Client_ID (AppKey)</i>	109
7.4.2	<i>Obtaining Initial Access and Refresh Tokens</i>	109
7.4.3	<i>Refreshing the Access Token and Sending a Login Reissue</i>	110
7.4.4	<i>Managing the Password and Client Secret</i>	111
7.4.5	<i>Session Management per User Credential</i>	111
7.5	Version 2 Authentication Using OAuth Client Credentials	111
7.5.1	<i>Configuring and Managing Version 2 Credentials</i>	111
7.5.2	<i>Version 2 OAuth Client Credentials Token Lifespan</i>	112
7.6	Service Discovery	112
7.7	Consuming Market Data	113
7.8	HTTP Error Handling for Reactor Token Reissues	114
7.9	Cloud Connection Use Cases	114
7.9.1	<i>Session Management Using Reactor Watchlist</i>	115
7.9.2	<i>Session Management Using Reactor with Watchlist Disabled</i>	115
7.9.3	<i>Explicit Service Discovery Use Case</i>	116
7.10	Logging of Authentication and Service Discovery Interaction	116
7.10.1	<i>Logged Request Information</i>	116
7.10.2	<i>Logged Response Information</i>	116
8	Administration Domain Models Detailed View	117
8.1	Concepts	117
8.2	RDM Message Base	118
8.2.1	<i>RSSL RDM Message Base Structure Members</i>	118
8.2.2	<i>RSSL RDM Message Types</i>	118
8.2.3	<i>RSSL RDM Encoding and Decoding Functions</i>	119
8.3	RDM Login Domain	119
8.3.1	<i>RSSL RDM Login Request</i>	120
8.3.2	<i>RSSL RDM Login Refresh</i>	124
8.3.3	<i>RSSL RDM Login Status</i>	130
8.3.4	<i>RSSL RDM Login Close</i>	132
8.3.5	<i>RSSL RDM Consumer Connection Status</i>	132
8.3.6	<i>Login Round Trip Time Message Use</i>	135
8.3.7	<i>Login Post Message Use</i>	136
8.3.8	<i>Login Ack Message Use</i>	136
8.3.9	<i>RSSL RDM Login Message Union</i>	136
8.3.10	<i>RSSL RDM Login Encoding and Decoding</i>	137
8.4	RSSL RDM Source Directory Domain	142

8.4.1	RSSL RDM Directory Request	142
8.4.2	RSSL RDM Directory Refresh	143
8.4.3	RSSL RDM Directory Update	145
8.4.4	RSSL RDM Directory Status.....	146
8.4.5	RSSL RDM Directory Close.....	147
8.4.6	RSSL RDM Consumer Status	148
8.4.7	Source Directory RDM Service.....	149
8.4.8	Source Directory RDM Service Info.....	150
8.4.9	Source Directory RDM Service State	152
8.4.10	Source Directory RDM Service Group State	153
8.4.11	Source Directory RDM Service Load.....	154
8.4.12	Source Directory RDM Service Data	155
8.4.13	Source Directory RDM Service Link Information	156
8.4.14	Source Directory RDM Service Link	157
8.4.15	Source Directory RDM Sequenced Multicast Information	158
8.4.16	RSSL RDM Directory Message Union.....	159
8.4.17	Source Directory Encoding and Decoding.....	160
8.5	Dictionary Domain	167
8.5.1	RSSL RDM Dictionary Request.....	167
8.5.2	RSSL RDM Dictionary Refresh.....	168
8.5.3	RSSL RDM Dictionary Status	170
8.5.4	RSSL RDM Dictionary Close.....	171
8.5.5	RSSL RDM Dictionary Message Union	171
8.5.6	Dictionary Encoding and Decoding.....	172
8.6	RDM Queue Messages.....	177
8.6.1	Queue Data Message Persistence	177
8.6.2	Queue Request.....	177
8.6.3	Queue Refresh	178
8.6.4	Queue Status.....	178
8.6.5	Queue Close.....	178
8.6.6	Queue Data	179
8.6.7	QueueDataExpired	182
8.6.8	Queue Ack.....	183
9	Warm Standby Feature	184
9.1	Overview	184
9.2	Warm Standby Modes.....	184
9.3	Warm Standby Configuration and Feature Details.....	186
9.3.1	Configuration Example for a Starting Server and a Standby Server	188
10	Preferred Host Feature	189
10.1	Preferred Host Operation Triggers.....	189
10.1.1	Function Call.....	189
10.1.2	Timer Trigger	189
10.2	Preferred Host Reconnection Behavior Changes	189
10.3	Preferred Host Operation Steps.....	189
10.4	Preferred Host Private Stream and Tunnel Stream Handling	191
10.5	Preferred Host IOCTL Functionality	191
11	Value Added Utilities	192
11.1	Utility Overview	192
11.2	Memory Buffer.....	192
11.3	Using the Memory Buffer	193

12	Payload Cache Detailed View	194
12.1	Concepts	194
12.2	Payload Cache.....	195
12.2.1	<i>Managing the Payload Cache.....</i>	<i>195</i>
12.2.2	<i>Cache Error Handling</i>	<i>195</i>
12.2.3	<i>Payload Cache Instances</i>	<i>196</i>
12.2.4	<i>Managing RDM Field Dictionaries for Payload Cache.....</i>	<i>197</i>
12.2.5	<i>Payload Cache Utilities.....</i>	<i>199</i>
12.3	Payload Cache Entries.....	200
12.3.1	<i>Managing Payload Cache Entries</i>	<i>200</i>
12.3.2	<i>Applying Data</i>	<i>201</i>
12.3.3	<i>Retrieving Data</i>	<i>202</i>

List of Figures

Figure 1.	Network Diagram Notation	5
Figure 2.	UML Diagram Notation.....	5
Figure 3.	OMM APIs with Value Added Components	6
Figure 4.	Enterprise Transport API Value Added Components.....	7
Figure 5.	ETA Reactor Thread Model	21
Figure 6.	Enterprise Transport API Reactor Application Lifecycle	24
Figure 7.	Flow Chart for writing data using rsslReactorSubmit	66
Figure 8.	Tunnel Stream Illustration	73
Figure 9.	Obtaining an Authentication Token	109
Figure 10.	Login Reissue	110
Figure 11.	Service Discovery	113
Figure 12.	Login Based Warm Standby Order of Events in a Cutover from Active to Standby.....	185
Figure 13.	Service Based Warm Standby Order of Events in a Cutover from Active to Standby	186
Figure 14.	Consumer Application using Cache to Store Payload Data for Item Streams	194

List of Tables

Table 1:	Acronyms and Abbreviations	1
Table 2:	ETA Functionality and ETA Reactor Comparison	22
Table 3:	RsslErrorInfo Structure Members	23
Table 4:	Reactor Error Info Codes	23
Table 5:	RsslReactor Structure Members	25
Table 6:	RsslReactor Creation Function	25
Table 7:	RsslCreateReactorOptions Structure Members	26
Table 8:	RsslCreateReactorOptions Utility Function	27
Table 9:	Reactor Debugging Levels	28
Table 10:	RsslReactor Destruction Function	28
Table 11:	RsslReactorChannel Structure Members	29
Table 12:	RsslReactorChannelRoleBase Structure Members	31
Table 13:	RsslReactorChannelRoleBase.roleType Enumerated Values	31
Table 14:	RsslReactorOMMConsumerRole Structure Members	32
Table 15:	RsslReactorOMMConsumerRole.dictionaryDownloadMode Enumerated Values	33
Table 16:	OMM Consumer Role Watchlist Options	33
Table 17:	RsslReactorOMMConsumerRole Utility Function	34
Table 18:	RsslReactorOMMProviderRole Structure Members	34
Table 19:	RsslReactorOMMProviderRole Utility Function	35
Table 20:	RsslReactorOMMNIProviderRole Structure Members	35
Table 21:	RsslReactorOMMNIProviderRole Utility Function	35
Table 22:	RsslReactorChannelRole Union Members	36
Table 23:	RsslReactorChannelRole Utility Function	36
Table 24:	rsslReactorConnect Function	37
Table 25:	RsslReactorConnectOptions Structure Members	37
Table 26:	RsslReactorConnectInfo Structure Members	38
Table 27:	RsslReactorPreferredHostOptions Structure Members	39
Table 28:	RsslReactorConnectOptions Utility Function	40
Table 29:	RsslReactorAccept Function	41
Table 30:	RsslReactorAcceptOptions Structure Members	41
Table 31:	RsslReactorAcceptOptions Utility Function	41
Table 32:	RsslReactorChannelIoctlCodes Enumerations	42
Table 33:	RsslReactorPreferredHostInfo Structure Members	43
Table 34:	reconnectAttemptLimit Configuration Behaviors	44
Table 35:	reconnectAttemptLimit Configuration Warm Standby Behaviors	46
Table 36:	rsslReactorCloseChannel Function	50
Table 37:	RsslReactorLoginRequestMsgCredential Structure Members	51
Table 38:	rsslReactorDispatch Function	52
Table 39:	RsslReactorDispatchOptions Structure Members	52
Table 40:	RsslReactorDispatchOptions Utility Function	53
Table 41:	RsslReactorCallbackRet Callback Return Codes	53
Table 42:	RsslReactorChannelEvent Structure Members	54
Table 43:	RsslReactorChannelEventType Enumeration Values	54
Table 44:	RsslReactorChannelEvent Utility Functions	55
Table 45:	RsslMsgEvent Structure Members	57
Table 46:	RsslMsgEvent Utility Function	57
Table 47:	RsslRDMLLoginMsgEvent Structure Members	58
Table 48:	RsslRDMLLoginMsgEvent Utility Function	59
Table 49:	RsslRDMDirectoryMsgEvent Structure Members	60
Table 50:	RsslRDMDirectoryMsgEvent Utility Function	60
Table 51:	RsslRDMDictionaryMsgEvent Structure Members	61

Table 52:	RsslRDMDictionaryMsgEvent Utility Function	62
Table 53:	rsslReactorSubmitMsg Function	63
Table 54:	RsslReactorSubmitMsgOptions Structure Members	63
Table 55:	RsslReactorRequestMsgOptions Structure Members	64
Table 56:	RsslReactorSubmitMsgOptions Utility Function	64
Table 57:	rsslReactorSubmitMsg Return Codes.....	64
Table 58:	Reactor Buffer Management Functions	67
Table 59:	rsslReactorGetBuffer Return Values	68
Table 60:	rsslReactorSubmitMsg Function	68
Table 61:	RsslReactorSubmitOptions Structure Members	69
Table 62:	rsslReactorSubmitMsg Return Codes.....	69
Table 63:	RsslReactorSubmitOptions Utility Function	70
Table 64:	rsslReactorPackBuffer Function.....	71
Table 65:	rsslReactorPackBuffer Return Values.....	71
Table 66:	RsslTunnelStreamAuthInfo Structure Members	74
Table 67:	rsslReactorOpenTunnelStream Method.....	74
Table 68:	RsslTunnelStreamOpenOptions	75
Table 69:	RsslClassofService.common Structure Members	76
Table 70:	RsslClassofService.authentication Structure Members	77
Table 71:	RsslClassofService.flowControl Structure Members	77
Table 72:	RsslClassofService.dataIntegrity Structure Members	78
Table 73:	RsslClassofService.guarantee Structure Members.....	78
Table 74:	Tunnel Stream Callback Event Types.....	79
Table 75:	Tunnel Stream Callback Functions	80
Table 76:	Tunnel Stream Callback Event Types.....	81
Table 77:	RsslTunnelStreamRequestEvent Structure Members.....	84
Table 78:	rsslReactorAcceptTunnelStream Function	84
Table 79:	RsslReactorAcceptTunnelStreamOptions Options.....	85
Table 80:	rsslReactorRejectTunnelStream Function	85
Table 81:	RsslReactorRejectTunnelStreamOptions Options.....	85
Table 82:	Tunnel Stream Buffer Methods	88
Table 83:	Tunnel Stream Submit Method	88
Table 84:	RsslTunnelStreamSubmitOptions Structure Members.....	89
Table 85:	RsslTunnelStreamSubmitMsgOptions Structure Members	89
Table 86:	RsslTunnelStreamInfo Structure Members	89
Table 87:	rsslReactorCloseTunnelStream Method	90
Table 88:	RsslTunnelStreamCloseOptions Structure Members.....	91
Table 89:	rsslReactorQueryServiceDiscovery Method.....	92
Table 90:	RsslReactorServiceDiscoveryOptions Structure Members.....	92
Table 91:	RsslReactorDiscoveryTransportProtocol Enumerations.....	93
Table 92:	RsslReactorDiscoveryDataFormatProtocol Enumerations	93
Table 93:	RsslReactorServiceEndpointEvent Structure Members	94
Table 94:	RsslReactorServiceEndpointEvent Structure Members	94
Table 95:	RsslReactorOAuthCredential Structure Members	95
Table 96:	RsslReactorOAuthCredentialEvent Structure Members	96
Table 97:	RsslReactorOAuthCredentialRenewal Members.....	96
Table 98:	rsslReactorSubmitOAuthCredentialRenewal	97
Table 99:	rsslReactorSubmitOAuthCredentialRenewal Options.....	97
Table 100:	RsslReactorOAuthCredentialRenewalMode Enums	98
Table 101:	RsslReactorJsonConverterOptions Structure Members	103
Table 102:	RsslReactorServiceNameToldCallback Parameters	104
Table 103:	RsslReactorServiceNameToIdEvent Structure Members	105
Table 104:	RsslReactorJsonConversionEvent Structure Members	105
Table 105:	Reactor Utility Functions	106
Table 106:	RsslReactorChannelInfo Structure Members.....	106

Table 107:	Domains Representations in the Administration Domain Model Value Added Component.....	117
Table 108:	RsslRDMMsgBase Structure Members	118
Table 109:	RsslRDMMsg	118
Table 110:	RDM Encoding and Decoding Functions	119
Table 111:	RsslRDMLginRequest Structure Members	120
Table 112:	RsslRDMLginRequest Flags	122
Table 113:	RsslRDMLginRequest Utility Functions.....	123
Table 114:	RsslRDMLginRefresh Structure Members	124
Table 115:	RsslRDMLginRefresh Flags	127
Table 116:	RsslRDMLginRefresh Utility Functions.....	129
Table 117:	RsslRDMServerInfo Structure Members.....	129
Table 118:	RsslRDMServerInfo Flags	129
Table 119:	RsslRDMServerInfo Utility Functions.....	129
Table 120:	RsslRDMLginStatus Structure Members.....	130
Table 121:	RsslRDMLginStatus Flags	131
Table 122:	RsslRDMLginStatus Utility Functions.....	132
Table 123:	RsslRDMLginClose Structure Member.....	132
Table 124:	RsslRDMLginClose Utility Functions.....	132
Table 125:	RsslRDMLginConsumerConnectionStatus Structure Members	133
Table 126:	RsslRDMLginConsumerConnectionStatus Flags	133
Table 127:	RsslRDMLginWarmStandbyInfo Structure Members	133
Table 128:	RDMLginServerTypes Enumeration Values.....	133
Table 129:	RsslRDMLginConsumerConnectionStatus Utility Functions.....	134
Table 130:	Login Round Trip Time Members.....	135
Table 131:	Login Round Trip Time Flag Enumeration Values	135
Table 132:	Login Round Trip Time Utility Functions	135
Table 133:	RsslRDMLginMsg Union Members	136
Table 134:	RsslRDMLginMsg Utility Functions	136
Table 135:	RDM Login Encoding and Decoding Functions	137
Table 136:	RsslRDMDirectoryRequest Structure Members.....	142
Table 137:	RsslRDMDirectoryRequest Flags	143
Table 138:	RsslRDMDirectoryRequest Utility Functions.....	143
Table 139:	RsslRDMDirectoryRefresh Structure Members.....	143
Table 140:	RsslRDMDirectoryRefresh Flags	144
Table 141:	RsslRDMDirectoryRefresh Utility Functions.....	144
Table 142:	RsslRDMDirectoryUpdate Structure Members.....	145
Table 143:	RsslRDMDirectoryUpdate Flags.....	145
Table 144:	RsslRDMDirectoryUpdate Utility Functions	146
Table 145:	RsslRDMDirectoryStatus Structure Members.....	146
Table 146:	RsslRDMDirectoryStatus Flags.....	147
Table 147:	RsslRDMDirectoryStatus Utility Functions	147
Table 148:	RsslRDMDirectoryClose Structure Member.....	147
Table 149:	RsslRDMDirectoryClose Utility Functions	147
Table 150:	RsslRDMDirectoryConsumerStatus Structure Members	148
Table 151:	RsslRDMDirectoryConsumerStatusService Structure Members.....	148
Table 152:	RsslRDMDirectoryConsumerStatus Utility Functions.....	148
Table 153:	RsslRDMService Structure Members	149
Table 154:	RsslRDMService Flags	149
Table 155:	RsslRDMService Utility Function	150
Table 156:	RsslRDMServiceInfo Structure Members.....	150
Table 157:	RsslRDMServiceInfo Flags	151
Table 158:	RsslRDMServiceInfo Utility Functions.....	152
Table 159:	RsslRDMServiceState Structure Members	152
Table 160:	RsslRDMServiceState Flags	153
Table 161:	RsslRDMServiceState Utility Functions.....	153

Table 162:	RsslRDMServiceGroupState Structure Members	153
Table 163:	RsslRDMServiceGroupState Flags	154
Table 164:	RsslRDMServiceGroupState Utility Functions.....	154
Table 165:	RsslRDMServiceLoad Structure Members.....	154
Table 166:	RsslRDMServiceLoad Flags	155
Table 167:	RsslRDMServiceLoad Utility Functions.....	155
Table 168:	RsslRDMServiceData Structure Members.....	155
Table 169:	RsslRDMServiceData Flags	156
Table 170:	RsslRDMServiceData Utility Functions.....	156
Table 171:	RsslRDMServiceLinkInfo Structure Members.....	156
Table 172:	RsslRDMServiceLinkInfo Utility Functions	156
Table 173:	RsslRDMServiceLink Structure Members.....	157
Table 174:	RsslRDMServiceLink Flags	157
Table 175:	RsslRDMServiceLink Utility Functions.....	158
Table 176:	RsslRDMServiceSeqMcastInfo Structure Members	158
Table 177:	RSSL RDM Service Sequenced Multicast Info Enumeration Values.....	158
Table 178:	RSSL RDM Address/Port Information Structure Members	159
Table 179:	RsslRDMServiceSeqMcastInfo Utility Functions	159
Table 180:	RsslRDMDirectoryMsg Union Members.....	159
Table 181:	RsslRDMDirectoryMsg Utility Functions.....	159
Table 182:	RDM Directory Encoding and Decoding Functions	160
Table 183:	RsslRDMDictionaryRequest Structure Members	167
Table 184:	RsslRDMDictionaryRequest Flag	167
Table 185:	RsslRDMDictionaryRequest Utility Functions.....	168
Table 186:	RsslRDMDictionaryRefresh Structure Members	168
Table 187:	RsslRDMDictionaryRefreshFlags	169
Table 188:	RsslRDMDictionaryRefresh Utility Functions.....	170
Table 189:	RsslRDMDictionaryStatus Structure Members.....	170
Table 190:	RsslRDMDictionaryStatus Flags	170
Table 191:	RsslRDMDictionaryStatus Utility Functions.....	171
Table 192:	RsslRDMDictionaryClose Structure Members.....	171
Table 193:	RsslRDMDictionaryClose Utility Functions	171
Table 194:	RsslRDMDictionaryMsg Union Members	171
Table 195:	RsslRDMDictionaryMsg Utility Functions	172
Table 196:	RDM Dictionary Encoding and Decoding Functions	172
Table 197:	RsslRDMQueueRequest Members.....	177
Table 198:	RsslRDMQueueRefresh Members.....	178
Table 199:	RsslRDMQueueStatus Members	178
Table 200:	RsslRDMQueueClose Members	178
Table 201:	RsslRDMQueueData Members	179
Table 202:	Queue Data Flag.....	179
Table 203:	RsslRDMQueueTimeoutCodes	180
Table 204:	Queue Data Message Encoding Methods	180
Table 205:	RsslRDMQueueDataExpired Structure Members.....	182
Table 206:	RsslRDMQueueDataUndeliverableCodes	182
Table 207:	RsslRDMQueueAck	183
Table 208:	Memory Buffer Functions	192
Table 209:	Payload Cache Management Functions	195
Table 210:	RsslCacheError Structure Members	195
Table 211:	Function for Cache Error Handling	196
Table 212:	Functions for Managing Cache Instances.....	196
Table 213:	RsslPayloadCacheConfigOptions Structure Members.....	196
Table 214:	Functions for Setting Dictionary to Cache.....	197
Table 215:	Payload Cache Utility Functions	199
Table 216:	Payload Cache Entry Management Functions	200

Table 217: Functions for Applying and Retrieving Cache Entry Data 202

Table 218: Functions for Using the Payload Cursor 203

1 Introduction

1.1 About this Manual

This document is authored by Enterprise Transport API architects and programmers who encountered and resolved many of the issues the reader might face. Several of its authors have designed, developed, and maintained the Enterprise Transport API product and other LSEG products which leverage it. As such, this document is concise and addresses realistic scenarios and use cases.

This guide documents the functionality and capabilities of the Enterprise Transport API C Edition Value Added Components. In addition to connecting to itself, the Enterprise Transport API can also connect to and leverage many different LSEG and customer components. If you want the Enterprise Transport API to interact with other components, consult that specific component's documentation to determine the best way to configure for optimal interaction.

1.2 Audience

This manual provides information and examples that aid programmers using the Enterprise Transport API C Edition Value Added Components. The level of material covered assumes that the reader is a user or a member of the programming staff involved in the design, coding, and test phases for applications which will use the Enterprise Transport API or its Value Added Components. It is assumed that the reader is familiar with the data types, classes, operational characteristics, and user requirements of real-time data delivery networks, and has experience developing products using the C programming language in a networked environment. Although Transport API Value Added Components offer alternate entry points to Transport API functionality, it is recommended that users are familiar with general Enterprise Transport API usage and interfaces.

1.3 Programming Language

The Enterprise Transport API Components are written to the C, Java, and C# languages. This guide discusses concepts related to the C Edition. All code samples in this document, value added component source, and all example applications provided with the product are written accordingly.

1.4 Acronyms and Abbreviations

ACRONYM / TERM	MEANING
ADH	LSEG Real-Time Advanced Distribution Hub is the horizontally scalable service component within the LSEG Real-Time Distribution System providing high availability for publication and contribution messaging, subscription management with optional persistence, conflation and delay capabilities.
ADS	LSEG Real-Time Advanced Distribution Server is the horizontally scalable distribution component within the LSEG Real-Time Distribution System providing highly available services for tailored streaming and snapshot data, publication and contribution messaging with optional persistence, conflation and delay capabilities.
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
DMM	Domain Message Model

Table 1: Acronyms and Abbreviations

ACRONYM / TERM	MEANING
Enterprise Message API	The Enterprise Message API (EMA) is an ease of use, open source, Open Message Model API. EMA is designed to provide clients rapid development of applications, minimizing lines of code and providing a broad range of flexibility. It provides flexible configuration with default values to simplify use and deployment. EMA is written on top of the Enterprise Transport API (ETA) utilizing the Value Added Reactor and Watchlist features of ETA.
Enterprise Transport API (ETA)	Enterprise Transport API is a high performance, low latency, foundation of the LSEG Real-Time SDK. It consists of transport, buffer management, compression, fragmentation and packing over each transport and encoders and decoders that implement the Open Message Model. Applications written to this layer achieve the highest throughput, lowest latency, low memory utilization, and low CPU utilization using a binary Rssl Wire Format when publishing or consuming content to/from LSEG Real-Time Distribution Systems.
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol (Secure)
JWK	JSON Web Key. Defined by RFC 7517, a JWK is a JSON formatted public or private key.
JWKS	JSON Web Key Set, This is a set of JWK, placed in a JSON array.
JWT	JSON Web Token. Defined by RFC 7519, JWT allows users to create a signed claim token that can be used to validate a user.
OMM	Open Message Model
QoS	Quality of Service
RDM	Domain Model
DP	Delivery Platform: this platform is used for REST interactions. In the context of Real-Time APIs, an API gets authentication tokens and/or queries Service Discovery to get a list of Real-Time - Optimized endpoints using DP.
LSEG Real-Time Distribution System	LSEG Real-Time Distribution System is LSEG's financial market data distribution platform. It consists of the LSEG Real-Time Advanced Distribution Server and LSEG Real-Time Advanced Distribution Hub. Applications written to the LSEG Real-Time SDK can connect to this distribution system.
Reactor	The Reactor is a low-level, open-source, easy-to-use layer above the Enterprise Transport API. It offers heartbeat management, connection and item recovery, and many other features to help simplify application code for users.
RFA	Robust Foundation API
RMTES	A multi-lingual text encoding standard
RSSL	Source Sink Library
RTT	Round Trip Time, this definition is used for round trip latency monitoring feature.
RWF	Rssl Wire Format, an LSEG proprietary binary format for data representation.
SOA	Service Oriented Architecture
SSL	Sink Source Library
UML	Unified Modeling Language
UTF-8	8-bit Unicode Transformation Format

Table 1: Acronyms and Abbreviations

1.5 References

- Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*
- *API Concepts Guide*
- *RMTES Specification*
- Enterprise Transport API C Edition *Developers Guide*
- The [LSEG Developer Community](#)

1.6 Documentation Feedback

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

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

1.7 Document Conventions

- Typographic
- Document Structure
- Diagrams

1.7.1 Optional, Conditional, and Required

Throughout this manual, all parameters, options, functions, Structure_ObjectVARIABLEs, flags, etc., are considered optional unless explicitly marked as **Conditional** or **Required**. If marked as conditional, the item's description will describe the conditions surrounding its use.

1.7.2 Typographic

This document uses the following types of conventions:

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

```
/* decode contents into the filter list structure */
if ((retVal = rsslDecodeFilterList(&decIter, &filterList)) >= RSSL_RET_SUCCESS)
{
    /* create single filter entry and reuse while decoding each entry */
    RsslFilterEntry filterEntry = RSSL_INIT_FILTER_ENTRY;
```


1.7.3 Document Structure

- General Concepts
- Detailed Concepts
- Interface Definitions
- Example Code

1.7.4 Diagrams

Diagrams that depict the interaction between components on a network use the following notation:





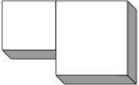





	Feed Handler, Real-Time server, or other application		Network of multiple servers
	Enterprise Transport API application		Point-to-point connection showing direction of primary data flow
	Application with local daemon		Point-to-point connection showing direction of client connecting to server
	Multicast network		Data from external source (e.g. consolidated network or exchange)
	Connection to Multicast network, no primary data flow direction		Connection to Multicast network showing direction of primary data flow

Figure 1. Network Diagram Notation





	Object
	Inheritance: object on left is like object on right
	Composition: object on left is made up of some number of objects on right
	Composition: object on left is made up of one object on right

Figure 2. UML Diagram Notation

2 Product Description and Overview

2.1 What is the Enterprise Transport API?

The Enterprise Transport API is a low-level Enterprise Transport API that provides the most flexible development environment to the application developer. It is the foundation on which all LSEG OMM-based components are built. The Enterprise Transport API allows applications to achieve the highest throughput and lowest latency available with any OMM API, but requires applications to perform all message encoding/decoding and manage all aspects of network connectivity. The Enterprise Enterprise Transport API, Enterprise Message API, and the Robust Foundation API make up the set of OMM API offerings.

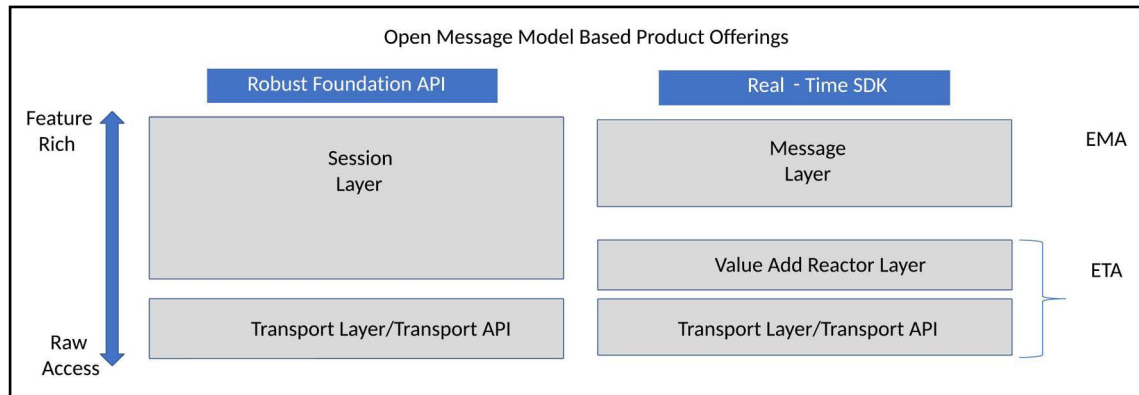


Figure 3. OMM APIs with Value Added Components

The Enterprise Transport API Value Added Components provide alternate entry points for applications to leverage OMM-based APIs with more ease and simplicity. These optional components help to offload much of the connection management code and perform encoding and decoding of some key OMM domain representations. Unlike older domain-based APIs that lock the user into capabilities or ease-of-use into the highest layer of API, Value Added components are independently implemented for use with the Enterprise Transport API and Robust Foundation API in their native languages (Example: Enterprise Transport API in C and Java, Robust Foundation API in C++ and Java). These implementations are then shipped with their respective API products as options for the application developer that may want these additional capabilities.

2.2 What are Enterprise Transport API Value Added Components?

The Value Added Components simplify and compliment the use of the Enterprise Transport API. These components (depicted in green in Figure 4) are offered along side the Enterprise Transport API to maximize the user experience and allow for more intuitive, straight forward, and rapid creation of Enterprise Transport API applications. Applications can write directly to the Enterprise Transport API interfaces or commingle some or all Value Added Components. The choice to leverage these components is up to the application developer; you do not need to use Value Added Components to use the Enterprise Transport API. Using Enterprise Transport API Value Added Components, you can choose and customize the balance between ultra high-performance raw access and ease-of-use feature functionality. Value Added Components are written to the Enterprise Transport API interfaces and are designed to work alongside the Enterprise Transport API. Their interfaces have a similar look and feel to Enterprise Transport API interfaces to provide simple migration and consistent use between all components and the Enterprise Transport API.

You can write applications directly to the transport layer (for details refer to the *Enterprise Transport API Developers Guide*) or to the Value Added layer. When writing to the value add layer, applications can make use of the feature-rich Watchlist (for details, refer to Section 2.4). Value Added layer and Watchlist features written to this are described in depth in this document. In general, the Enterprise Transport API packages together these layers as one API although most references to Enterprise Transport API are interchangeable with the transport layer.

All value added components provide fully supported library and header files ready to build into new or existing Enterprise Transport API applications. Examples and documentation are provided to show the full power and capability of the component for each layer of the Enterprise Transport API.

Some value added components provide buildable source code¹ to allow for customization and modification to suit specific user needs. This source code serves the following purposes:

- Clients may want to provide their own implementation of the component. Rather than starting from scratch, clients can modify the component to jump start their development efforts.

NOTE: If a client customizes a component's code, the client is responsible for its support and maintenance.

- Clients might want to build a new component that has similar behaviors to an existing component. Clients can leverage the code of one component to jump start their development efforts.
- Clients may want to collaborate in troubleshooting or suggesting improvements to the component for everyone's benefit.

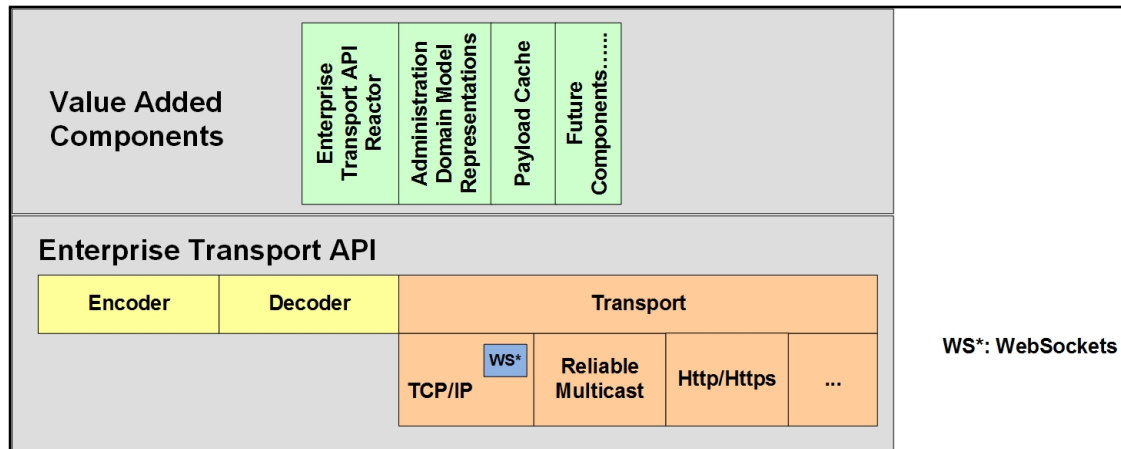


Figure 4. Enterprise Transport API Value Added Components

2.3 Enterprise Transport API Reactor

The *Enterprise Transport API reactor* is a connection management and event processing component that can significantly reduce the amount of code an application must write to leverage OMM in its own functions and to connect to other OMM-based devices. Consumer, interactive provider, and non-interactive provider applications can use the reactor and leverage it in managing consumer and non-interactive provider start-up processes, including user log in, source directory establishment, and dictionary download. The reactor also supports dispatching of events to user-implemented callback functions. In addition, it handles the flushing of user-written content and manages network pings on the user's behalf. The connection recovery feature allows the reactor to automatically recover from disconnects. Value Added domain representations are coupled with the reactor, allowing domain specific callbacks to be presented with their respective domain representation for easier, more logical access to content. For more information, refer to Chapter 6. This component depends on the Value Added Administration Domain Model Representation component, the Value Added Utilities, Enterprise Transport API Reliable Transport Package, Enterprise Transport API Message Package, and Enterprise Transport API Data Package.

The Preferred Host feature allows consumer applications to configure a specific host or warm standby group as a *preferred* connection, and upon either a timer-based (using either a static timer or a cron string) or function based trigger, cause the API to perform a single connection attempt to the preferred host without cutting any currently active connections. Once a connection has been established to the preferred host, the API will switch the connections and alert the user. If the connection attempt fails, the reactor will signal that the operation is complete, and will not make any changes to the current connection. The Preferred Host configuration for a reactor channel can be changed at any time through IOCTL calls.

To access all Enterprise Transport API reactor functionality, including the Administration Domain Model Representations, an application must include `rsslReactor.h`.

1. LSEG fully supports the use of its pre-built library and header files. Provided source code can help with user troubleshooting and debugging. However, the user, not LSEG, is responsible for supporting any modifications to the provided source.

2.4 Open Message Model Consumer Watchlist

The **RsslReactor** features a per-channel watchlist that provides a wealth of functionality for OMM consumer applications. The watchlist automatically performs various recovery behaviors for which developers would normally need to account.

The watchlist supports consuming from TCP-based connections (**RSSL_CONN_TYPE_WEBSOCKET**, **ConnectionTypes.WEBSOCKET**, etc.).

For details on configuring the **RsslReactor** to enable the consumer watchlist, refer to Section 6.3.2.

2.4.1 Data Stream Aggregation and Recovery

The watchlist automatically recovers data streams in response to failure conditions, such as disconnects and unavailable services, so that applications do not need special handling for these conditions. As conditions are resolved, the watchlist will re-request items on the application's behalf. Applications can also use this function to request data before a connection is fully established.

To recover from disconnects using a watchlist, enable the reactor's connection recovery. Options to reconnect disconnected channels are detailed in Section 6.4.1.2.

For efficient bandwidth usage, the watchlist also combines multiple requests for the same item into a single stream and forwards response messages to each requested stream as appropriate.

2.4.2 Additional Features

The watchlist provides additional features for convenience:

- **Group and Service Status Fanout:** The **RsslReactor** maintains a directory stream to receive service updates. As group status messages or service status messages are received, the **RsslReactor** forwards the status to all affected streams via **RsslStatusMsgs**.
- **Quality of Service Range Matching:** The **RsslReactor** will accept and aggregate item requests that specify a range of **RsslQos**, or requests that do not specify an **RsslQos**. After comparing these requests with the quality of service from the providing service, the watchlist uses the best matching quality of service.
- **Support for Enhanced Symbol List Behaviors:** The **RsslReactor** supports data streams when requesting a Symbol List item. For details on requesting Symbol list data streams, refer to the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.
- **Support for Batch Requests:** The **RsslReactor** will accept batch requests regardless of whether the connected provider supports them.

2.4.3 Usage Notes

Applications should note the following when enabling the watchlist:

- The application must use the **rsslReactorSubmitMsg** function to send messages. It cannot use **rsslReactorSubmit**.
- Only one login stream should be opened per **RsslReactorChannel**.
- To prevent unnecessary bandwidth use, the watchlist will not recover a dictionary request after a complete refresh is received.
- As private streams are intended for content delivery between two specific points, the watchlist does not aggregate nor recover them.
- The **RsslReactorOMMConsumerRole.dictionaryDownloadMode** option is not supported when the watchlist is enabled.

2.5 Administration Domain Model Representations

The **Administration Domain Model Representations** are RDM-specific representations of the OMM administrative domain models. This Value Added Component contains structures that represent the messages within the Login, Source Directory, and Dictionary domains. All

structures follow the formatting and naming specified in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*, so access to content is logical and specific to the content being represented. This component also handles all encoding and decoding functionality for these domain models, so the application needs only to manipulate the message's structure members to send or receive this content. This not only significantly reduces the amount of code an application needs to interact with OMM devices (i.e., LSEG Real-Time Distribution System infrastructure), but also ensures that encoding/decoding for these domain models follow OMM-specified formatting rules. Applications can use this Value Added Component directly to help with encoding, decoding, and representation of these domain models. When using the Enterprise Transport API Reactor, this component is embedded to manage and present callbacks with a domain-specific representation of content. For more information, refer to Chapter 8. This component depends on the Value Added Utilities, Enterprise Transport API Message Package, and Enterprise Transport API C Data Package.

To access all data package functionality, an application must include `rssIRDMMsg.h`.

2.6 Value Added Utilities

The Value Added Utilities are a collection of helper constructs, mainly used by the Enterprise Transport API Reactor. Included is a multi-purpose memory buffer type that can help with flexible, reusable memory - this is leveraged by the Administration Domain Model Representations when encoding or decoding messages. Other Value Added Utilities include a simple queue, mutex locks, thread helper functionality, and a simple event alerting component.

2.7 Value Added Cache

Applications can leverage the OMM payload cache feature. Using the payload cache, an application can maintain a local store of the OMM container data it consumes, publishes, or transforms. The cache maintains the latest values of the OMM data entries: container values update to reflect the most recent refresh and update message payloads whenever the application receives them. The Enterprise Transport API retrieves data from the cache entry in the form of an encoded OMM container. The payload cache is independent of other Value Added components, and only requires the Enterprise Transport API Message Package and Enterprise Transport API Data Package. Only library and API header files are available for the cache component.

3 Building an OMM Consumer

3.1 Overview

This chapter provides an overview of how to create an OMM consumer application using the ETA Reactor and Administration Domain Model Representation Value Added Components. The Value Added Components simplify the work done by an OMM consumer application when establishing a connection to other OMM interactive provider applications, including LSEG Real-Time Distribution System, Data Feed Direct, and Real-Time — Optimized. After the Reactor indicates that the connection is ready, an OMM consumer can then consume (i.e., send data requests and receive responses) and publish data (i.e., post data).

The general process can be summarized by the following steps.

- Leverage existing or create new **RsslReactor**
- Implement callbacks and populate role
- Establish connection using **rsslReactorConnect**
- Issue requests and/or post information
- Log out and shut down

The **rsslVACConsumer** example application, included with the ETA product, provides one implementation of an OMM consumer application that uses the ETA Value Added Components. The application is written with simplicity in mind and demonstrates usage of the ETA and ETA Value Added Components. Portions of functionality have been abstracted and can easily be reused, though you might need to modify it to achieve your own unique performance and functionality goals.

3.2 Leverage Existing or Create New RsslReactor

The **RsslReactor** can manage one or multiple **RsslReactorChannel** structures. This functionality allows the application to associate OMM consumer connections with an existing **RsslReactor**, having it manage more than one connection, or to create a new **RsslReactor** to use with the connection.

To create a new **RsslReactor**, the application must use the static method **rsslCreateReactor** of the **RsslReactor** class. This will create any necessary memory and threads that the **RsslReactor** uses to manage **RsslReactorChannels** and their content flow. If the application is using an existing **RsslReactor**, there is nothing additional to do.

For more information about the **RsslReactor** and its creation, refer to Section 6.2.1.

3.3 Implement Callbacks and Populate Role

Before creating the OMM consumer connection, the application needs to specify callback functions to use for all inbound content. The callback functions are specified on a per **RsslReactorChannel** basis so each channel can have its own unique callback functions or existing callback functions can be specified and shared across multiple **RsslReactorChannels**.

Use of an **RsslReactor** requires the use of several callback functions. The application must have the following:

- **RsslReactorChannelEventCallback**, which returns information about the **RsslReactorChannel** and its state (e.g., connection up)
- **RsslDefaultMsgCallback**, which processes all data not handled by other optional callbacks.

In addition to the required callbacks, an OMM consumer can specify several administrative domain-specific callback functions. Available domain-specific callbacks include:

- **RsslRDMLLoginMsgCallback**, which processes all data for the RDM Login domain.
- **RsslRDMDirectoryMsgCallback**, which processes all data for the RDM Source Directory domain.

- **RsslRDMDictionaryMsgCallback**, which processes all data for the RDM Dictionary domain.

The **RsslReactorOMMConsumerRole** structure should be populated with all callback information for the **RsslReactorChannel**.

The **RsslReactorOMMConsumerRole** allows the application to provide login, directory, and dictionary request information. This can be initialized with default information. The callback functions are specified on the **RsslReactorOMMConsumerRole** class or with specific information according to the application and user. The **RsslReactor** will use this information when starting up the **RsslReactorChannel**.

For more information about the **RsslReactorOMMConsumerRole**, refer to Section 6.3.1. For information about the various callback functions and their specifications, refer to Section 6.7.2.

3.4 Establish Connection using **rsslReactorConnect**

After populating the **RsslReactorOMMConsumerRole**, the application can use **rsslReactorConnect** to create a new outbound connection. **rsslReactorConnect** will create an OMM consumer-type connection using the provided configuration and role information.

After establishing the underlying connection, a channel event is returned to the application's **RsslReactorChannelEventCallback**; this provides the **RsslReactorChannel** and the state of the current connection. At this point, the application can begin using the **rsslReactorDispatch** function to dispatch directly on this **RsslReactorChannel**, or continue using **rsslReactorDispatch** to dispatch across all channels associated with the **RsslReactor**.

The **RsslReactor** will use the login, directory, and dictionary information specified on the **RsslReactorOMMConsumerRole** to perform all channel initialization for the user. After a user has logged in, received a source directory response, and downloaded field dictionaries, a channel event is returned to inform the application that the connection is ready.

The **rsslReactorConnect** function is described in Section 6.4.1.1. Dispatching is described in Section 6.7.

3.5 Issue Requests and/or Post Information

After the **RsslReactorChannel** is established, the channel can be used to request additional content. When issuing the request, the consuming application can use the **serviceId** of the desired service, along with the stream's identifying information. Requests can be sent for any domain using the formats defined in that domain model specification. Domains provided by LSEG are defined in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*. This content will be returned to the application via the **RsslDefaultMsgCallback**.

At this point, an OMM consumer application can also post information or forward generic messages to capable provider applications. All content requested, received, or posted is encoded and decoded using the ETA Message Package and the ETA Data Package described in the Enterprise Transport API C Edition *Developers Guide*.

3.6 Log Out and Shut Down

When the consumer application is done retrieving, forwarding, or posting content, the consumer can close the **RsslReactorChannel** by calling **rsslReactorCloseChannel**. This will close all item streams and log out the user. Prior to closing the **RsslReactorChannel**, the application should release any unwritten pool buffers to ensure proper memory cleanup.

If the application is done with the **RsslReactor**, the **rsslDestroyReactor** function can be used to shutdown and clean up any **RsslReactor** resources.

- Closing an **RsslReactorChannel** is described in Section 6.4.4.
- Shutting down an **RsslReactor** is described in Section 6.2.2.

3.7 Additional Consumer Details

The following locations provide specific details about using OMM consumers, the ETA, and ETA Value Added Components:

- The **rssIVAConsumer** application demonstrates one way of implementing of an OMM consumer application that uses ETA Value Added Components. The application's source code and **ReadMe** file contain additional information about specific implementation and behaviors.
- Chapter 6 provides a detailed look at the ETA Reactor.
- Chapter 8 provides more information about the Administration Domain Model Representations.
- The Enterprise Transport API C Edition *Developers Guide* provides specific ETA encoder/decoder and transport usage information.
- The Enterprise Transport API C Edition *LSEG Domain Model Usage Guide* provides specific information about the Domain Message Models used by this application type.

4 Building an OMM Interactive Provider

4.1 Overview

This chapter provides a high-level description of how to create an OMM interactive provider application using the ETA Reactor and Administration Domain Model Representation Value Added Components. An OMM interactive provider application opens a listening socket on a well-known port allowing OMM consumer applications to connect. The ETA Value Added Components simplify the work done by an OMM interactive provider application when accepting connections and handling requests from OMM consumers.

The following steps summarize this process:

- Leverage an existing **RsslReactor**, or create a new one
- Create an **RsslServer**
- Implement callbacks and populate role
- Associate incoming connections using **rsslReactorAccept**
- Perform login process
- Provide source directory information
- Provide necessary dictionaries
- Handle requests and post messages
- Dispatch Round Trip Time messages
- Disconnect consumers and shut down

Included with the ETA product, the **rsslIVAPProvider** example application provides one way of implementing an OMM interactive provider application that uses the ETA Value Added Components. The application is written with simplicity in mind and demonstrates the use of the ETA and ETA Value Added Components. Portions of the functionality are abstracted for easy reuse, though you might need to customize it to achieve your own unique performance and functionality goals.

4.2 Leverage Existing or Create New RsslReactor

The **RsslReactor** can manage one or multiple **RsslReactorChannel** structures. This allows the application to choose to associate OMM provider connections with an existing **RsslReactor**, have it manage more than one connection, or create a new **RsslReactor** to use with the connection.

If the application is creating a new **RsslReactor**, the **RsslCreateReactor** function is used. This will create any necessary memory and threads that the **RsslReactor** uses to manage **RsslReactorChannels** and their content flow. If the application is using an existing **RsslReactor**, there is nothing additional to do.

Detailed information about the **RsslReactor** and its creation are available in Section 6.2.1.

4.3 Create an RsslServer

The first step of any ETA Interactive Provider application is to establish a listening socket, usually on a well-known port so that consumer applications can easily connect. The provider uses the **rsslBind** function to open the port and listen for incoming connection attempts. This uses the standard ETA Transport functionality described in the Enterprise Transport API C Edition *Developers Guide*.

Whenever an OMM consumer application attempts to connect, the provider will use the **RsslServer** and associate the incoming connections with an **RsslReactor**, which will accept the connection and perform any initialization as described in Section 4.4 and Section 4.5.

4.4 Implement Callbacks and Populate Role

Before accepting an incoming connection with an OMM provider, the application needs to specify callback functions to use for all inbound content. Callback functions are specified on a per **RsslReactorChannel** basis so each channel can have its own unique callback functions or existing callback functions can be specified and shared across multiple **RsslReactorChannels**.

The following callback functions are required for use with an **RsslReactor**:

- **RsslReactorChannelEventCallback**, which returns information about the **RsslReactorChannel** and its state (e.g., connection up)
- **RsslDefaultMsgCallback**, which processes all data not handled by other optional callbacks.

In addition to the required callbacks, an OMM provider can specify several administrative domain-specific callback functions. Available domain-specific callbacks are:

- **RsslRDMLLoginMsgCallback**, which processes all data for the RDM Login domain.
- **RsslRDMDirectoryMsgCallback**, which processes all data for the RDM Source Directory domain.
- **RsslRDMDictionaryMsgCallback**, which processes all data for the RDM Dictionary domain.

The **RsslReactorOMMProviderRole** structure should be populated with all callback information for the **RsslReactorChannel**.

Detailed information about the **RsslReactorOMMProviderRole** is in Section 6.3.1. Information about the various callback functions and their specifications are available in Section 6.8.2.

4.5 Associate Incoming Connections Using **rsslReactorAccept**

After the **RsslReactorOMMProviderRole** is populated, the application can use **rsslReactorAccept** to accept a new inbound connection. **rsslReactorAccept** will accept an OMM provider connection from the passed-in **RsslServer** using provided configuration and role information.

When the underlying connection is established, a channel event is returned to the application's **RsslReactorChannelEventCallback**; this will provide the **RsslReactorChannel** and indicate the current connection state. At this point, the application can begin using the **rsslReactorDispatch** function to dispatch directly on this **RsslReactorChannel**, or continue using **rsslReactorDispatch** to dispatch across all channels associated with the **RsslReactor**.

The **RsslReactor** will perform all channel initialization and pass any administrative domain information to the application via the callbacks specified with the **RsslReactorOMMProviderRole**.

- For more details on the **rsslReactorAccept** function, refer to Section 6.4.1.7.
- For more details on dispatching, refer to Section 6.6.

4.6 Perform Login Process

Applications authenticate with one another using the Login domain model. An OMM interactive provider must handle consumer Login request messages and supply appropriate responses. Login information will be provided to the application via the **RsslRDMLLoginMsgCallback**, when specified on the **RsslReactorOMMProviderRole**.

After receiving a Login request, an interactive provider can perform any necessary authentication and permissioning.

- If the interactive provider grants access, it should send an **RsslRDMLLoginRefresh** to convey that the user successfully connected. This message should indicate the feature set supported by the provider application.
- If the interactive provider denies access, it should send an **RsslRDMLLoginStatus**, closing the connection and informing the user of the reason for denial.

Login messages can be encoded and decoded using the **RsslRDMLLoginMsg**. More details and code examples are in Section 8.3.

All content requested, received, or posted is encoded and decoded using the ETA Message Package and the ETA Data Package described in the Enterprise Transport API C Edition *Developers Guide*.

Information about the Login domain and expected content formatting is available in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

4.7 Provide Source Directory Information

The Source Directory domain model conveys information about all available services in the system. An OMM consumer typically requests a Source Directory to retrieve information about available services and their capabilities. This includes information about supported domain types, the service's state, the quality of service, and any item group information associated with the service. LSEG recommends that at a minimum, an interactive provider supply the Info, State, and Group filters for the Source Directory.

- The Source Directory Info filter contains the name and **RsslRDMSERVICEID** for each available service. The interactive provider should populate the filter with information specific to the services it provides.
- The Source Directory State filter contains status information for the service informing the consumer whether the service is Up (available), or Down (unavailable).
- The Source Directory Group filter conveys item group status information, including information about group states, as well as the merging of groups. If a provider determines that a group of items is no longer available, it can convey this information by sending either individual item status messages (for each affected stream) or a Directory message containing the item group status information. Additional information about item groups is available in the Enterprise Transport API C Edition *Developers Guide*.

Source Directory messages can be encoded and decoded using the **RsslRDMDirectoryMsg**. More details and code examples are in Section 8.4.

All content requested, received, or posted is encoded and decoded using the ETA Message Package and the ETA Data Package described in the Enterprise Transport API C Edition *Developers Guide*.

Information about the Source Directory domain and expected content formatting is available in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

4.8 Provide or Download Necessary Dictionaries

Some data requires the use of a dictionary for encoding or decoding. The dictionary typically defines type and formatting information, and tells the application how to encode or decode information. Content that uses the **RsslFieldList** type requires the use of a field dictionary (usually the **RDMFieldDictionary**, though it can instead be a user-defined or modified field dictionary).

The Source Directory message should notify the consumer about dictionaries needed to decode content sent by the provider. If the consumer needs a dictionary to decode content, it is ideal that the interactive provider application also make this dictionary available to consumers for download. The provider can inform the consumer whether the dictionary is available via the Source Directory.

If consuming from an LSEG Real-Time Advanced Distribution Hub and providing content downstream, a provider application can also download the RWFFld and RWFEnum dictionaries. Using these dictionaries, the ETA can retrieve appropriate dictionary information for providing field list content. A provider can use this feature to ensure they are using the appropriate version of the dictionary or to encode data. An LSEG Real-Time Advanced Distribution Hub that supports provider dictionary downloads sends a Login request message containing the **SupportProviderDictionaryDownload** login element. The ETA sends the dictionary request using the Dictionary domain model.¹ For details on using the Login domain and expected message content, refer to the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

Dictionary messages can be encoded and decoded using the **RsslRDMDictionaryMsg**. More details and code examples are in Section 8.5. Dictionary requests will be provided via the **RsslRDMDictionaryMsgCallback**, when specified on the **RsslReactorOMMProviderRole**.

Whether loading a dictionary from file or requesting it from an LSEG Real-Time Advanced Distribution Hub, the ETA offers several utility functions for loading, downloading, and managing a properly-formatted field dictionary. The ETA also has utility functions that help the provider encode into an appropriate format for downloading or decoding downloaded dictionaries.

- All content requested, received, or posted is encoded and decoded using the ETA Message Package and the ETA Data Package described in the Enterprise Transport API C Edition *Developers Guide*.
- Information about the Dictionary domain, dictionary utility functions, and expected content formatting is available in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

4.9 Handle Requests and Post Messages

A provider can receive a request for any domain, though this should typically be limited to the domain capabilities indicated in the Source Directory. When a request is received, the provider application must determine if it can satisfy the request by:

- Comparing **RsslMsgKey** identification information
- Determining whether it can provide the requested quality of service
- Ensuring that the consumer does not already have a stream open for the requested information

If a provider can service a request, it should send appropriate responses. However, if the provider cannot satisfy the request, the provider should send an **RsslStatusMsg** to indicate the reason and close the stream. All requests and responses should follow specific formatting as defined in the domain model specification. The Enterprise Transport API C Edition *LSEG Domain Model Usage Guide* defines all domains provided by LSEG. This content will be returned to the application via the **RsslDefaultMsgCallback**.

The provider can specify that it supports post messages via the **RsslRDMLoginRefresh**. If a provider application receives a post message, the provider should determine the correct handling for the post. This depends on the application's role in the system and might involve storing the post in its cache or passing it farther up into the system. If the provider is the destination for the post, the provider should send any requested acknowledgments, following the guidelines described in the Enterprise Transport API C Edition *Developers Guide*. Any posted content will be returned to the application via the **RsslDefaultMsgCallback**.

All content requested, received, or posted is encoded and decoded using the ETA Message Package and the ETA Data Package as described in the Enterprise Transport API C Edition *Developers Guide*.

1. Because this is instantiated by the provider, the application should use a **streamId** with a negative value. Additional details are provided in subsequent chapters.

4.10 Dispatch Round Trip Time Messages

Optionally, a provider can send a Round Trip Time message to gather Round Trip Time statistics. While the ETA does not regulate rules for implementing the Round Trip Time message, the ETA provides several examples for applying this feature in provider applications. Generally, if the provider wants to support the Round Trip Time feature, the provider must provide methods for sending generic Round Trip Time messages to a consumer and extend callback methods for Round Trip Time calculation.

For detailed information, refer to the Enterprise Transport API C Edition *RDM Usage Guide*.

4.11 Disconnect Consumers and Shut Down

If the **RsslReactor** application must shut down, it can either leave consumer connections intact or shut them down. If the provider decides to close consumer connections, the provider should send an **RsslStatusMsg** on each connection's Login stream closing the stream. At this point, the consumer should assume that its other open streams are also closed.

It can then close the **RsslReactorChannels** by calling **rsslReactorCloseChannel**. Prior to closing the **RsslReactorChannel**, the application should release any unwritten pool buffers to ensure proper memory cleanup.

If the application is done with the **RsslReactor**, the **rsslDestroyReactor** function can be used to shutdown and cleanup any **RsslReactor** resources.

- Closing an **RsslReactorChannel** is described in Section 6.4.4.
- Shutting down an **RsslReactor** is described in Section 6.2.2.

4.12 Additional Interactive Provider Details

For specific details about OMM interactive providers, the ETA, and ETA Value Added Component use, refer to the following locations:

- The **rsslIVAPProvider** application demonstrates one implementation of an OMM interactive provider application that uses ETA Value Added Components. The application's source code and **ReadMe** file have additional information about specific implementation and behaviors.
- Chapter 6 provides a detailed look at the ETA Reactor.
- Chapter 8 provides more information about the Administration Domain Model Representations.
- The Enterprise Transport API C Edition *Developers Guide* provides specific ETA encoder/decoder and transport usage information.
- The Enterprise Transport API C Edition *LSEG Domain Model Usage Guide* provides specific information about the Domain Message Models used by this application type.

5 Building an OMM Non-Interactive Provider

5.1 Overview

This chapter provides an overview of how to create an OMM non-interactive provider application using the ETA Reactor and Administration Domain Model Representation Value Added Components. The Value Added Components simplify the work done by an OMM non-interactive provider application when establishing a connection to an LSEG Real-Time Advanced Distribution Hub. After the reactor indicates that the connection is ready, an OMM non-interactive provider can publish information into the LSEG Real-Time Advanced Distribution Hub cache without needing to handle requests for the information. The LSEG Real-Time Advanced Distribution Hub and other LSEG Real-Time Distribution System components can cache the information and provide it to any OMM consumer applications that indicate interest.

The general process can be summarized by the following steps.

- Leverage existing or create new **RsslReactor**
- Implement callbacks and populate role
- Establish connection using **rsslReactorConnect**
- Perform dictionary download
- Provide content
- Log out and shut down

The **rsslIVANIPProvider** example application, included with the ETA product, provides one implementation of an OMM non-interactive provider application that uses the ETA Value Added Components. The application is written with simplicity in mind and demonstrates usage of the ETA and ETA Value Added Components. Portions of functionality have been abstracted and can easily be reused, though you might need to modify it to achieve your own unique performance and functionality goals.

5.2 Leverage Existing or Create New RsslReactor

The **RsslReactor** can manage one or multiple **RsslReactorChannel** structures. This allows the application to choose to associate OMM non-interactive provider connections with an existing **RsslReactor**, having it manage more than one connection, or to create a new **RsslReactor** to use with the connection.

If the application is creating a new **RsslReactor**, the **rsslCreateReactor** function is used. This will create any necessary memory and threads that the **RsslReactor** uses to manage **RsslReactorChannel** and their content flow. If the application is using an existing **RsslReactor**, there is nothing more to do.

Detailed information about the **RsslReactor** and its creation are available in Section 6.2.1.

5.3 Implement Callbacks and Populate Role

Before creating the OMM non-interactive provider connection, the application needs to specify callback functions to use for all inbound content. Callback functions are specified on a per **RsslReactorChannel** basis so each channel can have its own unique callback functions or existing callback functions can be specified and shared across multiple **RsslReactorChannels**.

An **RsslReactor** requires the use of the following callback functions:

- **RsslReactorChannelEventCallback**, which returns information about the **RsslReactorChannel** and its state (e.g., connection up)
- **RsslDefaultMsgCallback**, which processes all data not handled by other optional callbacks.

Additionally, an OMM non-interactive provider can specify the administrative domain-specific callback function **RsslRDMLLoginMsgCallback**, which processes all data for the RDM Login domain.

The **RsslReactorOMMNIProviderRole** structure should be populated with all callback information for the **RsslReactorChannel**. **RsslReactorOMMNIProviderRole** allows the application to provide login request and initial directory refresh information. This can be initialized with default information. Callback functions are specified on the **RsslReactorOMMNIProviderRole** structure or with specific information according to the application and user. The **RsslReactor** will use this information when starting up the **RsslReactorChannel**.

- For detailed information on the **RsslReactorOMMNIProviderRole**, refer to Section 6.3.1.
- For information on the various callback functions and their specifications, refer to Section 6.7.2.

5.4 Establish Connection using **rsslReactorConnect**

After populating the **RsslReactorOMMNIProviderRole**, the application can use **rsslReactorConnect** to create a new outbound connection. **rsslReactorConnect** will create an OMM non-interactive provider type connection using the provided configuration and role information.

When the underlying connection is established, a channel event will be returned to the application's **RsslReactorChannelEventCallback**, which provides the **RsslReactorChannel** and indicates the current connection state. At this point, the application can begin using the **rsslReactorDispatch** function to dispatch directly on this **RsslReactorChannel**, or continue using **rsslReactorDispatch** to dispatch across all channels associated with the **RsslReactor**.

The **RsslReactor** will use the login and directory information specified on the **RsslReactorOMMNIProviderRole** to perform all channel initialization for the user. After the user is logged in and has sent a source directory response, a channel event is returned to inform the application that the connection is ready.

- For further details on the **rsslReactorConnect** function, refer to Section 6.4.1.1.
- For further details on dispatching, refer to Section 6.7.

5.5 Perform Dictionary Download

If connected to a supporting LSEG Real-Time Advanced Distribution Hub, an OMM non-interactive provider can download the RWFFId and RWFEnum dictionaries to retrieve the appropriate dictionary information for providing field list content. An OMM non-interactive provider can use this feature to ensure they use the appropriate version of the dictionary or to encode data. To support the Provider Dictionary Download feature, the LSEG Real-Time Advanced Distribution Hub sends a Login response message containing the

SupportProviderDictionaryDownload login element. The dictionary request is sent using the Dictionary domain model.¹

The ETA offers several utility functions for downloading and managing a properly-formatted field dictionary. The provider can also use utility functions to encode the dictionary into an appropriate format for downloading or decoding.

For details on using the Login domain, expected message content, and available dictionary utility functions, refer to the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

5.6 Provide Content

After the **RsslReactorChannel** is established, it can begin pushing content to the LSEG Real-Time Advanced Distribution Hub. Each unique information stream should begin with an **RsslRefreshMsg**, conveying all necessary identification information for the content. Because the provider instantiates this information, a negative value **streamId** should be used for all streams. The initial identifying refresh can be followed by other status or update messages.

All content is encoded and decoded using the ETA Message C Codec Package and the ETA Data Package described in the Enterprise Transport API C Edition *Developers Guide*.

1. Because the provider instantiates this request, the application should use a **streamId** with a negative value. Additional details are provided in subsequent chapters.

5.7 Log Out and Shut Down

When the Consumer application is done retrieving or posting content, it can close the **RsslReactorChannel** by calling **rsslReactorCloseChannel**. This will close all item streams and log out the user. Prior to closing the **RsslReactorChannel**, the application should release any unwritten pool buffers to ensure proper memory cleanup.

If the application is done with the **RsslReactor**, the **rsslDestroyReactor** function can be used to shutdown and cleanup any **RsslReactor** resources.

- For details on closing an **RsslReactorChannel**, refer to Section 6.4.4.
- Shutting down an **RsslReactor** is described in Section 6.2.2.

5.8 Additional Non-Interactive Provider Details

The following locations discuss specific details about using OMM non-interactive providers and the ETA:

- The **rsslIVANIPProvider** application demonstrates one implementation of an OMM non-interactive provider application that uses ETA Value Added Components. The application's source code and **ReadMe** file have additional information about the specific implementation and behaviors.
- Chapter 6 provides a detailed look at the ETA Reactor.
- Chapter 8 provides more information about Administration Domain Model Representations.
- The Enterprise Transport API C Edition *Developers Guide* provides specific ETA encoder/decoder and transport usage information.
- The Enterprise Transport API C Edition *LSEG Domain Model Usage Guide* provides specific information about the Domain Message Models used by this application type.

6 Reactor Detailed View

6.1 Concepts

The **ETA Reactor** is a connection management and event processing component that can significantly reduce the amount of code an application must write to leverage OMM. This component helps simplify many aspects of a typical ETA application, regardless of whether the application is an OMM consumer, OMM interactive provider, or OMM non-interactive provider. The ETA Reactor can help manage consumer and non-interactive provider start up processing, including user log in, source directory establishment, and dictionary download. It also allows for dispatching of events to user-implemented callback functions, handles flushing of user-written content, and manages network pings on the user's behalf. Value Added domain representations are coupled with the reactor, allowing domain-specific callbacks to be presented with their respective domain representation for easier, more logical access to content. For a list and comparison of ETA and ETA Reactor functionalities, refer to Section 6.1.1.

The ETA Reactor internally depends on the Administration Domain Model Representation component. This allows the user to provide and consume the administrative RDM types in a more logical format. This additionally hides encoding and decoding of these domains from the Reactor user, all interaction is via a simple structural representation. More information about the Administration Domain Model Representation value added component is available in Chapter 8. The ETA Reactor also leverages several utility components, contained in the Value Added Utilities. This includes constructs like mutex locks, a simple queue, and memory buffers.

The ETA Reactor helps to manage the life-cycle of a connection on the user's behalf. When a channel is associated with a reactor, the reactor performs all necessary transport level initialization and alerts the user, via a callback, when the connection is up, ready for use, or is down. An application can simultaneously run multiple unique reactor instances, where each reactor instance can associate and manage a single channel or multiple channels. This functionality allows users to quickly and easily horizontally scale their application to leverage multi-core systems or distribute content across multiple connections.

Each instance of the ETA Reactor leverages multiple threads to help manage inbound and outbound data efficiently. The following figure illustrates a high-level view of the reactor threading model.

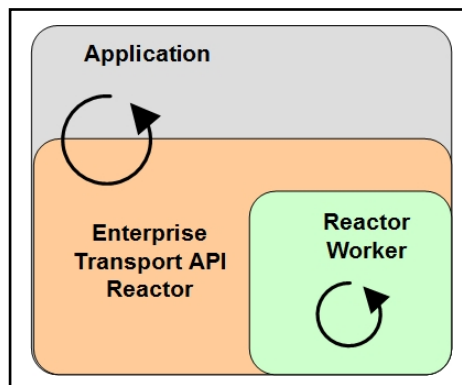


Figure 5. ETA Reactor Thread Model

There are two main threads associated with each ETA Reactor instance. The application thread is the main driver of the reactor; all event dispatching (e.g., reading), callback processing, and submitting of data to the ETA is done from this thread. Such architecture reduces latency and simplifies any threading model associated with user-defined callback functions – because callbacks happen from the application thread, a single-threaded application does not need to have additional mutex locking. The ETA Reactor also leverages an internal worker thread. The worker thread flushes any queued outbound data and manages outbound network pings for all channels associated with the Reactor.

The application drives the reactor with the use of a dispatch function. The dispatch function reads content from the network, performs some light processing to handle inbound network pings, and provides the information to the user through a series of per-channel, user-defined callback functions. Callback functions are separated based on whether they are reactor callbacks or channel callbacks. Channel callbacks are separated by domain, with a default callback where all unhandled domains or non-OMM content are provided to the user. The application can choose whether to dispatch on a single channel or across all channels managed by the reactor. The application can leverage an I/O notification mechanism (e.g. select, poll) or periodically call dispatch – it is all up to the user.

6.1.1 Functionality: Enterprise Transport API Versus Enterprise Transport API Reactor

FUNCTIONALITY	ETA	ETA REACTOR
Automatic Flushing of Data	***	X
Controlled Fragmentation and Assembly of Large Messages	X	X
Controlled Locking / Threading Model	X	X
Controlled Message Buffers with Ability to Change During Runtime	X	X
Controlled Message Packing	X	X
Downloading Field Dictionary	***	X
Loading Field Dictionary File	***	X
Network Ping Management	***	X
Programmatic Configuration	X	X
Programmatic Logging	X	X
Requesting Source Directory	***	X
Round Trip Latency Monitoring	***	For particular roles: <ul style="list-style-type: none"> • c: consumer • nip: Non-Interactive provider • p: provider
Session Management	***	X
Support for Unified and Segmented Network Connection Types	X	X
User-Defined Callbacks for Data	***	X
User Login	***	X
***: ETA users can implement this functionality themselves. They can also use or modify the ETA Reactor functionality.		

Table 2: ETA Functionality and ETA Reactor Comparison

6.1.2 Reactor Error Handling

The **RsslErrorInfo** structure is used to return error or warning information to the application. This can be returned from the various reactor functions as well as part of a callback function.

- If returned directly from a reactor function: an error occurred while processing in that function.
- If returned as part of a callback function: an error has occurred on one of the channels managed by the reactor.

RsslErrorInfo members are as follows:

PROPERTY	DESCRIPTION
<code>rsslErrorInfoCode</code>	An informational code about this error. Indicates whether it reports a failure condition or is intended to provide non-failure-related information to the user. For details on available codes, refer to Table 21.
<code>rsslError</code>	Returns an rsslError structure (i.e., the underlying error information from the Enterprise Transport API). rsslError includes a pointer to the RsslChannel on which the error occurred, both a Enterprise Transport API and a system error number, and more descriptive error text. The rsslError and its values are described in the Enterprise Transport API C Edition <i>Developers Guide</i> .
<code>errorLocation</code>	Provides information about the file and line on which the error occurred. Detailed error text is provided via the rsslError portion of this structure. RsslErrorInfo.errorLocation length is limited to 1,024 bytes.

Table 3: RsslErrorInfo Structure Members

6.1.3 Reactor Error Info Codes

It is important that the application monitors return values from the **RsslReactor** callbacks and functions. Error codes indicate whether the returned **RsslErrorInfo** is the result of a failure condition or is simply providing information regarding a successful operation.

RETURN CODE	DESCRIPTION
<code>RSSL_EIC_FAILURE</code>	A general failure has occurred. The RsslErrorInfo code contains more information about the specific error.
<code>RSSL_EIC_SUCCESS</code>	Indicates a success code. Used to inform the user of success and provide additional information.

Table 4: Reactor Error Info Codes

6.1.4 Enterprise Transport API Reactor Application Lifecycle

The following figure depicts the typical lifecycle of an application using the Enterprise Transport API Reactor, as well as associated function calls. Subsequent sections in this document provide more detailed information.

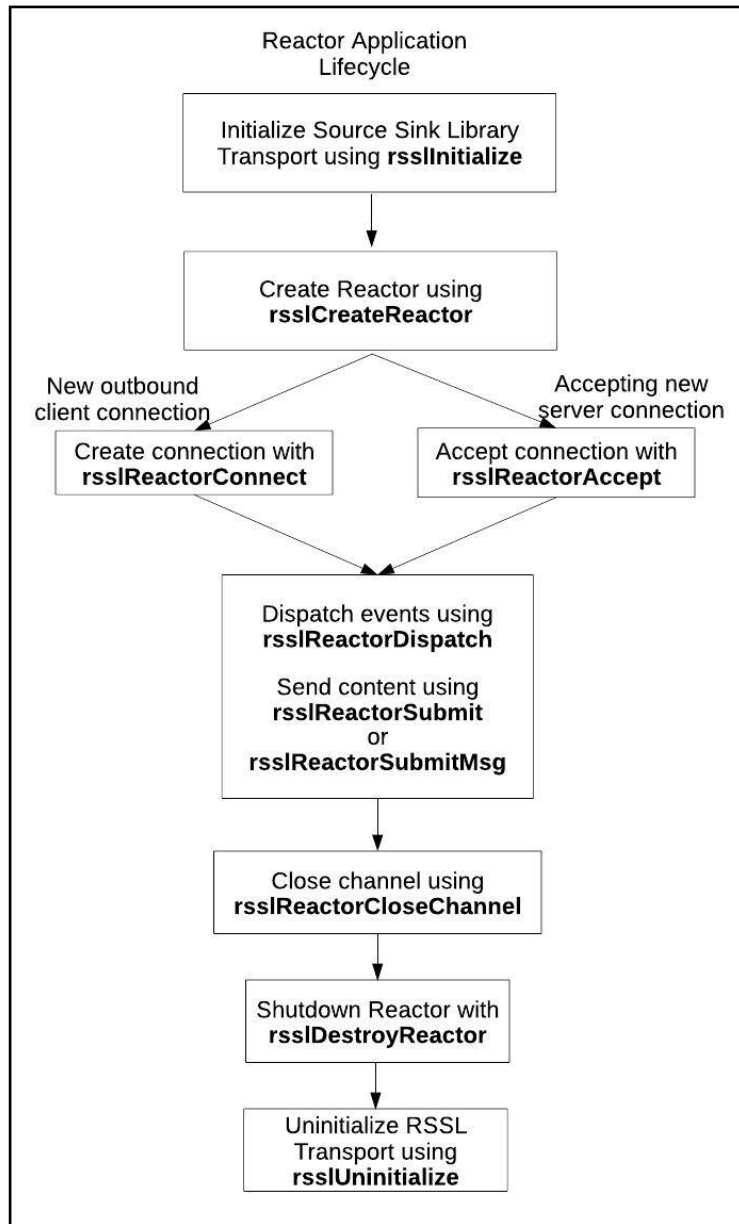


Figure 6. Enterprise Transport API Reactor Application Lifecycle

6.2 Reactor Use

This section describes use of **RsslReactor**. The **RsslReactor** manages **RsslReactorChannels** (described in Section 6.3). An understanding of both constructs is necessary for application writers.

Before creating any **RsslReactor** instance, the user must ensure that the Enterprise Transport API has been properly initialized. This is accomplished through the use of the **rsslInitialize** function, as documented in the *Enterprise Transport API C Edition Developers Guide*. Because the **RsslReactor** internally leverages multiple threads, the **RSSL_LOCK_GLOBAL_AND_CHANNEL** option must be specified in the call to **rsslInitialize**. After the Enterprise Transport API has been properly initialized, the application can create an **RsslReactor** instance. The **RsslReactor** is represented by a structure as defined in the following table.

NOTE: An application can leverage multiple **RsslReactor** instances to scale across multiple cores and distribute their **RsslReactorChannels** as needed.

PROPERTY	DESCRIPTION
eventFd	Represents a file descriptor that can be used in some kind of I/O notification mechanism (e.g. select, poll). This file descriptor is associated with RsslReactorChannel connection events or RsslReactor specific events, for example an RsslReactorChannel up or down notification. All RsslReactorChannel data event notification occurs on the RsslReactorChannel 's specific socketId , as detailed in Section 6.3.
userSpecPtr	A pointer that can be set by the user of the RsslReactor . This value can be set directly or via the creation options. This information can be useful for identifying a specific instance of an RsslReactor or coupling this RsslReactor with other user-defined information.

Table 5: RsslReactor Structure Members

6.2.1 Creating a Reactor

The lifecycle of an **RsslReactor** is controlled by the application, which controls creation and destruction of each reactor instance. The following sections describe creation functionality in more detail.

6.2.1.1 Reactor Creation

The creation of an **RsslReactor** instance can be accomplished through the use of the following function.

NOTE: Before the first use of any Enterprise Transport API Reactor functionality, the application must ensure that **rsslInitialize** has been called with the **RSSL_LOCK_GLOBAL_AND_CHANNEL** option.

FUNCTION NAME	DESCRIPTION
rsslCreateReactor	Creates an RsslReactor instance, including all necessary internal memory and threads. After creating the RsslReactor , RsslReactorChannels can be associated, as described in Section 6.3. Options are passed in via the RsslCreateReactorOptions , as defined in Section 6.2.1.2.

Table 6: RsslReactor Creation Function

6.2.1.2 RsslCreateReactorOptions Structure Members

STRUCTURE MEMBER	DESCRIPTION
cpuBindWorkerThread	<p>Specifies the CPU core in string format (CPU core ID or P:X C:Y T:Z format) for the internal Reactor worker thread binding.</p> <p>The application specifies the physical mapping which binds the thread to the specified physical processor, core, and thread (P:X C:Y T:Z).</p> <p>This syntax specifies a physical CPU to bind to. P refers to processor, C refers to core, and T refers to thread. If T is not specified (or T:#), the thread will be bound to all threads on the specified processor. If C is not specified (or C:#), the thread will be bound to all cores and threads on that processor.</p> <p>Specifying only one number causes a logical core ID to be bound instead of a physical one. If the value is not set (default), then there is no limit of the binding processor cores for the Reactor worker thread.</p> <p>Reactor API leverages rsslBindThread method to bind threads. Reactor creation will fail when rsslBindThread returns an error.</p>
debugBufferSize	Specifies the buffer size for the Reactor debug logs. For more details, see Section 6.2.1.4.
debugLevel	Specifies the Reactor debug logs level. Debug level is disabled by default. For more details, see Section 6.2.1.4.
dispatchDecodeMemoryBufferSize	The size, in bytes, of an internally created memory buffer. The memory buffer is used by the RsslReactor when performing any necessary message decoding required for callbacks. When cleared, defaults to 65,536 bytes.
maxEventsInPool	Specifies the maximum number of event objects in the event's pool.
port	Deprecated. RsslReactor now chooses an ephemeral port upon creation. Any values specified in this parameter are ignored.
reissueTokenAttemptInterval	The time (in milliseconds) that the RsslReactor waits before attempting to reissue the token. The minimum interval is 1000 milliseconds, while the default setting is 5000.
reissueTokenAttemptLimit	The maximum number of times the RsslReactor attempts to reissue the token. If set to default (i.e., -1), there is no maximum limit.
restEnableLog	Enables REST request/response logging via outputstream to either a file or standard output (stdout). Application may specify where to output the logs using the restLogOutputStream member. Ability to dynamically enable or disable this parameter is supported.
restEnableLogViaCallback	Enables REST request/response logging via callback. This parameter works in conjunction with setting the callback: restLoggingCallback . That is, logs are available by servicing callback: restLoggingCallback . Ability to dynamically enable or disable this parameter is supported.

Table 7: RsslCreateReactorOptions Structure Members

STRUCTURE MEMBER	DESCRIPTION
restProxyOptions	<p>Specifies the proxy settings for REST requests that the RTSDK API uses to discover service information (specified by serviceDiscoveryURL) or to obtain an authentication token (specified by tokenServiceURL). This proxy is used when both mandatory fields (proxyHostName and proxyPort) are specified. For more information on structure RsslProxyOpts, refer to the Enterprise Transport API <i>C Edition Developers Guide</i>.</p> <p>NOTE: This proxy overrides the proxy settings for REST requests in the RsslReactorConnectOptions (rsslConnectOptions.proxyOpts), RsslReactorServiceDiscoveryOptions and RsslReactorOAuthCredentialRenewalOptions.</p> <p>For more details on RsslReactorConnectOptions, refer to Section 6.4.1.2.</p> <p>For more details on RsslReactorServiceDiscoveryOptions, refer to Section 6.10.1.2.</p> <p>For more details on RsslReactorOAuthCredentialRenewalOptions, refer to Section 6.10.2.4.</p>
restLogOutputStream	Redirects REST logs (enabled by restEnableLog) to a specified file or stream. If this value was not set the log would be put out to standard output (stdout).
restRequestTimeout	<p>Specifies the timeout (in seconds) for token service and service discovery request. If the request times out, the Enterprise Transport API Reactor resends the token reissue and the timeout restarts. When using the rsslReactorConnect() method, if the request times out, the Reactor does not retry.</p> <p>If set to 0, there is no timeout. By default, the Enterprise Transport API behaves as if set to 90 seconds.</p>
serviceDiscoveryURL	Specifies the URL of Delivery Platform that the RTSDK API uses to discover service information such as the host and port to which the API connects to retrieve real-time data from the Real-Time solution/offering.
tokenReissueRatio	Specifies a ratio to multiply the access token's expiration time (in seconds) to determine the length of time the RsslReactor waits before retrieving a new access token and refreshing its connection to Real-Time - Optimized. The valid range is from 0.05 to 0.95 . By default, the Enterprise Transport API behaves as if set to 0.8 .
tokenServiceURL	Specifies the URL of Delivery Platform that the RTSDK API uses to obtain an authentication token.
userSpecPtr	A pointer that can be set by the application. This value is preserved and stored in the userSpecPtr of the RsslReactor returned from rsslCreateReactor . This information can be useful for coupling this RsslChannel with other user-created information, such as a watch list associated with this connection.

Table 7: **RsslCreateReactorOptions** Structure Members(Continued)

6.2.1.3 RsslCreateReactorOptions Utility Function

The Enterprise Transport API provides the following utility function for use with the **RsslCreateReactorOptions**.

FUNCTION NAME	DESCRIPTION
rsslClearCreateReactorOptions	Clears the RsslCreateReactorOptions structure. Useful for structure reuse.

Table 8: **RsslCreateReactorOptions** Utility Function

6.2.1.4 Reactor Debugger Options

The **ReactorDebuggerOptions** interface provides methods that set debugging levels for the Reactor and the output stream for the debugging information.

The debugging levels available for the Reactor are provided in the **RsslReactorDebuggerLevels** enumeration.

DEBUGGING LEVEL	DESCRIPTION
RSSL_RC_DEBUG_LEVEL_NONE	No debugging information is logged.
RSSL_RC_DEBUG_LEVEL_CONNECTION	Provides debugging points for connection events such as CHANNEL_UP , CHANNEL_DOWN , etc.
RSSL_RC_DEBUG_LEVEL_EVENTQUEUE	Allows obtaining information about the number of events in the Reactor queue and the number of events to be dispatched.
RSSL_RC_DEBUG_LEVEL_TUNNELSTREAM	Provides debugging points for TunnelStream events.

Table 9: Reactor Debugging Levels

By default, the output is directed to **debugInfoBuffer** with a default capacity of 1024 KB. The minimum buffer size that the user can set is 500 KB.

If this limit is exceeded, debugging messages are no longer written into the buffer until the user requests the already logged information. In this case the buffer is cleared and more information can be logged.

6.2.2 Destroying a Reactor

The lifecycle of an **RsslReactor** is controlled by the application, which controls creation and destruction of each reactor instance. The following sections describe the destruction functionality in more detail.

6.2.2.1 Reactor Destruction

When the application no longer requires an **RsslReactor** instance, it can destroy it using the following function.

FUNCTION NAME	DESCRIPTION
<code>rsslDestroyReactor</code>	Destroys an RsslReactor instance, including all internal memory and threads. This also sends RsslReactorChannelEvents , indicating channel down, to all RsslReactorChannels associated with this RsslReactor .

Table 10: RsslReactor Destruction Function

6.2.2.2 Reactor Creation and Destruction Example

```
RsslCreateReactorOptions reactorCreateOptions;

/* Use of reactors requires that RSSL be initialized with both global
 * and per-channel locks. */
ret = rsslInitialize(RSSL_LOCK_GLOBAL_AND_CHANNEL, &rsslError);

rsslClearCreateReactorOptions (&reactorCreateOptions);
```

```

/* Create the RsslReactor. */
pReactor = rsslCreateReactor(&reactorCreateOptions, &rsslErrorInfo);

/* Any use of the reactor occurs here -- see following sections for all other functionality */

/* Destroy the RsslReactor. */
ret = rsslDestroyReactor(pReactor, &rsslErrorInfo);

/* Uninitialize RSSL. */
ret = rsslUninitialize();

```

Code Example 1: Reactor Creation and Destruction Example

6.3 Reactor Channel Use

The **RsslReactorChannel** structure is used to represent a connection that can send or receive information across a network. This structure is used to represent a connection, regardless of whether it is an outbound connection or a connection accepted by a listening socket via an **RsslServer**. The **RsslReactorChannel** is the application's point of access, used to perform any action on the connection that it represents (e.g., dispatching events, writing, disconnecting, etc.). See the subsequent sections for more information about **RsslReactorChannel** and how to associate with an **RsslReactor**.

NOTE: Only Enterprise Transport API Reactor functions, like those defined in this chapter, should be called on a channel managed by an **RsslReactor**.

The following table describes the members of the **RsslReactorChannel** structure.

STRUCTURE MEMBER	DESCRIPTION
hostname	Provides the name of the host to which a consumer or NIP application connects.
majorVersion	When an RsslReactorChannel is up (RSSL_RC_CET_CHANNEL_UP), this is populated with the major version number associated with the content sent on this connection. Typically only minor version increases are associated with a fully backward compatible change or extension. The Enterprise Transport API Reactor will leverage the versioning information for any content it is encoding or decoding. Proper use of versioning should be handled by the application for any other application encoded or decoded content. For more information on versioning, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
minorVersion	When an RsslReactorChannel is up (RSSL_RC_CET_CHANNEL_UP), this is populated with the minor version number associated with the content sent on this connection. Typically, a minor version increase is associated with a fully backward compatible change or extension. The Enterprise Transport API Reactor will leverage the versioning information for any content it is encoding or decoding. Proper use of versioning should be handled by the application for any other application encoded or decoded content. For more information on versioning, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
oldSocketId	It is possible for a file descriptor to change over time, typically due to some kind of connection keep-alive mechanism. If this occurs, this is typically communicated via a callback indicating RSSL_RC_CET_FD_CHANGE . The previous RsslReactorChannel is stored in oldSocketId so the application can properly handle the change.
port	Provides the server port number to which the consumer or NIP application connects.

Table 11: RsslReactorChannel Structure Members

STRUCTURE MEMBER	DESCRIPTION
protocolType	<p>When an RsslReactorChannel is up (RSSL_RC_CET_CHANNEL_UP), this is populated with the protocolType associated with the content being sent on this connection. If the server indicates a protocolType that does not match the protocolType specified by the client, the connection is rejected.</p> <p>The Enterprise Transport API Reactor will leverage the versioning information for any content it is encoding or decoding. Proper use of versioning should be handled by the application for any other application encoded or decoded content. For more information on versioning, refer to the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>
pRsslChannel	A pointer to the underlying RsslChannel structure, as defined in the Enterprise Transport API C Edition <i>Developers Guide</i> , mainly for reference purposes. All operations should be performed using the Enterprise Transport API Reactor functionality; the application should not use this RsslChannel directly with any RSSL Transport functionality.
pRsslServer	A pointer to the underlying RsslServer structure, as defined in the Enterprise Transport API C Edition <i>Developers Guide</i> , mainly for reference purposes. This is populated only if the channel was created via the rsslReactorAccept function, as described in Section 6.4.1.7.
socketId	Represents a file descriptor that can be used in some kind of I/O notification mechanism (e.g. select, poll) to alert users when dispatch is required on a specific RsslReactorChannel . This is the file descriptor associated with this end of the network connection; the file descriptor value may be different from the other end of the connection.
userSpecPtr	A pointer that can be set by the user of the RsslChannel . This value can be set directly or via the RsslReactorConnectOptions and RsslReactorAcceptOptions . This information can be useful for coupling this RsslReactorChannel with other user-created information, such as a watch list associated with this connection.

Table 11: RsslReactorChannel Structure Members (Continued)

6.3.1 Reactor Channel Roles

An **RsslReactorChannel** can be configured to fulfill several specific roles, which overlap with the typical OMM application types. Provided role definitions include:

- **RsslReactorOMMConsumerRole** for OMM consumer applications
- **RsslReactorOMMProviderRole** for OMM interactive provider applications
- **RsslReactorOMMNIPProviderRole** for OMM non-interactive provider applications

All roles have the same common element, **RsslReactorChannelRoleBase**.

6.3.1.1 RsslReactorChannelRoleBase Structure

RsslReactorChannelRoleBase contains information and callback functions common to all role types and consists of the following members:

STRUCTURE MEMBER	DESCRIPTION
channelEventCallback	This RsslReactorChannel 's user-defined callback function to handle all RsslReactorChannel specific events, like RSSL_RC_CET_CHANNEL_UP or RSSL_RC_CET_CHANNEL_DOWN . This callback function is required for all role types. This callback is defined in more detail in Section 6.7.2.
defaultMsgCallback	This RsslReactorChannel 's user-defined callback function to handle RsslMsg content not handled by another domain-specific callback function. This callback function is required for all role types and is defined in more detail in Section 6.7.2.
roleType	The role type enumeration value, as defined in Section 6.3.1.2.

Table 12: RsslReactorChannelRoleBase Structure Members

6.3.1.2 roleType Enumerations

ENUMERATED NAME	DESCRIPTION
RSSL_RC_RT_INIT	Role is not specified. This is intended for structure initialization only.
RSSL_RC_RT_OMM_CONSUMER	Indicates that the RsslReactorChannel should act as a consumer.
RSSL_RC_RT_OMM_NI_PROVIDER	Indicates that the RsslReactorChannel should act as a non-interactive provider.
RSSL_RC_RT_OMM_PROVIDER	Indicates that the RsslReactorChannel should act as an interactive provider.

Table 13: RsslReactorChannelRoleBase.roleType Enumerated Values

6.3.2 Reactor Channel Role: OMM Consumer

When an **RsslReactorChannel** is acting as an OMM consumer application, it connects to an OMM interactive provider. As part of this process it is expected to perform a login to the system. After the login is completed, the consumer acquires a source directory, which provides information about the available services and their capabilities. Additionally, a consumer can download or load field dictionaries, providing information to help decode some types of content. The messages that are exchanged during this connection establishment process are administrative RDMs and are described in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

An **RsslReactorChannel** in a consumer role helps to simplify this connection process by exchanging these messages on the user's behalf. The user can choose to provide specific information or leverage a default populated message, which uses the information of the user currently logged into the machine running the application. In addition, the ETA Reactor allows the application to specify user-defined callback functions to handle the processing of received messages on a per-domain basis.

6.3.2.1 OMM Consumer Role

When creating an **RsslReactorChannel**, this information can be specified with the **RsslReactorOMMConsumerRole** structure as follows:

STRUCTURE MEMBER	DESCRIPTION
base	The role base structure, as defined in Section 6.3.1.1.
clientId	This member has been deprecated. Instead, use the pOAuthCredential member to set all Delivery Platform token service request credentials.
dictionaryDownloadMode	Informs the RsslReactorChannel of the method to use when requesting dictionaries. Allowable modes are defined in Section 6.3.2.2.
dictionaryMsgCallback	This RsslReactorChannel 's user-defined callback function to handle dictionary message content. If not specified, all received dictionary messages will be passed to the defaultMsgCallback . <ul style="list-style-type: none"> For more details on this callback, refer to Section 6.7.2. Dictionary messages are described in Section 8.5.
directoryMsgCallback	This RsslReactorChannel 's user-defined callback function to handle directory message content. If not specified, all received directory messages will be passed to the defaultMsgCallback . <ul style="list-style-type: none"> For more details on this callback, refer to Section 6.7.2. Directory messages are described in Section 8.4.
loginMsgCallback	This RsslReactorChannel 's user-defined callback function to handle login message content. If not specified, all received login messages will be passed to the defaultMsgCallback . <ul style="list-style-type: none"> For more details on this callback, refer to Section 6.7.2. Login messages are described in Section 8.3.
loginRequestMsgCredentialCount	This is the total number of login message credentials present in pLoginRequestList .
oAuthCredentialCount	This is the total number of oAuth credentials configured in pOAuthCredentialList .
pDirectoryRequest	The RsslRDMDirectoryRequest (defined in Section 8.4.1) sent during the connection establishment process. This can be populated with specific source directory request information or invoke the rsslInitDefaultRDMDirectoryRequest function to populate with default information. <ul style="list-style-type: none"> If this parameter is specified, a pDirectoryRequest is required. If this parameter is empty, a directory request is not sent to the system.
pOAuthCredential	Specifies the credentials for an application that makes a Delivery Platform token service request. This is required when connecting to an LSEG Real-Time Advanced Distribution Server in the cloud. See Section 6.10.2.1.
pOAuthCredentialList	This is an array of RsslReactorOAuthCredential pointers that defines oAuth credentials used with Session Management. This is a 0 indexed array. If specified, this array will override the credential set defined in pOAuthCredential . Connections will have the credentials mapped to them with RsslReactorConnectInfo.oAuthCredentialIndex .
pLoginRequest	The RsslRDMLLoginRequest (defined in Section 8.3.1) sent during the connection establishment process. This can be populated with a user's specific information or invoke the rsslInitDefaultRDMLLoginRequest function to populate with default information. If this parameter is empty, a login is not sent to the system; useful for systems that do not require a login. Use pLoginRequest when the application needs the Enterprise Transport API Reactor to send the initial login request.

Table 14: RsslReactorOMMConsumerRole Structure Members

STRUCTURE MEMBER	DESCRIPTION
pLoginRequestList	This is an array of RsslReactorLoginRequestMsgCredential pointers that defines login messages to be used with all channels. This is a 0 indexed array. This can also define an optional per-channel login message credential callback that will be called on reconnection. If specified, this array will override the default login messages for all configured connections both in the reactorConnectionList and warm standby groups configured in RsslReactorConnectOptions .
watchlistOptions	Configurable options for the consumer watchlist. Options are described in more detail in Section 6.3.2.3.

Table 14: RsslReactorOMMConsumerRole Structure Members (Continued)

6.3.2.2 OMM Consumer Role Dictionary Download Modes

There are several dictionary download options available to an **RsslReactorChannel**. The application can determine which option is desired and specify using the **RsslReactorOMMConsumerRole.dictionaryDownloadMode** parameter.

ENUMERATED NAME	DESCRIPTION
RSSL_RC_DICTIONARY_DOWNLOAD_FIRST_AVAILABLE	The RsslReactor will search received directory messages for the RDMFieldDictionary (RWFFId) and the enumtype.def (RWFEnum) dictionaries. Once found, the RsslReactor will request these dictionaries for the application. After transmission is completed, the streams are closed because this content does not update.
RSSL_RC_DICTIONARY_DOWNLOAD_NONE	The RsslReactor will not request dictionaries for this RsslReactorChannel . This is typically used when the application has loaded a file-based dictionary or has acquired the dictionary elsewhere.

Table 15: RsslReactorOMMConsumerRole.dictionaryDownloadMode Enumerated Values

6.3.2.3 OMM Consumer Role Watchlist Options

The consumer may enable an internal watchlist and configure behaviors. For more detail on the consumer watchlist feature, refer to Section 2.4.

OPTION	DESCRIPTION
enableWatchlist	Enables the watchlist.
itemCountHint	Can improve performance when used with the watchlist. If possible, set this to the approximate number of item requests the application expects to open.
maxOutstandingPosts	Sets the maximum allowable number of on-stream posts waiting for acknowledgment before the reactor disconnects.
obeyOpenWindow	Sets whether the RsslReactor obeys the OpenWindow of services advertised in a provider's Source Directory response.
postAckTimeout	Sets the time (in milliseconds) a stream waits to receive an ACK for an outstanding post before forwarding a negative acknowledgment RsslAckMsg to the application.
requestTimeout	Sets the time (in milliseconds) the watchlist waits for a response to a request.

Table 16: OMM Consumer Role Watchlist Options

6.3.2.4 OMM Consumer Role Utility Method

The Enterprise Transport API provides the following utility function for use with the **RsslReactorOMMConsumerRole**.

FUNCTION NAME	DESCRIPTION
<code>rsslClearOMMConsumerRole</code>	Clears the RsslReactorOMMConsumerRole structure. Useful for structure reuse.

Table 17: RsslReactorOMMConsumerRole Utility Function

6.3.3 Reactor Channel Role: OMM Provider

When an **RsslReactorChannel** is acting as an OMM provider application, it allows connections from OMM consumer applications. As part of this process it is expected to respond to login requests and source directory information requests. Additionally, a provider can optionally allow consumers to download field dictionaries. Messages exchanged during this connection establishment process are administrative RDMs and are described in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*

An **RsslReactorChannel** in an interactive provider role allows the application to specify user-defined callback functions to handle the processing of received messages on a per-domain basis.

6.3.3.1 OMM Provider Role Members

When creating an **RsslReactorChannel**, this information can be specified with the **RsslReactorOMMProviderRole** structure, as follows:

STRUCTURE MEMBER	DESCRIPTION
<code>base</code>	The role base structure, as defined in Section 6.3.1.1.
<code>dictionaryMsgCallback</code>	This RsslReactorChannel 's user-defined callback function to handle dictionary message content. If unspecified, all received dictionary messages will be passed to the defaultMsgCallback . <ul style="list-style-type: none"> For further details on this callback, refer to Section 6.7.2. Dictionary messages are described in Section 8.5.
<code>directoryMsgCallback</code>	This RsslReactorChannel 's user-defined callback function to handle directory message content. If unspecified, all received directory messages will be passed to the defaultMsgCallback . <ul style="list-style-type: none"> For further details on this callback, refer to Section 6.7.2. Directory messages are described in Section 8.4.
<code>loginMsgCallback</code>	This RsslReactorChannel 's user-defined callback function to handle login message content. If unspecified, all received login messages are passed to the defaultMsgCallback . <ul style="list-style-type: none"> For further details on this callback, refer to Section 6.7.2. Login messages are described in Section 8.3.
<code>tunnelStreamListenerCallback</code>	This RsslReactorChannel 's user-defined callback for accepting or rejecting tunnel streams. For further details on this callback, refer to Section 6.9.6.

Table 18: RsslReactorOMMProviderRole Structure Members

6.3.3.2 OMM Provider Role Utility Function

The Enterprise Transport API provides the following utility function for use with the **RsslReactorOMMProviderRole**.

FUNCTION NAME	DESCRIPTION
<code>rsslClearOMMProviderRole</code>	Clears the RsslReactorOMMProviderRole structure. Useful for structure reuse.

Table 19: RsslReactorOMMProviderRole Utility Function

6.3.4 Reactor Channel Role: OMM Non-Interactive Provider

When an **RsslReactorChannel** acts as an OMM non-interactive provider application, it connects to an LSEG Real-Time Advanced Distribution Hub and logs into the system. After login, the non-interactive provider publishes a source directory, which provides information about the available services and their capabilities. Messages exchanged while establishing the connection are administrative RDMS and are described in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

An **RsslReactorChannel** in a non-interactive provider role helps to simplify this connection process by exchanging these messages on the user's behalf. The user can choose to provide specific information or leverage a default populated message, which uses the information of the user currently logged into the machine running the application. In addition, the ETA Reactor allows the application to specify user-defined callback functions to handle the processing of received messages on a per-domain basis.

6.3.4.1 OMM Non-Interactive Role Members

When creating an **RsslReactorChannel**, this information can be specified with the **RsslReactorOMMNIPProviderRole** structure, as follows:

PROPERTY	DESCRIPTION
<code>base</code>	The role base structure, as defined in Section 6.3.1.1.
<code>pLoginRequest</code>	The RsslRDMLLoginRequest , defined in Section 8.3.1, sent when establishing a connection. You can populate this with a user's specific information or invoke the rsslInitDefaultRDMLLoginRequest function to populate with a default set of information. If empty, a login is not sent to the system; useful for systems that do not require a login.
<code>pDirectoryRefresh</code>	The RsslRDMDirectoryRefresh , defined in Section 8.4.2, sent when establishing a connection. You can populate this with specific source directory refresh information. <ul style="list-style-type: none"> If this parameter is specified, a pDirectoryRefresh is required. If this parameter is left empty, a refresh is not automatically sent to the system.
<code>loginMsgCallback</code>	The RsslReactorChannel 's user-defined callback function that handles login message content. If unspecified, all received login messages are passed to the defaultMsgCallback . For further details on this callback, refer to Section 6.7.2.

Table 20: RsslReactorOMMNIPProviderRole Structure Members

6.3.4.2 OMM Non-Interactive Provider Role Utility Function

The Enterprise Transport API provides the following utility function for use with the **RsslReactorOMMNIPProviderRole**.

FUNCTION NAME	DESCRIPTION
<code>rsslClearOMMNIPProviderRole</code>	Clears the RsslReactorOMMNIPProviderRole structure. Useful for structure reuse.

Table 21: RsslReactorOMMNIPProviderRole Utility Function

6.3.5 Reactor Channel: Role Union

A union is provided that allows use of any of the role structures. This is mainly for use within the **RsslReactor** implementation; however it is documented in the event that it can be useful to an application.

6.3.5.1 Union Members

UNION MEMBER	DESCRIPTION
base	The role's base structure, as defined in Table 12.
ommConsumerRole	The RsslReactorOMMConsumerRole , as defined in Section 6.3.2.
ommProviderRole	The RsslReactorOMMProviderRole , as defined in Section Section 6.3.3.
ommNIPProviderRole	The RsslReactorOMMNIPProviderRole , as defined in Section 6.3.4.

Table 22: RsslReactorChannelRole Union Members

6.3.5.2 Union Utility Function

The Enterprise Transport API provides the following utility function for use with the **RsslReactorOMMProviderRole**.

FUNCTION NAME	DESCRIPTION
rsslClearReactorChannelRole	Clears the RsslReactorChannelRole union.

Table 23: RsslReactorChannelRole Utility Function

6.4 Managing Reactor Channels

6.4.1 Adding Reactor Channels

A single **RsslReactor** instance can manage multiple **RsslReactorChannels**. An **RsslReactorChannel** can be instantiated as an outbound client style connection or as a connection that is accepted from an **RsslServer**. Thus, users can mix connection styles within or across Reactors and have consistent usage and behavior.

NOTE: A single **RsslReactor** can simultaneously manage **RsslReactorChannels** from **rsslReactorConnect** and **rsslReactorAccept**.

6.4.1.1 Reactor Connect

The **rsslReactorConnect** function will create a new **RsslReactorChannel** and associate it with an **RsslReactor**. This function creates a new outbound connection. The **RsslReactorChannel** is returned to the application via a callback, as described in Section 6.7.2, at which point it begins dispatching.

Client applications can specify that **RsslReactor** automatically reconnect an **RsslReactorChannel** whenever a connection fails. To enable this, the application sets the appropriate members of the **RsslReactorConnectOptions** structure. The application can specify that **RsslReactor** reconnect the **RsslReactorChannel** to the same host, or to one from among multiple hosts.

Consumer applications can combine the reactor connect feature with the watchlist feature to enable recovery of item streams across connections. For more information on the watchlist feature, refer to Section 2.4.

FUNCTION NAME	DESCRIPTION
rsslReactorConnect	Creates an RsslReactorChannel that makes an outbound connection to the configured host. This establishes a connection in a manner similar to the rsslConnect function, as described in the Enterprise Transport API C Edition <i>Developers Guide</i> . Connection options are passed in via the RsslReactorConnectOptions , as defined in Section 6.4.1.2. RsslReactorChannel specific information, such as the per-channel callback functions, the type of behavior, default RDM messages, and such are passed in via the RsslReactorChannelRole , as defined in Section 6.3.1.

Table 24: rsslReactorConnect Function

6.4.1.2 RsslReactorConnectOptions Structure Members

STRUCTURE MEMBER	DESCRIPTION
connectionCount	Specifies the number of connections listed in reactorConnectionList . If set to 0 , rsslConnectOptions is used.
connectionDebugFlags	Specifies a set of RsslDebugFlags for use when calling user-set debug callbacks as set by rsslSetDebugFunctions . If set to 0 , debug callbacks are not used
initializationTimeout	Specifies the amount of time (in seconds) to wait to successfully establish an RsslReactorChannel . If a RsslReactorChannel is not established in this timeframe, an event is dispatched to the application to indicate that the RsslReactorChannel is down.
reactorConnectionList	Specifies an array of connection information. When used with reconnectAttemptLimit , the RsslReactor attempts to connect to each host in the list with each reconnection attempt.
reconnectAttemptLimit	The maximum number of times the RsslReactor attempts to reconnect a channel when it fails. If set to -1 , there is no limit.

Table 25: RsslReactorConnectOptions Structure Members

STRUCTURE MEMBER	DESCRIPTION
reconnectMinDelay	Specifies the minimum length of time the RsslReactor waits (in milliseconds) before attempting to reconnect a failed channel. The time increases with each reconnection attempt, from reconnectMinDelay to reconnectMaxDelay .
reconnectMaxDelay	Specifies the maximum length of time the RsslReactor waits (in milliseconds) before attempting to reconnect a failed channel. The time increases with each reconnection attempt, from reconnectMinDelay to reconnectMaxDelay .
rsslConnectOptions	<p>Specifies information (rsslConnectOptions) about the host or network to which to connect, the type of connection to use, and other transport-specific configuration information associated with the underlying rsslConnect function.</p> <p>This is described in more detail in the Enterprise Transport API C Edition <i>Developers Guide</i>.</p> <p>NOTE: The proxy settings specified in rsslConnectOptions can be used to invoke REST requests such as service discovery and obtain an authentication token. They have lower precedence than the proxy settings specified in RsslCreateReactorOptions.restProxyOptions. For further details, refer to Section 6.2.1.2.</p>
statisticFlags	<p>Specifies RsslReactorChannelStatisticFlags which set the type of statistics reporting (if any) to perform on the RsslReactor channel. RsslReactorChannelStatisticFlags uses the following enums:</p> <ul style="list-style-type: none"> RSSL_RC_ST_NONE (or 0x0000): Turns off statistics reporting. RSSL_RC_ST_READ (or 0x0001): Turns on statistics reporting for the number of bytes read and the number of uncompressed bytes read. RSSL_RC_ST_WRITE (or 0x0002): Turns on statistics reporting for the number of bytes written and uncompressed bytes written. RSSL_RC_ST_PING (or 0x0004): Turns on statistics reporting for the number of pings received and the number of pings sent.
preferredHostOptions	Specifies the Preferred Host configuration for this connection. RsslPreferredHostOptions described in Section 6.4.1.4.

Table 25: RsslReactorConnectOptions Structure Members (Continued)

6.4.1.3 RsslReactorConnectInfo Structure Members

STRUCTURE MEMBER	DESCRIPTION
rsslConnectOptions	<p>Specifies information (RsslConnectOptions) about the host or network to which to connect, the type of connection to use, and other transport-specific configuration information associated with the underlying rsslConnect function.</p> <p>This is described in more detail in the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>
enableSessionManagement	Specifies whether the channel manages the authentication token on behalf of the user used to keep the session alive. Boolean. If set to true , the channel obtains the authentication token and refreshes it on behalf of user to keep session active. The default setting is false .
initializationTimeout	Specifies the amount of time (in seconds) to wait to successfully establish an RsslReactorChannel . If a RsslReactorChannel is not established in this timeframe, an event is dispatched to the application to indicate that the RsslReactorChannel is down.
location	Specifies the cloud location (e.g., us-east-1) of the service provider endpoint to which the RTSDK API establishes a connection. If location is not specified, the default setting is us-east-1 . In any particular cloud location, the Reactor connects to the endpoint that provides two available zones for the location (e.g., [us-east-1a , us-east-1b]).

Table 26: RsslReactorConnectInfo Structure Members

STRUCTURE MEMBER	DESCRIPTION
loginReqIndex	This specifies the login message to be used for this connection, contained in the pointer array of RsslReactorOMMConsumerRole.pLoginRequestList . If RsslReactorOMMConsumerRole.pLoginRequestList is specified, then this connection will use that login message and login credential callback when it connects. This must be less than RsslReactorOMMConsumerRole.pLoginRequestCount . This is a 0 indexed array. For details, see Section 6.5.
oAuthCredentialIndex	This specifies the oAuth credential to be used for this connection, contained in the pointer array of RsslReactorOMMConsumerRole.pOAuthCredentialList , if enableSessionManagement is turned on and RsslReactorOMMConsumerRole.oAuthCredentialCount has been set. This must be less than the number set in RsslReactorOMMConsumerRole.oAuthCredentialCount . This is a 0 indexed array. For details, see Section 6.5.
serviceDiscoveryRetryCount	The number of times the RsslReactor attempts to reconnect a channel before forcing the API to retry service discovery. See the description of RsslCreateReactorOptions.serviceDiscoveryURL in Section 6.2.1.2. This functionality works when the user sets enableSessionManagement to true, but is ignored when the user sets the values of UnifiedNetworkInfo.address and UnifiedNetworkInfo.serviceName in ConnectOptions directly. For details, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> . The default value is 3. After each 3 attempts to reconnect a channel, the RsslReactor will force to retry service discovery. The RsslReactor will not retry to get an endpoint from the service discovery when the value is 0.
pAuthTokenEventCallback	A callback function that receives RsslReactorAuthTokenEvents . The Reactor requests a token for the Consumer (i.e., disabling watchlist) and NiProvider applications to send login requests and reissues with the token.

Table 26: RsslReactorConnectInfo Structure Members (Continued)

6.4.1.4 RsslPreferredHostOptions

For the Preferred Host feature description, see Section 10.

STRUCTURE MEMBER	DESCRIPTION
enablePreferredHostOptions	Specifies if the Preferred Host feature is enabled. If set to RSSL_TRUE , then the preferred host feature will be used during reconnection, and, if configured, any timers will be set to automatically perform the Preferred Host operation.
detectionTimeSchedule	Specifies a cron-formatted string used to schedule Preferred Host operation attempts. The cron formatted string will use the system's clock to determine the time when the operation will be preformed. This will default to empty, meaning no scheduled time is set, and if both are specified, this will override the detectionTimeInterval . The Reactor is using supertinycron (https://github.com/exander77/supertinycron) for parsing the input string. For information about the supported string formatting, please refer to that library's documentation on GitHub.
detectionTimeInterval	Specifies a timeout interval, in seconds, that the Reactor will wait before attempting to switch to the configured Preferred Host. Optional, a value of 0 means that there is no timeout.
connectionListIndex	Specifies the index value of the preferred host connection information contained in RsslReactorConnectionOptions.reactorConnectionList . This is required, and will be the connection that the Reactor will fall back to when the Preferred Host Operation is performed. This value is ignored if a connection list is not specified, or if there is not a connection list and warm standby groups are configured.

Table 27: RsslReactorPreferredHostOptions Structure Members.

STRUCTURE MEMBER	DESCRIPTION
warmStandbyGroupListIndex	Specifies the index value of the preferred host Warm Standby Group contained in RsslReactorConnectionOptions.reactorWarmStandbyGroupList . This is required, and will be the connection that the Reactor will fall back to when the Preferred Host Operation is performed. This value is ignored if there are not any Warm Standby Groups configured.
fallBackWithInWSBGroup	This only applies if Warm Standby Groups are configured. If set to RSSL_TRUE , the preferred host operation will not switch from the currently connected Warm Standby Group, and will instead attempt to reset the currently active connection/services to their initially configured connections. If set to RSSL_FALSE , the reactor will attempt to switch to the configured preferred warm standby group, but will reset the current active connection/services if the Reactor Channel is currently connected to the preferred warm standby group.

Table 27: RsslReactorPreferredHostOptions Structure Members.

6.4.1.5 RsslReactorConnectOptions Utility Function

The Enterprise Transport API provides the following utility function for use with the **RsslReactorConnectOptions**.

FUNCTION NAME	DESCRIPTION
rsslClearReactorConnectOptions	Clears the RsslReactorConnectOptions structure. Useful for structure reuse.

Table 28: RsslReactorConnectOptions Utility Function

6.4.1.6 rsslReactorConnect Example

```

RsslReactorConnectOptions reactorConnectOpts;
RsslReactorOMMConsumerRole consumerRole;

RsslRDMLLoginRequest loginRequest;
RsslRDMDirectoryRequest directoryRequest;

/* Configure connection options.*/
rsslClearReactorConnectOptions(&reactorConnectOpts);
reactorConnectOpts.rsslConnectOptions.connectionInfo.unified.address = "localhost";
reactorConnectOpts.rsslConnectOptions.connectionInfo.unified.serviceName = "14002";

/* Configure a role for this connection as an OMM Consumer. */
rsslClearOMMConsumerRole(&consumerRole);

/* Set the functions to which rsslDispatch will deliver events. */
consumerRole.base.channelEventCallback = channelEventCallback;
consumerRole.base.defaultMsgCallback = defaultMsgCallback;
consumerRole.loginMsgCallback = loginMsgCallback;
consumerRole.directoryMsgCallback = directoryMsgCallback;
consumerRole.dictionaryMsgCallback = dictionaryMsgCallback;

/* Prepare a login request. Once the channel is initialized this message will be sent. */
rsslInitDefaultRDMLLoginRequest(&loginRequest, 1);

```

```

consumerRole.pLoginRequest = &loginRequest;

/* Prepare a directory request. Once the application has logged in, this message will be sent. */
rsslInitDefaultRDMDirectoryRequest(&directoryRequest, 2);
consumerRole.pDirectoryRequest = &directoryRequest;

/* Add the connection to the RsslReactor. */
ret = rsslReactorConnect(pReactor, &reactorConnectOpts, (RsslReactorChannelRole*)&consumerRole,
    &rsslErrorInfo);

```

Code Example 2: rsslReactorConnect Example

6.4.1.7 Reactor Accept

The **rsslReactorAccept** function creates a new **RsslReactorChannel** and associates it with an **RsslReactor**. This function accepts the connection from an already running **RsslServer**. The **RsslReactorChannel** will be returned to the application via a callback, as described in Section 6.7.2, at which point it can begin dispatching on the channel.

METHOD NAME	DESCRIPTION
rsslReactorAccept	<p>Creates an RsslReactorChannel by accepting it from an RsslServer. This establishes a connection in a manner similar to the rsslAccept function, as described in the Enterprise Transport API C Edition <i>Developers Guide</i>.</p> <ul style="list-style-type: none"> Connection options are passed in via RsslReactorAcceptOptions, as defined in Section 6.4.1.8. RsslReactorChannel-specific information (such as the per-channel callback functions, the type of behavior, default RDM messages, and etc.) are passed in via the RsslReactorChannelRole, as defined in Section 6.3.1.

Table 29: RsslReactorAccept Function

6.4.1.8 RsslReactorAcceptOptions Structure Members

STRUCTURE MEMBER	DESCRIPTION
rsslAcceptOptions	The RsslAcceptOptions associated with the underlying rsslAccept function. This includes an option to reject the connection as well as a userSpecPtr . This is described in more detail in the Enterprise Transport API C Edition <i>Developers Guide</i> .
initializationTimeout	The amount of time (in seconds) to wait for the successful connection establishment of an RsslReactorChannel . If a timeout occurs, an event is dispatched to the application to indicate that the RsslReactorChannel is down.

Table 30: RsslReactorAcceptOptions Structure Members

6.4.1.9 RsslReactorAcceptOptions Utility Function

The Enterprise Transport API provides the following utility function for use with the **RsslReactorAcceptOptions**.

FUNCTION NAME	DESCRIPTION
rsslClearReactorAcceptOptions	Clears the RsslReactorAcceptOptions structure. Useful for structure reuse.

Table 31: RsslReactorAcceptOptions Utility Function

6.4.1.10 rsslReactorAccept Example

```
RsslReactorAcceptOptions reactorAcceptOpts;
RsslReactorOMMProviderRole providerRole;

/* Configure accept options.*/
rsslClearReactorAcceptOptions (&reactorAcceptOpts);

/* Configure a role for this connection as an OMM Provider. */
rsslClearOMMProviderRole (&providerRole);
providerRole.base.channelEventCallback = channelEventCallback;
providerRole.base.defaultMsgCallback = defaultMsgCallback;
providerRole.loginMsgCallback = loginMsgCallback;
providerRole.directoryMsgCallback = directoryMsgCallback;
providerRole.dictionaryMsgCallback = dictionaryMsgCallback;

/* Add the connection to the RsslReactor. */
rsslClearReactorAcceptOptions (&reactorAcceptOpts);
ret = rsslReactorAccept (pReactor, pRsslServer, &reactorAcceptOpts,
                        (RsslReactorChannelRole*)&providerRole, &rsslErrorInfo);
```

Code Example 3: rsslReactorAccept Example

6.4.2 Modifying Reactor Channels

6.4.2.1 RsslReactorChannelIoctl

The **RsslReactorChannelIoctl** function can be used to change the configuration of the reactor channels. This function can take in Option enumeration values from **RsslIoctlCodes** (for details, see Enterprise Transport API C Edition Developers Guide, section 10.14.8) as well as **RsslReactorChannelIoctlCodes** enumerated codes. For Warm Standby channels, when **RsslReactorChannelIoctl** is called, any configuration change will be applied to all underlying connections.

OPTION ENUMERATION	DESCRIPTION
RSSL_REACTOR_CHANNEL_IOCTL_DIRECT_WRITE	This option allows the application to turn on or off the direct write functionality for the underlying rsslChannel. Enabling this feature will force ETA to attempt to flush any writes to the network immediately, avoiding the internal flushing queues. Value should be a pointer to RsslUInt32, and setting value to 1 will enable direct write, a value of 0 will disable it.
RSSL_REACTOR_CHANNEL_IOCTL_PREFERRED_HOST_OPTIONS	This option allows the application to change all of the Preferred Host options currently applied to the channel. Value should be a pointer to an RsslPreferredHostOptions structure (for details, see Table 25), and all structure members will be applied whenever RsslReactorChannelIoctl is called.

Table 32: RsslReactorChannelIoctlCodes Enumerations

6.4.2.2 RsslReactorPreferredHostInfo

The following table describes the values contained in the **RsslReactorPreferredHostInfo** structure. This information is available through the **rsslReactorGetChannelInfo** function, and is returned as a part of the **RsslReactorChannelInfo** structure. See Section 6.12.

STRUCTURE MEMBER	DESCRIPTION
enablePreferredHostOptions	Indicates if the Preferred Host feature is enabled. If set to RSSL_TRUE , then the preferred host feature will be used during reconnection, and, if configured, any timers will be set to automatically perform the Preferred Host operation.
detectionTimeSchedule	Indicates the cron-formatted string which is used to schedule Preferred Host operation attempts. The cron formatted string will use the system's clock to determine the time when the operation will be preformed. This will default to empty, meaning no scheduled time is set, and if both are specified, this will override the detectionTimeInterval .
detectionTimeInterval	Indicates a timeout interval, in seconds, that the Reactor will wait before attempting to switch to the configured Preferred Host. Optional, a value of 0 means that there is no timeout.
connectionListIndex	Indicates the index value of the preferred host connection information contained in RsslReactorConnectionOptions.reactorConnectionList . This is required, and will be the connection that the Reactor will fall back to when the Preferred Host Operation is performed. This value is ignored if a connection list is not specified, or if there is not a connection list and warm standby groups are configured.
warmStandbyGroupListIndex	Indicates the index value of the preferred host Warm Standby Group contained in RsslReactorConnectionOptions.reactorWarmStandbyGroupList . This is required, and will be the connection that the Reactor will fall back to when the Preferred Host Operation is performed. This value is ignored if there are not any Warm Standby Groups configured.
fallBackWithInWSBGroup	This only applies if Warm Standby Groups are configured. If set to RSSL_TRUE , the preferred host operation will not switch from the currently connected Warm Standby Group, and will instead attempt to reset the currently active connection/services to their initially configured connections. If set to RSSL_FALSE , the reactor will attempt to switch to the configured preferred warm standby group, but will reset the current active connection/services if the Reactor Channel is currently connected to the preferred warm standby group.
remainingDetectionTime	Indicates the remaining time, in seconds, until the Reactor will perform the next Preferred Host operation.

Table 33: RsslReactorPreferredHostInfo Structure Members

6.4.3 Reactor Channel Reconnection and Recovery Behaviors

For the client-based connections (Consumer and NIProvider), the Reactor can automatically reconnect upon loss of the underlying network connection. This behavior is controlled by the **reconnectAttemptLimit**, **reconnectMinDelay**, and **reconnectMaxDelay** configuration in **RsslReactorConnectOptions**.

For the delay options, for each disconnect, the reactor will start a **reconnectMinDelay**, and double it each subsequent connection failure until **reconnectMaxDelay** is reached.

For the **reconnectAttemptLimit** configuration, please see the following table for the behaviors.

NIProviders do not support Warm Standby or Preferred Host features.

	RECONNECTATTEMPTLIMIT VALUE		
	-1: INFINITE RECONNECT ATTEMPTS	0: NO RECONNECTION ATTEMPTS	N > 0: 1 OR MORE RECONNECTION ATTEMPTS
Single Connection	Reactor will initially attempt to connect to the connection indefinitely after the initial connection attempt or upon loss of the underlying network connection.	Reactor will attempt to connect to the configured server (first in the connection list, if specified), if this fails upon initial connect or the underlying network connection fails after establishing a connection, the channel will be in a CHANNEL_DOWN state, and must be cleaned up and fully closed with rsslReactorChannelClose .	Reactor will initially attempt to connect to the single connection N times after the initial connection or upon loss of the underlying network connection. This attempt count will be reset once the Reactor Channel is able to connect to a server. Once N attempts have been reached, the channel will be in a CHANNEL_DOWN state, and must be cleaned up and fully closed rsslReactorChannelClose .
Connection List with Preferred Host disabled	Reactor will round robin through the connection list, initially starting with the first connection and continuing to the next channel in the list if the attempt fails. After the lost of the underlying network connection, the reactor channel will attempt reconnecting to the next channel in the list. The Reactor will continue to attempt to reconnect infinitely until the channel is closed by the application. Behavior example: C1, C2, C3 configured connections Connection order: C1, C2, C3, repeat from beginning	Reactor will attempt to connect to the configured server (first in the connection list, if specified), if this fails upon initial connect or the underlying network connection fails after establishing a connection, the channel will be in a CHANNEL_DOWN state, and must be cleaned up and fully closed with rsslReactorChannelClose .	Reactor will initially attempt to connect first connection in the list, and will round robin through the list, attempting N times after the initial connection attempt or upon loss of the underlying network connection. This attempt count will be reset once the Reactor Channel is able to connect to a server. Once N attempts have been reached, the channel will be in a CHANNEL_DOWN state, and must be cleaned up and fully closed rsslReactorChannelClose . Behavior example: C1, C2, C3 configured connections Connection order: C1, C2, C3, repeat from beginning, until N total attempts.

Table 34: reconnectAttemptLimit Configuration Behaviors

	RECONNECTATTEMPTLIMIT VALUE		
	-1: INFINITE RECONNECT ATTEMPTS	0: NO RECONNECTION ATTEMPTS	N > 0: 1 OR MORE RECONNECTION ATTEMPTS
Connection List with Preferred Host enabled	<p>Reactor will attempt to connect to the preferred host index, and then will round robin through the connection list, alternating the preferred connection index between each non-preferred connection attempt.</p> <p>After the lost of the underlying network connection, the reactor channel will attempt reconnecting to the next channel in the list.</p> <p>The Reactor will continue to attempt to reconnect infinitely until the channel is closed by the application.</p> <p>Behavior example: C1, C2*, C3 configured, C2* is the preferred (index 1)</p> <p>Connection order: C2*, C1, C2*, C3, repeat from beginning</p>	<p>Reactor will attempt to connect to the preferred server index. if this fails upon initial connect or the underlying network connection fails after establishing a connection, the channel will be in a CHANNEL_DOWN state, and must be cleaned up and fully closed with rsslReactorChannelClose.</p>	<p>Reactor will initially attempt to connect to the preferred host index, and then will round robin through the connection list, alternating the preferred connection index between each non-preferred connection attempt. The reactor will attempt N times after the initial connection attempt or upon loss of the underlying network connection.</p> <p>This attempt count will be reset once the Reactor Channel is able to connect to a server.</p> <p>Once N attempts have been reached, the channel will be in a CHANNEL_DOWN state, and must be cleaned up and fully closed rsslReactorChannelClose.</p> <p>After the lost of the underlying network connection, the reactor channel will attempt reconnecting to the next channel in the list.</p> <p>Behavior example: C1, C2*, C3 configured, C2* is the preferred (index 1)</p> <p>Connection order: C2*, C1, C2*, C3, repeat from beginning, until N total attempts</p>

Table 34: reconnectAttemptLimit Configuration Behaviors

	RECONNECTATTEMPTLIMIT VALUE		
	-1: INFINITE RECONNECT ATTEMPTS	0: NO RECONNECTION ATTEMPTS	N > 0: 1 OR MORE RECONNECTION ATTEMPTS
Warm Standby, one or more groups and Preferred Host disabled	<p>Reactor will attempt to connect to the starting active server in the first configured WSB group, and then will attempt to connect to the standby servers once the starting active connection has been established.</p> <p>On a failed initial connection or upon loss of the underlying network connection, the starting active connection will attempt to reconnect infinitely, and all standby servers in the group's standby server list will attempt to reconnect indefinitely.</p> <p>The Reactor will continue to attempt to reconnect infinitely until the channel is closed by the application.</p>	<p>Reactor will attempt to connect to the starting active server in the first configured WSB group, and then will attempt to connect to the standby servers once the starting active connection has been established.</p> <p>For both the starting active connection and the standby connections, if the initial connection attempt fails or upon loss of the underlying network connection, the individual channels will be considered down.</p> <p>Once all configured connections are down, the channel will be in a CHANNEL_DOWN state, and be cleaned up and fully closed with rsslReactorChannelClose.</p>	<p>Reactor will attempt to connect to the starting active server in the first configured WSB group, and then will attempt to connect to the standby servers once the starting active connection has been established.</p> <p>On a failed initial connection or upon loss of the underlying network connection, the starting active connection will attempt to reconnect N times, and all standby servers in the group's standby server list will attempt to reconnect N times. For any given connection within a WSB group, after the N reconnection attempts, that connection will be marked as down, and the reactor will not attempt to connect any more on that connection.</p> <p>This attempt count will be reset for each underlying WSB connection when that connection is established to its configured server.</p> <p>Once all connections in the WSB group are marked as down, the reactor will send a DOWN_RECONNECTING event, and will attempt the next warm standby group index. If there are not any remaining WSB groups, the channel will be in a CHANNEL_DOWN state, and must be cleaned up and fully closed rsslReactorChannelClose.</p>

Table 35: reconnectAttemptLimit Configuration Warm Standby Behaviors

	RECONNECTATTEMPTLIMIT VALUE		
	-1: INFINITE RECONNECT ATTEMPTS	0: NO RECONNECTION ATTEMPTS	N > 0: 1 OR MORE RECONNECTION ATTEMPTS
Warm Standby, one or more groups with Channel List and Preferred Host disabled	<p>Reactor will attempt to connect to the starting active server in the first configured WSB group, and then will attempt to connect to the standby servers once the starting active connection has been established.</p> <p>The starting active connection will attempt to reconnect infinitely, and all standby servers in the group's standby server list will attempt to reconnect indefinitely on a failed initial connection or upon loss of the underlying network connection.</p> <p>The Reactor will continue to attempt to reconnect infinitely until the channel is closed by the application.</p>	<p>Reactor will attempt to connect to the starting active server in the first configured WSB group, and then will attempt to connect to the standby servers once the starting active connection has been established.</p> <p>For both the starting active connection and the standby connections, if the initial connection attempt fails or upon loss of the underlying network connection, the individual channels will be considered down.</p> <p>Once all configured connections are down, the channel will be in a CHANNEL_DOWN state, and be cleaned up and fully closed with rsslReactorChannelClose.</p>	<p>Reactor will attempt to connect to the starting active server in the first configured WSB group, and then will attempt to connect to the standby servers once the starting active connection has been established.</p> <p>On a failed initial connection or upon loss of the underlying network connection, the starting active connection will attempt to reconnect N times, and all standby servers in the group's standby server list will attempt to reconnect N times. For any given connection within a WSB group, once the N reconnection attempts have failed, that connection will be marked as down, and the reactor will not attempt to connect any more on that connection.</p> <p>This attempt count will be reset for each underlying WSB connection when that connection is established to its configured server.</p> <p>Once all connections in the WSB group are marked as down, the reactor will send a DOWN_RECONNECTING event, and will attempt the next warm standby group index. If there are not any remaining WSB groups, the channel will move to the configured channel list, and will round robin connections through that. At that point, the channel will follow the reconnection behaviors specified in the "Connection List, No Preferred Host" box on this row.</p> <p>Behavior example: WSB groups: G1, G2, G3 Channel List: C1, C2, C3 configured Connection order (WSB groups will move to the next group once all connections have been marked as down): G1, G2, G3, then once G3 is down: C1, C2, C3, repeat the channel list from beginning until N total channel list attempts have been reached.</p>

Table 35: reconnectAttemptLimit Configuration Warm Standby Behaviors

	RECONNECTATTEMPTLIMIT VALUE		
	-1: INFINITE RECONNECT ATTEMPTS	0: NO RECONNECTION ATTEMPTS	N > 0: 1 OR MORE RECONNECTION ATTEMPTS
Warm Standby, one or more groups with Preferred Host enabled	<p>Reactor will attempt to connect to the starting active server on the preferred host warm standby group index configured WSB group.</p> <p>The starting active connection will attempt to reconnect infinitely, and all standby servers in the group's standby server list will attempt to reconnect indefinitely on a failed initial connection or upon loss of the underlying network connection.</p> <p>The Reactor will continue to attempt to reconnect infinitely until the channel is closed by the application.</p>	<p>Reactor will attempt to connect to the starting active server on the preferred host WSB group index, and then will attempt to connect to the standby servers once the starting active connection has been established.</p> <p>For both the starting active connection and the standby connections, if the initial connection attempt fails or upon loss of the underlying network connection, the individual channels will be considered down.</p> <p>Once all configured connections are down, the channel will be in a CHANNEL_DOWN state, and be cleaned up and fully closed with rsslReactorChannelClose.</p>	<p>Reactor will attempt to connect to the starting active server on the preferred host WSB group index, and then will attempt to connect to the standby servers once the starting active connection has been established.</p> <p>On a failed initial connection or upon loss of the underlying network connection, the starting active connection will attempt to reconnect N times, and all standby servers in the group's standby server list will attempt to reconnect N times. For any given connection within a WSB group, once the N reconnection attempts have failed, that connection will be marked as down, and the reactor will not attempt to connect any more on that connection.</p> <p>This attempt count will be reset for each underlying WSB connection when that connection is established to its configured server.</p> <p>Once all connections in the WSB group are marked as down, the reactor will send a DOWN_RECONNECTING event, and will round robin through WSB groups, alternating the preferred connection WSB index between each non-preferred connection attempt. See below for an example order.</p> <p>Behavior example: WSB groups: G1, G2*, G3, C2* is the preferred (index 1) Connection Order (For WSB groups, the reactor will move on once all underlying connections in the WSB group have been marked as down): G2*, G1, G2*, G3, repeat from the beginning of this pattern.</p>

Table 35: reconnectAttemptLimit Configuration Warm Standby Behaviors

	RECONNECTATTEMPTLIMIT VALUE		
	-1: INFINITE RECONNECT ATTEMPTS	0: NO RECONNECTION ATTEMPTS	N > 0: 1 OR MORE RECONNECTION ATTEMPTS
Warm Standby, one or more groups with channel list and Preferred Host enabled	<p>Reactor will attempt to connect to the starting active server on the preferred host warm standby group index configured WSB group.</p> <p>The starting active connection will attempt to reconnect infinitely, and all standby servers in the group's standby server list will attempt to reconnect indefinitely on a failed initial connection or upon loss of the underlying network connection.</p> <p>The Reactor will continue to attempt to reconnect infinitely until the channel is closed by the application.</p>	<p>Reactor will attempt to connect to the starting active server on the preferred host WSB group index, and then will attempt to connect to the standby servers once the starting active connection has been established.</p> <p>For both the starting active connection and the standby connections, if the initial connection attempt fails or upon loss of the underlying network connection, the individual channels will be considered down.</p> <p>Once all configured connections are down, the channel will be in a CHANNEL_DOWN state, and be cleaned up and fully closed with rsslReactorChannelClose.</p>	<p>Reactor will attempt to connect to the starting active server on the preferred host WSB group index, and then will attempt to connect to the standby servers once the starting active connection has been established.</p> <p>On a failed initial connection or upon loss of the underlying network connection, the starting active connection will attempt to reconnect N times, and all standby servers in the group's standby server list will attempt to reconnect N times. For any given connection within a WSB group, after the N reconnection attempts, that connection will be marked as down, and the reactor will not attempt to connect any more on that connection.</p> <p>Once all connections in the WSB group are marked as down, the reactor will send a DOWN_RECONNECTING event, and will round robin through WSB groups, alternating the preferred connection WSB index between each non-preferred connection attempt. After attempting all of the non-preferred WSB groups, the Reactor will attempt the preferred WSB group, followed by one attempt on a channel in the channel list. The channel list order will also be controlled by the preferred channel list index. See below for an example order.</p> <p>Behavior example:</p> <p>WSB groups: G1, G2*, G3 (preferred group is G2 with index set to 1)</p> <p>Channel List: C1*, C2, C3 (preferred channel is C1 with index set to 0)</p> <p>Connection Order (For WSB groups, the reactor will move on once all underlying connections in the WSB group have been marked as down. The WSB channels will make N connection attempts before going down. All channel list connections will be attempted once, and then move back to the WSB groups.):</p> <p>G2*, G1, G2*, G3, G2*, C1*, G2*, G1, G2*, G3, G2*, C2, G2*, G1, G2*, G3, G2*, C1*, G2*, G1, G2*, G3, G2*, C3, repeat from the first row.</p>

Table 35: reconnectAttemptLimit Configuration Warm Standby Behaviors

6.4.4 Removing Reactor Channels

6.4.4.1 rsslReactorCloseChannel Function

You use the following function to remove an **RsslReactorChannel** from an **RsslReactor** instance. It can also close and clean up resources associated with the **RsslReactorChannel**.

FUNCTION NAME	DESCRIPTION
rsslReactorCloseChannel	Removes an RsslReactorChannel from the corresponding RsslReactor associated with the current ReactorChannel instance and cleans up associated resources. This additionally invokes the rsslCloseChannel function, as described in the Enterprise Transport API C Edition <i>Developers Guide</i> , to clean up any resources associated with the underlying RsslChannel . This function can be called from either outside or within a callback.

Table 36: rsslReactorCloseChannel Function

6.4.4.2 rsslReactorCloseChannel Example

```
RsslErrorInfo rsslErrorInfo;
/* Can be used inside or outside of a callback */
ret = rsslReactorCloseChannel(pReactor, pReactorChannel, &rsslErrorInfo);
```

Code Example 4: rsslReactorCloseChannel Example

6.5 OMM Consumer Multi-Credential Configuration

OMM Consumer **RsslReactorChannels** can have multiple credentials configured for each individually configured connections in it. This functionality works for connections configured with **RsslReactorConnectInfo** in both **RsslReactorConnectOptions.reactorConnectionList** and any connections configured for Warm Standby.

If **RsslReactorOmmConsumerRole.pOAuthCredentialList** and/or **RsslReactorOmmConsumerRole.pLoginRequestList** are configured, each connection attempt will use the specific configured OAuth credentials/login request messages to connect to the upstream provider.

This functionality only works with the OmmConsumer role.

The connections can be configured to use both Login messages and OAuth credentials. For more information about the Elektron Transport API's OAuth credential usage, refer to Chapter 7. For a usage example of this functionality, refer to the MultiConfigWLConsumer example.

6.5.1 Login Message Configuration and Callback

Individual login messages can be used in each channel connection. When connecting to an upstream provider, each channel will use the configured **RsslRDMLoginRequest** to login to the provider.

All login messages configured in **pLoginRequestList** must have the same SingleOpen and allow Suspect Data Verification flags and values on the **RsslRDMLoginRequest**.

6.5.1.1 RsslReactorLoginRequestMsgCredential Structure

Login messages in the **RsslReactorOMMConsumerRole.pLoginRequestList** are configured in the **RsslReactorLoginRequestMsgCredential** structure.

STRUCTURE MEMBER	DESCRIPTION
loginRequestMsg	Pointer to an RsslRDMLLoginRequest . All login messages configured in pLoginRequestList must have the same SingleOpen and allow Suspect Data Verification flags and values on the RsslRDMLLoginRequest .
pLoginRenewalEventCallback	Optional callback that will allow the application to supply new userName and authenticationExtended information that will be used with the new login. This callback MUST call rsslReactorSubmitLoginCredentialRenewal to provide the credentials. This callback will not be used if the channel that is using this login has Session Management on.
userSpecPtr	User specified pointer that will be passed to the user via RsslReactorLoginRenewalEvent.userSpecPtr in pLoginRenewalEventCallback .

Table 37: RsslReactorLoginRequestMsgCredential Structure Members

6.6 Reporting on Channel Statistics

You can use the **rsslReactorRetrieveChannelStatistic()** method to report on channel statistics. To use this method, you must first activate channel statistics reporting in **RsslReactorConnectOptions** by setting the **statisticFlags** member (for details on this member and the types of statistics on which you can report, refer to Section 6.4.1.2).

To get statistics, create an **RsslReactorChannelStatistic** structure and pass it in with the method. The Enterprise Transport API responds with the data for which the **statisticFlags** expressed interest.

6.7 Dispatching Data

Once an application has an **RsslReactor**, it can begin dispatching messages. Until there is at least one associated **RsslReactorChannel**, there is nothing to dispatch. When **RsslReactorChannels** are available for dispatching, each channel begins seeing its user-defined per-channel callbacks being invoked. For more information about available callbacks and their specifications, refer to Section 6.7.2.

An application can choose to dispatch across all associated **RsslReactorChannels** or on a particular **RsslReactorChannel**. If dispatching across multiple **RsslReactorChannels**, the **RsslReactor** attempts to fairly dispatch over all channels. In either case, the application can use the dispatch call to specify the maximum number of messages that will be processed and returned via callback.

Typically, an application registers both the **RsslReactor**'s **.eventFd** internal **RsslReactorChannel**'s **socketId** and each **RsslReactorChannel**'s **socketId** with an I/O notifier (e.g., **select**, **poll**). The I/O notifier can help inform the application when data is available on particular **RsslReactorChannels** or when channel information is available from the **RsslReactor**. An application can also forgo the use of notifiers and instead periodically call the dispatch function to process data as described in Section 6.7.1.

6.7.1 rsslReactorDispatch Function

NOTE: Applications should not call **rsslDestroyReactor** or **rsslReactorDispatch** from within a callback function. All other **RsslReactor** functionality is safe to use from within a callback.

Events received in callback functions should be assumed to be invalid when the callback function returns. For callbacks that provide **RsslMsg** or **RsslRDMMsg** structures, a deep copy of the object should be made if the application wishes to preserve it. To copy an **RsslMsg**, refer to the **rsslCopyMsg** function in the Enterprise Transport API C Edition *Developers Guide*; for copying an **RsslRDMMsg**, refer to the copy utility function for the appropriate **RsslRDMMsg** structure.

FUNCTION NAME	DESCRIPTION
<code>rsslReactorDispatch</code>	<p>This function processes events and messages across the provided RsslReactor and all of its associated RsslReactorChannels. When channel information or data is available for an RsslReactorChannel, the channel's user-defined callback function is invoked.</p> <p>The application can dispatch on a specified channel or over all channels associated with the RsslReactor. The application can also control the maximum number of messages dispatched with a single call to <code>rsslReactorDispatch</code>. This can be controlled through passed-in RsslReactorDispatchOptions, as described in Section 6.7.1.1.</p>

Table 38: `rsslReactorDispatch` Function

6.7.1.1 Reactor Dispatch Options

An application can use **RsslReactorDispatchOptions** to control various aspects of the call to `rsslReactorDispatch`.

STRUCTURE MEMBER	DESCRIPTION
<code>pReactorChannel</code>	The specific RsslReactorChannel to dispatch on in this call. If NULL , <code>rsslReactorDispatch</code> will process across all RsslReactorChannels associated with the passed in RsslReactor .
<code>maxMessages</code>	Controls the maximum number of events or messages processed in this call. If this is larger than the number of available messages, <code>rsslReactorDispatch</code> will return when there is no more data to process. This value is initialized to allow up to 100 messages to be returned with a single call to <code>rsslReactorDispatch</code> .

Table 39: `RsslReactorDispatchOptions` Structure Members

6.7.1.2 RsslReactorDispatchOptions Utility Function

The Enterprise Transport API provides the following utility function for use with **RsslReactorDispatchOptions**.

FUNCTION NAME	DESCRIPTION
<code>rsslClearReactorDispatchOptions</code>	Clears the RsslReactorDispatchOptions structure. Useful for structure reuse.

Table 40: RsslReactorDispatchOptions Utility Function

6.7.1.3 rsslReactorDispatch Example

```
RsslReactorDispatchOptions dispatchOpts;

/* Set dispatching options. */
rsslClearReactorDispatchOptions(&dispatchOpts);
dispatchOpts.maxMessages = 200;

/* Call rsslReactorDispatch(). It will keep dispatching events until there is nothing to read or
 * maxMessages is reached. */
ret = rsslReactorDispatch(pReactor, &dispatchOpts, &rsslErrorInfo);
```

Code Example 5: rsslReactorDispatch Example

6.7.2 Reactor Callback Functions

A series of callback functions returns (to the application) any state information about the **RsslReactorChannel** connection as well as messages for that channel. Each **RsslReactorChannel** can define its own unique callback functions or specify callback functions that can be shared across channels.

There are several values that can be returned from a callback function implementation. These can trigger specific **RsslReactor** behaviors based on the outcome of the callback function. Callback return values are as follows:

RETURN CODE	DESCRIPTION
<code>RSSL_RC_CRET_SUCCESS</code>	Indicates that the callback function was successful and the message or event has been handled.
<code>RSSL_RC_CRET_FAILURE</code>	Indicates that the message or event has failed to be handled. Returning this code from any callback function will cause the RsslReactor to shutdown.
<code>RSSL_RC_CRET_RAISE</code>	Can be returned from any domain-specific callback (e.g., RsslRDMLLoginMsgCallback). This will cause the RsslReactor to invoke the RsslDefaultMsgCallback for this message upon the domain-specific callbacks return.

Table 41: RsslReactorCallbackRet Callback Return Codes

6.7.3 Reactor Callback: Channel Event

The **RsslReactor** channel event callback communicates **RsslReactorChannel** and connection state information to the application. This callback function has the following prototype:

```
RsslReactorChannelEventCallback(RsslReactor*, RsslReactorChannel*, RsslReactorChannelEvent*)
```

When invoked, this returns the **RsslReactor** and the **RsslReactorChannel** on which the event occurred. In addition, an **RsslReactorChannelEvent** structure is returned, containing more information about the event.

6.7.3.1 Reactor Channel Event

The **RsslReactorChannelEvent** is returned to the application via the **RsslReactorChannelEventCallback**.

STRUCTUREMEMBER	DESCRIPTION
channelEventType	The type of event that has occurred on the RsslReactorChannel . For a list of enumeration values, refer to Section 6.7.3.2.
pReactorChannel	The RsslReactorChannel on which the event occurred.
pError	An RsslErrorInfo structure that is populated with error and warning information that occurred. This is only populated for RSSL_RC_CET_CHANNEL_DOWN and RSSL_RC_CET_WARNING event types.

Table 42: RsslReactorChannelEvent Structure Members

6.7.3.2 Reactor Channel Event Type Enumeration Values

FLAG ENUMERATION	MEANING
RSSL_RC_CET_CHANNEL_DOWN	Indicates that the RsslReactorChannel is not available for use. This could be a result of an initialization failure, a ping timeout, or some other kind of connection-related issue. RsslErrorInfo will contain more detailed information about what occurred. To clean up the failed RsslReactorChannel , the application should call rsslReactorCloseChannel .
RSSL_RC_CET_CHANNEL_DOWN_RECONNECTING	Indicates that the RsslReactorChannel is temporarily unavailable for use. The Reactor will attempt to reconnect the channel according to the values specified in RsslReactorConnectOptions when rsslReactorConnect was called. Before exiting the channelEventCallback , the application should release any resources associated with the channel, such as RsslBuffers , and remove its file-descriptor, if valid, from any notification sets.
RSSL_RC_CET_CHANNEL_OPEN	This event occurs only when the watchlist is enabled and only via the optional channelOpenCallback function. Indicates that a channel has been created via rsslReactorConnect . Though the channel is still not ready for dispatch, the application can begin submitting request messages, which are sent after the channel successfully initializes.

Table 43: RsslReactorChannelEventType Enumeration Values

FLAG ENUMERATION	MEANING
RSSL_RC_CET_CHANNEL_READY	Indicates that the RsslReactorChannel has successfully completed any necessary initialization processes. Where applicable, this includes exchanging any provided Login, Directory, or Dictionary content. The application should now be able to consume or provide content.
RSSL_RC_CET_CHANNEL_UP	Indicates that the RsslReactorChannel is successfully initialized and available for dispatching. Where applicable, any specified Login, Directory, or Dictionary messages are exchanged by the RsslReactor .
RSSL_RC_CET_FD_CHANGE	Indicates that a file-descriptor change occurred on the RsslReactorChannel . If the application is using its own I/O notification mechanism, it should replace the oldSocketId with socketId , both of which can be found on the RsslReactorChannel .
RSSL_RC_CET_INIT	Channel event initialization value. This should not be used by nor returned to the application.
RSSL_RC_CET_PREFERRED_HOST_COMPLETE	This event indicates that the Preferred Host operation has completed. If the preferred host operation has failed, RsslErrorInfo will contain additional information for logging purposes.
RSSL_RC_CET_WARNING	Indicates that the RsslReactorChannel has experienced an event that did not result in connection failure, but may require the attention of the application. RsslErrorInfo contains more detailed information about what occurred.

Table 43: RsslReactorChannelEventType Enumeration Values (Continued)

6.7.3.3 Reactor Channel Event Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearReactorChannelEvent	Clears an RsslReactorChannelEvent structure.

Table 44: RsslReactorChannelEvent Utility Functions

6.7.3.4 Reactor Channel Event Callback Example

```

RsslReactorCallbackRet channelEventCallback(RsslReactor *pReactor, RsslReactorChannel
    *pReactorChannel, RsslReactorChannelEvent *pChannelEvent)
{
    switch (pChannelEvent->channelEventType)
    {
        case RSSL_RC_CET_CHANNEL_UP:
            /* Channel has successfully initialized, add its descriptors to our notification
               mechanism. */
            FD_SET(pReactorChannel->socketId, &readFds);
            FD_SET(pReactorChannel->socketId, &exceptFds);
            break;
        case RSSL_RC_CET_CHANNEL_DOWN:
            /* Channel has failed. Clean up all references and close the channel. */
            FD_CLR(pReactorChannel->socketId, &readFds);
            FD_CLR(pReactorChannel->socketId, &exceptFds);
    }
}

```

```

    /* If all references are already clean up, channel can be closed now. Otherwise the
    * application can wait for a more appropriate time. */
    ret = rsslReactorCloseChannel(pReactor, pReactorChannel, &rsslErrorInfo);
break;

case RSSL_RC_CET_CHANNEL_READY:
    /* Channel has exchanged its initial messages(if any were provided on the role object)
    * and is ready for use. */
    sendItemRequests(pReactorChannel);
break;

case RSSL_RC_CET_FD_CHANGE:
    /* The descriptor representing this channel has changed. Normally the application only needs
    * to update its notification mechanism in response to this event. */
    FD_CLR(pReactorChannel->oldSocketId, &readFds);
    FD_CLR(pReactorChannel->oldSocketId, &exceptFds);
    FD_SET(pReactorChannel->socketId, &readFds);
    FD_SET(pReactorChannel->socketId, &exceptFds);
break;

case RSSL_RC_CET_WARNING:
    /* Received a warning about the channel. The channel is still active, but the event may
    * require the application's attention. */
    printf("Received channel warning event: %d(%s) ",
           pChannelEvent->pError->rsslError.rsslErrorId,
           pChannelEvent->pError->rsslError.text);
break;

}
return RSSL_RC_CRET_SUCCESS;
}

```

Code Example 6: Reactor Channel Event Callback Example

6.7.4 Reactor Callback: Default Message

The **RsslReactor** default message callback communicates all received content that is not handled directly by a domain-specific callback function. This callback is also invoked after any domain-specific callback that returns the **RSSL_RC_CET_RAISE** value. This callback function has the following prototype:

```
RsslDefaultMsgCallback(RsslReactor*, RsslReactorChannel*, RsslMsgEvent*)
```

When invoked, this returns the **RsslReactor** and the **RsslReactorChannel** on which the event occurred. In addition, an **RsslMsgEvent** structure is returned, containing more information about the event information.

6.7.4.1 Reactor Message Event

The **RsslMsgEvent** is returned to the application via the **RsslDefaultMsgCallback**.

STRUCTURE MEMBER	DESCRIPTION
pRsslMsgBuffer	An RsslBuffer containing the raw, undecoded message that was read and processed by the callback. NOTE: When the consumer watchlist is enabled, an RsslBuffer is not provided, because the message might not match this buffer, or the message might be internally generated.
pRsslMsg	An RsslMsg structure populated with message content by calling the rsslDecodeMsg . If not present, an error was encountered while processing the information. NOTE: When the consumer watchlist is enabled, pRsslMsg is not provided to callback functions that provide RDM messages.
pError	An RsslErrorInfo structure that is populated with error and warning information that occurred, likely related to message decoding or processing.
pStreamInfo	Any information associated with a stream (only when the watchlist is enabled).
pSeqNum	The sequence number associated with a message, if present (only when using multicast).
pFTGroupId	The fault-tolerant group associated with a message, if present (only when using multicast).

Table 45: RsslMsgEvent Structure Members

6.7.4.2 Reactor Message Event Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearMsgEvent	Clears an RsslMsgEvent structure.

Table 46: RsslMsgEvent Utility Function

6.7.4.3 Reactor Message Event Callback Example

```
RsslReactorCallbackRet defaultMsgCallback(RsslReactor *pReactor, RsslReactorChannel *pReactorChannel,
    RsslMsgEvent *pMsgEvent)
{
    RsslMsg *pRsslMsg = pMsgEvent->pRsslMsg;

    /* Received an RsslMsg --- or, if the decode failed, an error. */
    /* The RsslMsg will have already been passed through rsslDecodeMsg. Only the payload requires
       additional decoding */
    if (pRsslMsg)
        processRsslMsg(pRsslMsg);
    else
        printf("Error: %s(%s)\n", pMsgEvent->pErrorInfo->rsslError.text,
            pMsgEvent->pErrorInfo->errorLocation);
}
```

Code Example 7: Reactor Message Event Callback Example

6.7.5 Reactor Callback: RDM Login Message

The **RsslReactor** RDM Login Message callback is used to communicate all received RDM Login messages. This callback function has the following prototype:

```
RsslRDMLLoginMsgCallback(RsslReactor*, RsslReactorChannel*, RsslRDMLLoginMsgEvent*)
```

When invoked, this will return the **RsslReactor** and the **RsslReactorChannel** on which the event occurred. In addition, an **RsslRDMLLoginMsgEvent** structure is returned, containing more information about the event information.

6.7.5.1 Reactor RDM Login Message Event

The **RsslRDMLLoginMsgEvent** is returned to the application via the **RsslRDMLLoginMsgCallback**.

STRUCTURE MEMBER	DESCRIPTION
baseMsgEvent	An RsslMsgEvent populated with the raw buffer (RsslMsg) and any error information. This structure is defined in Section 6.7.4.1.
pRDMLLoginMsg	The RDM representation of the decoded Login message. If not present, an error was encountered while processing the information. This message is presented as the RsslRDMLLoginMsg , described in Section 8.3.

Table 47: RsslRDMLLoginMsgEvent Structure Members

6.7.5.2 Reactor RDM Login Message Event Utility Function

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMLLoginMsgEvent</code>	Clears an RsslRDMLLoginMsgEvent structure.

Table 48: RsslRDMLLoginMsgEvent Utility Function

6.7.5.3 Reactor RDM Login Message Event Callback Example

```

RsslReactorCallbackRet loginMsgCallback(RsslReactor *pReactor, RsslReactorChannel *pReactorChannel,
    RsslRDMLLoginMsgEvent *pLoginMsgEvent)
{
    RsslRDMLLoginMsg *pLoginMsg = pLoginMsgEvent->pRDMLLoginMsg;

    /* Received an RsslRDMLLoginMsg --- or, if the decode failed, an error. */
    /* The login message will already be fully decoded */
    if (pLoginMsg)
    {
        switch(pLoginMsg->rdmMsgBase.rdmMsgType)
        {
            case RDM_LG_MT_REFRESH:
                RsslRDMLLoginRefresh *pRefresh = &pLoginMsg->refresh;
                break;
            case RDM_LG_MT_STATUS:
                RsslRDMLLoginStatus *pStatus = &pLoginMsg->status;
                break;
            default:
                printf("Received unhandled login message.\n"); break;
        }
    }
    else
        printf("Error: %s(%s)\n", pLoginMsgEvent->baseMsgEvent.pErrorInfo->rsslError.text,
            pLoginMsgEvent->baseMsgEvent.pErrorInfo->errorLocation);
}

```

Code Example 8: Reactor RDM Login Message Event Callback Example

6.7.6 Reactor Callback: RDM Directory Message

The **RsslReactor** RDM Directory Message callback is used to communicate all received RDM Directory messages. This callback function has the following prototype:

```
RsslRDMDirectoryMsgCallback(RsslReactor*, RsslReactorChannel*, RsslRDMDirectoryMsgEvent*)
```

When invoked, this will return the **RsslReactor** and the **RsslReactorChannel** on which the event occurred. In addition, an **RsslRDMDirectoryMsgEvent** structure is returned, containing more information about the event information.

6.7.6.1 Reactor RDM Directory Message Event

The **RsslRDMDirectoryMsgEvent** is returned to the application via the **RsslRDMDirectoryMsgCallback**.

STRUCTURE MEMBER	DESCRIPTION
baseMsgEvent	An RsslMsgEvent populated with the raw buffer (RsslMsg) and any error information. This structure is defined in Section 6.7.4.1.
pRDMDirectoryMsg	The RDM representation of the decoded Source Directory message. If not present, an error was encountered while processing the information. This message is presented as the RsslRDMDirectoryMsg , described in Section 8.4.

Table 49: RsslRDMDirectoryMsgEvent Structure Members

6.7.6.2 Reactor RDM Directory Message Event Utility Function

FUNCTION NAME	DESCRIPTION
rsslClearRDMDirectoryMsgEvent	Clears an RsslRDMDirectoryMsgEvent structure.

Table 50: RsslRDMDirectoryMsgEvent Utility Function

6.7.6.3 Reactor RDM Directory Message Event Callback Example

```
RsslReactorCallbackRet directoryMsgCallback(RsslReactor *pReactor, RsslReactorChannel
    *pReactorChannel, RsslRDMDirectoryMsgEvent *pDirectoryMsgEvent)
{
    RsslRDMDirectoryMsg *pDirectoryMsg = pDirectoryMsgEvent->pRDMDirectoryMsg;

    /* Received an RsslRDMDirectoryMsg --- or, if the decode failed, an error. */
    /* The directory message will already be fully decoded */
    if (pDirectoryMsg)
    {
        switch(pDirectoryMsg->rdmMsgBase.rdmMsgType)
        {
            case RDM_DR_MT_REFRESH:
                RsslRDMDirectoryRefresh *pRefresh = &pDirectoryMsg->refresh;
```

```

        break;
    case RDM_DR_MT_UPDATE:
        RsslRDMDirectoryUpdate *pUpdate = &pDirectoryMsg->update;
        break;
    case RDM_DR_MT_STATUS:
        RsslRDMDirectoryStatus *pStatus = &pDirectoryMsg->status;
        break;
    default:
        printf("Received unhandled directory message.\n");
    }
}
else
    printf("Error: %s(%s)\n", pDirectoryMsgEvent->baseMsgEvent.pErrorInfo->rsslError.text,
        pDirectoryMsgEvent->baseMsgEvent.pErrorInfo->errorLocation);
}

```

Code Example 9: Reactor RDM Directory Message Event Callback Example

6.7.7 Reactor Callback: RDM Dictionary Message

The **RsslReactor** RDM Dictionary Message callback is used to communicate all received RDM Dictionary messages. This callback function has the following prototype:

```
RsslRDMDictionaryMsgCallback(RsslReactor*, RsslReactorChannel*, RsslRDMDictionaryMsgEvent*)
```

When invoked, this will return the **RsslReactor** and the **RsslReactorChannel** on which the event occurred. In addition, an **RsslRDMDictionaryMsgEvent** structure is returned, containing more information about the event information.

6.7.7.1 Reactor RDM Dictionary Message Event

The **RsslRDMDictionaryMsgEvent** is returned to the application via the **RsslRDMDictionaryMsgCallback**.

STRUCTURE MEMBER	DESCRIPTION
baseMsgEvent	An RsslMsgEvent populated with the raw buffer (RsslMsg) and any error information. This structure is defined in Section 6.7.4.1.
pRDMDictionaryMsg	The RDM representation of the decoded Dictionary message. If not present, an error was encountered while processing the information. This message is presented as the RsslRDMDictionaryMsg , described in Section 8.5.

Table 51: RsslRDMDictionaryMsgEvent Structure Members

6.7.7.2 Reactor RDM Dictionary Message Event Utility Function

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMDictionaryMsgEvent</code>	Clears an RsslRDMDictionaryMsgEvent structure.

Table 52: RsslRDMDictionaryMsgEvent Utility Function

6.7.7.3 Reactor RDM Dictionary Message Event Callback Example

```
RsslReactorCallbackRet dictionaryMsgCallback(RsslReactor *pReactor, RsslReactorChannel
    *pReactorChannel, RsslRDMDictionaryMsgEvent *pDictionaryMsgEvent)
{
    RsslRDMDictionaryMsg *pDictionaryMsg = pDictionaryMsgEvent->pRDMDictionaryMsg;

    /* Received an RsslRDMDictionaryMsg --- or, if the decode failed, an error. */
    if (pDictionaryMsg)
    {
        switch(pDictionaryMsg->rdmMsgBase.rdmMsgType)
        {
            case RDM_DC_MT_REFRESH:
                RsslRDMDictionaryRefresh *pRefresh = &pDictionaryMsg->refresh;
                break;
            case RDM_DC_MT_STATUS:
                RsslRDMDictionaryStatus *pStatus = &pDictionaryMsg->status;
                break;
            default:
                printf("Received unhandled dictionary message.\n");
        }
    }
    else
        printf("Error: %s(%s)\n", pDictionaryMsgEvent->baseMsgEvent.pErrorInfo->rsslError.text,
            pDictionaryMsgEvent->baseMsgEvent.pErrorInfo->errorLocation);
}
```

Code Example 10: Reactor RDM Dictionary Message Event Callback Example

6.8 Writing Data

The Enterprise Transport API Reactor helps streamline the high performance writing of content. The **RsslReactor** flushes content to the network so the application does not need to. The **RsslReactor** does so through the use of a separate worker thread that becomes active whenever there is queued content that needs to be passed to the connection.

The Enterprise Transport API Reactor functionality offers two methods for writing content: **rsslReactorSubmitMsg** and **rsslReactorSubmit**. When writing applications to the Reactor, consider which is most appropriate for your needs:

rsslReactorSubmitMsg

- Takes an **RsslMsg** structure as part of its options; does not require retrieval of an **RsslBuffer** from the channel.
- Must be used when the consumer watchlist is enabled.

rsslReactorSubmit

- Takes an **RsslBuffer** which the application retrieves from the channel.
- More efficient: the application encodes directly into the buffer, and can use buffer packing.
- Cannot be used when the consumer watchlist is enabled.

6.8.1 Writing Data using rsslReactorSubmitMsg()

rsslReactorSubmitMsgMsg provides a simple interface for writing **RsslMsgs**. To send a message, the application populates an **RsslMsg** structure, sets it (along with any other desired options) on an **RsslReactorSubmitMsgOptions** structure, and calls **rsslReactorSubmitMsg** with the structure.

A buffer is not needed to use **rsslReactorSubmitMsg**. If the application needs to include any encoded content, it can encode the content into any available memory, and set the appropriate member of the **RsslMsg** to point to the memory (as well as set the length of the encoded content).

6.8.1.1 rsslReactorSubmitMsg Function

FUNCTION NAME	DESCRIPTION
rsslReactorSubmitMsgMsg	Encodes and submits an RsslMsg to the Reactor. This function expects a properly populated RsslMsg . This function allows for several modifications and additional parameters to be specified via the RsslReactorSubmitMsgOptions structure.

Table 53: rsslReactorSubmitMsg Function

6.8.1.2 Reactor Submit Message Options

An application can use **RsslReactorSubmitMsgOptions** to control various aspects of the call to **rsslReactorSubmitMsgMsg**.

STRUCTURE MEMBER	DESCRIPTION
majorVersion	The RWF major version of any encoded content in the message.
minorVersion	The RWF minor version of any encoded content in the message.
pRsslMsg	The RsslMsg structure to submit. Use only one instance of either pRsslMsg or pRDMMsg.
pRDMMsg	The RsslRDMMsg structure to submit. Use only one instance of either pRsslMsg or pRDMMsg.

Table 54: RsslReactorSubmitMsgOptions Structure Members

STRUCTURE MEMBER	DESCRIPTION
pServiceName	<p>The application can use this instead of the serviceId member specified on the RsslMsgKey of an RsslMsg.</p> <p>When used to open streams via request messages, the RsslReactor will recover using this service name.</p> <p>When used for other message types such as RsslPostMsg or RsslGenericMsg, the RsslReactor converts the name to its corresponding ID before writing the message.</p> <p>NOTE: This option is supported only when the consumer watchlist is enabled.</p>
requestMsgOptions	Provides additional functionality that may be used when using request messages to send requests.

Table 54: **RsslReactorSubmitMsgOptions** Structure Members (Continued)

6.8.1.3 RsslReactorRequestMsgOptions

RsslReactorRequestMsgOptions provide additional functionality when requesting items. These options are available only when the watchlist is enabled.

STRUCTURE MEMBER	DESCRIPTION
pUserSpec	A user-specified pointer that will be associated with the stream. This pointer will be provided in responses to this stream via the RsslStreamInfo provided with each message event.

Table 55: **RsslReactorRequestMsgOptions** Structure Members

6.8.1.4 RsslReactorSubmitMsgOptions Utility Function

The Enterprise Transport API provides the following utility function for use with **RsslReactorSubmitMsgOptions**.

FUNCTION NAME	DESCRIPTION
rsslClearReactorSubmitMsgOptions	Clears the RsslReactorSubmitMsgOptions structure. Useful for structure reuse.

Table 56: **RsslReactorSubmitMsgOptions** Utility Function

6.8.1.5 rsslReactorSubmitMsg Return Codes

The following table defines the return codes that can occur when using **rsslReactorSubmitMsg**.

RETURN CODE	DESCRIPTION
RSSL_RET_SUCCESS	Indicates that the rsslReactorSubmitMsg function has succeeded.
RSSL_RET_BUFFER_NO_BUFFERS	Indicates that not enough pool buffers are available to write the message. The application can try to submit the message later, or it can use rsslReactorChannelIoctl to increase the number of available pool buffers and try again.
RSSL_RET_FAILURE	Indicates that a general failure has occurred and the message was not submitted. The RsslErrorInfo structure passed to the function will contain more details.

Table 57: **rsslReactorSubmitMsg** Return Codes

6.8.1.6 rsslReactorSubmitMsg Example

The following example shows typical use of `rsslReactorSubmitMsg`.

```
RsslMsg requestMsg;
RsslReactorSubmitMsgOptions opts;
RsslErrorInfo errorInfo;
RsslRet ret;

rsslClearRequestMsg(&requestMsg);
requestMsg.msgBase.streamId = 2;
requestMsg.msgBase.domainType = RSSL_DMT_MARKET_PRICE;
requestMsg.msgBase.containerType = RSSL_DT_NO_DATA;
requestMsg.flags = RSSL_RQMF_STREAMING | RSSL_RQMF_HAS_QOS;
requestMsg.qos.timeliness = RSSL_QOS_TIME_REALTIME;
requestMsg.qos.rate = RSSL_QOS_RATE_TICK_BY_TICK;
requestMsg.msgBase.msgKey.flags = RSSL_MKF_HAS_NAME | RSSL_MKF_HAS_SERVICE_ID;
requestMsg.msgBase.msgKey.name.data = "TRI.N";
requestMsg.msgBase.msgKey.name.length = 5;
requestMsg.msgBase.msgKey.serviceId = 1;

rsslClearReactorSubmitMsgOptions(&opts);
opts.pRsslMsg = (RsslMsg*)&requestMsg;
ret = rsslReactorSubmitMsg(pReactor, pReactorChannel, &opts, &errorInfo);
```

Code Example 11: rsslReactorSubmitMsg Example

6.8.2 Writing data using `rsslReactorSubmit()`

The `rsslReactorSubmit` function offers efficient writing of data by using buffers retrieved directly from the Enterprise Transport API transport buffer pool. It also provides additional features not normally available from `rsslReactorSubmitMsgMsg`, such as buffer packing or the Enterprise Transport API priority queue. When ready to send data, the application acquires a buffer from the Enterprise Transport API pool. This allows the content to be encoded directly into the output buffer, reducing the number of times the content needs to be copied. Once content is encoded and the buffer is properly populated, the application can submit the data to the reactor. The Enterprise Transport API will ensure that successfully submitted buffers reach the network. Applications can also pack multiple messages into a single buffer by following a similar process as described above, however instead of getting a new buffer for each message the application uses the reactor's pack function instead. The following flow chart depicts the typical write process.

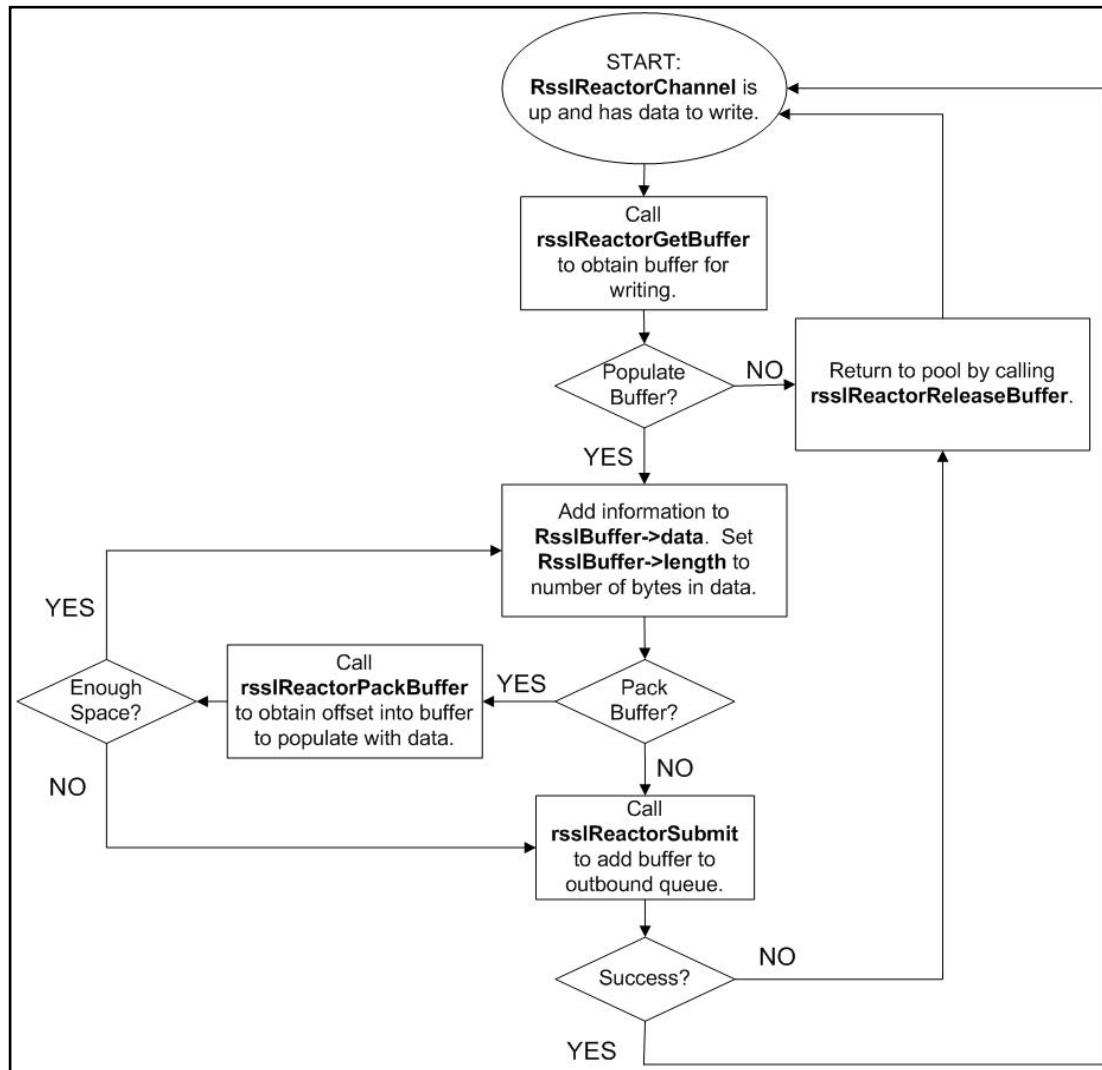


Figure 7. Flow Chart for writing data using `rsslReactorSubmit`

6.8.2.1 Obtaining a Buffer: Overview

Before you can submit information, you must obtain a buffer from the internal Enterprise Transport API buffer pool, as described in the Enterprise Transport API C Edition *Developers Guide*. After acquiring the buffer, you can populate the **RsslBuffer.data** and set the **RsslBuffer.length** to the number of bytes referred to by **data**. If the buffer is not used or the **rsslReactorSubmit** function call fails, the buffer must be released back into the pool to ensure proper reuse and cleanup. If the buffer is successfully passed to **rsslReactorSubmit**, the reactor will return the buffer to the pool.

The number of buffers made available to an **RsslReactorChannel** is configurable through the **RsslReactorConnectOptions** or **RsslReactorAcceptOptions**. For more information about available **rsslReactorConnect** and **rsslReactorAccept** options, refer to Section 6.4.1.2 and Section 6.4.1.8.

6.8.2.2 Obtaining a Buffer: Buffer Management Functions

FUNCTION NAME	DESCRIPTION
rsslReactorGetBuffer	<p>Obtains a buffer of the requested size from the buffer pool. When the RsslBuffer is returned, the length member indicates the number of bytes available in the buffer (which should match the amount the application requested). When populating, length must be set to the number of bytes actually used. This ensures that only the required bytes are written to the network.</p> <p>If the requested size is larger than the maxFragmentSize, the transport will create and return the buffer to the user. When written, this buffer will be fragmented by the rsslReactorSubmitMsg function (for further details, refer to Section 6.8.2.4).</p> <p>Because of some additional book keeping required when packing, the application must specify whether a buffer should be 'packable' when calling rsslReactorGetBuffer. For more information on packing, refer to Section 6.8.2.8.</p> <p>For performance purposes, an application is not permitted to request a buffer larger than maxFragmentSize and have the buffer be 'packable.'</p> <p>If the buffer is not used or the rsslReactorSubmitMsg call fails, the buffer must be returned to the pool using rsslReactorReleaseBuffer. If the rsslReactorSubmitMsg call is successful, the buffer will be returned to the correct pool by the transport.</p> <p>This function calls the rsslGetBuffer function which has its use and return values described in the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>
rsslReactorReleaseBuffer	<p>Releases a buffer back to the correct pool. This should only be called with buffers that originate from rsslReactorGetBuffer and are not successfully passed to rsslReactorSubmitMsg.</p> <p>This function calls the Enterprise Transport API rsslReactorReleaseBuffer function which has its use and return values described in the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>
rsslReactorChannelBufferUsage	<p>Returns the number of buffers currently in use by the RsslReactorChannel, this includes buffers that the application holds and buffers internally queued and waiting to be flushed to the connection by the RsslReactor.</p> <p>This function calls the rsslBufferUsage function which has its use and return values described in the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>

Table 58: Reactor Buffer Management Functions

6.8.2.3 Obtaining a Buffer: `rsslReactorGetBuffer` Return Values

The following table defines return and error code values that can occur while using `rsslReactorGetBuffer`.

RETURN CODE	DESCRIPTION
Valid buffer returned Success Case	An RsslBuffer is returned to the user. The RsslBuffer.length indicates the number of bytes available to populate and the RsslBuffer.data provides a starting location for population.
NULL buffer returned Error Code: <code>RSSL_RET_BUFFER_NO_BUFFERS</code>	NULL is returned to the user. This value indicates that there are no buffers available to the user. See RsslErrorInfo content for more details. This typically occurs because all available buffers are queued and pending flushing to the connection. The <code>rsslReactorChannelIoctl</code> function can be used to increase the number of guaranteedOutputBuffers (for details, refer to Section 6.12).
NULL buffer returned Error Code: <code>RSSL_RET_FAILURE</code>	NULL is returned to the user. This value indicates that some type of general failure has occurred. The RsslReactorChannel should be closed.
NULL buffer returned Error Code: <code>RSSL_RET_INIT_NOT_INITIALIZED</code>	Indicates that the underlying RSSL Transport has not been initialized. See the RsslErrorInfo content for more details.

Table 59: `rsslReactorGetBuffer` Return Values

6.8.2.4 Writing Data: Overview

After an **RsslBuffer** is obtained from `rsslReactorGetBuffer` and populated with the user's data, the buffer can be passed to the `rsslReactorSubmitMsg` function. This function manages queuing and flushing of user content. It will also perform any fragmentation or compression. If an unrecoverable error occurs, any **RsslBuffer** that has not been successfully passed to `rsslReactorSubmitMsg` should be released to the pool using `rsslReactorReleaseBuffer`. Section 6.8.2.5 describes the `rsslReactorSubmitMsg` function and its associated parameters.

6.8.2.5 Writing Data: `rsslReactorSubmit` Function

NOTE: Before passing a buffer to `rsslReactorSubmit`, it is required that the application set **length** to the number of bytes actually used. This ensures that only the required bytes are written to the network.

FUNCTION NAME	DESCRIPTION
<code>rsslReactorSubmit</code>	Writes data. This function expects the buffer to be properly populated, where length reflects the actual number of bytes used. This function calls the Enterprise Transport API rsslWrite function and also triggers the rsslFlush function (described in the Enterprise Transport API C Edition <i>Developers Guide</i>). This function allows for several modifications and additional parameters to be specified via the RsslReactorSubmitOptions structure, defined in Section 6.8.2.6. For a list of return codes, refer to Section 6.8.2.7.

Table 60: `rsslReactorSubmitMsg` Function

6.8.2.6 Writing Data: Reactor Submit Options

The application uses **RsslReactorSubmitOptions** to control various aspects of the call to **rsslReactorSubmit**.

STRUCTURE MEMBER	DESCRIPTION
priority	Controls the priority at which the data will be written. Valid priorities are <ul style="list-style-type: none"> RSSL_HIGH_PRIORITY RSSL_MEDIUM_PRIORITY RSSL_LOW_PRIORITY More information about write priorities, including an example scenario, are available in the <i>Transport API C Edition Developers Guide</i> .
writeFlags	Flag values that allow the application to modify the behavior of this rsslReactorSubmit call. This includes options to bypass queuing or compression. More information about the specific flag values are available in the <i>Transport API C Edition Developers Guide</i> .
pBytesWritten	If specified, will return the number of bytes to be written, including any transport header overhead and taking into account any savings from compression.
pUncompressedBytesWritten	If specified, will return the number of bytes to be written, including any transport header overhead but not taking into account any compression savings.

Table 61: RsslReactorSubmitOptions Structure Members

6.8.2.7 Writing Data: rsslReactorSubmit Return Codes

The following table defines the return codes that can occur when using **rsslReactorSubmitMsg**.

RETURN CODE	DESCRIPTION
RSSL_RET_SUCCESS	Indicates that the rsslReactorSubmitMsg function has succeeded. The RsslBuffer will be released by the Enterprise Transport API Reactor.
RSSL_RET_WRITE_CALL_AGAIN	Indicates that a large buffer could not be fully written with this rsslReactorSubmitMsg call. This is typically due to all pool buffers being unavailable. The rsslReactorSubmitMsg will flush for the user to free up buffers. The application can optionally use rsslReactorChannelIoctl to increase the number of available pool buffers. After pool buffers become available again, the same buffer should be used to call rsslReactorSubmitMsg an additional time (using the same priority level for proper ordering of each fragment). This will continue the fragmentation process from where it left off. <p>If the application does not subsequently pass the buffer to rsslReactorSubmitMsg, the application should release it by calling rsslReactorReleaseBuffer.</p>
RSSL_RET_FAILURE	Indicates that a general write failure has occurred. The RsslReactorChannel should be closed. <p>The application should release the RsslBuffer by calling rsslReactorReleaseBuffer.</p>

Table 62: rsslReactorSubmitMsg Return Codes

6.8.2.8 Writing Data: RsslReactorSubmitOptions Utility Function

The Enterprise Transport API provides the following utility function for use with the **RsslReactorDispatchOptions**.

FUNCTION NAME	DESCRIPTION
<code>rsslClearReactorSubmitOptions</code>	Clears the RsslReactorSubmitOptions structure. Useful for structure reuse.

Table 63: RsslReactorSubmitOptions Utility Function

6.8.2.9 Example: rsslReactorGetBuffer and rsslReactorSubmit Example

The following example shows typical use of **rsslReactorGetBuffer** and **rsslReactorSubmit**.

```
RsslBuffer *pMsgBuffer;
RsslEncodeIterator encodeIter;
RsslReactorSubmitOptions submitOpts;

pMsgBuffer = rsslReactorGetBuffer(pReactorChannel, 1024, RSSL_FALSE, &rsslErrorInfo);

rsslClearEncodeIterator(&encodeIter);
rsslSetEncodeIteratorRWFVersion(&encodeIter, pReactorChannel->majorVersion, pReactorChannel-
    >minorVersion);
rsslSetEncodeIteratorBuffer(&encodeIter, pMsgBuffer);
encodeMsgIntoBuffer(&encodeIter, pMsgBuffer);

pMsgBuffer->length = rsslGetEncodedBufferLength(&encodeIter);
rsslClearReactorSubmitOptions(&submitOpts);
ret = rsslReactorSubmit(pReactor, pReactorChannel, pMsgBuffer, &submitOpts, &rsslErrorInfo);
/* check return code */
switch (ret)
{
    case RSSL_RET_SUCCESS:
        /* successful write, nothing left to do */
        return 0;
    break;
    case RSSL_RET_FAILURE:
        /* an error occurred, need to release buffer */
        rsslReactorReleaseBuffer(pReactorChannel, pMsgBuffer, &rsslErrorInfo);
    break;
    case RSSL_RET_WRITE_CALL_AGAIN:
        /* large message couldn't be fully written with one call, pass it to submit again */
        ret = rsslReactorSubmit(pReactor, pReactorChannel, pMsgBuffer, &rsslErrorInfo);
    break;
}
```

Code Example 12: Writing Data Using rsslReactorSubmit, rsslReactorGetBuffer, and rsslReactorReleaseBuffer

6.8.2.10 Packing Additional Data into a Buffer

If an application is writing many small buffers, it may be advantageous to combine the small buffers into one larger buffer. This can increase efficiency of the transport layer by reducing the overhead associated with each write operation, although it may add to the latency associated with each smaller buffer.

It is up to the writing application to determine when to stop packing, and the mechanism used can vary greatly. A simple algorithm can pack a fixed number of messages each time. A slightly more complex technique could use the returned **RsslBuffer.length** to determine the amount of space remaining and pack until the buffer is nearly full. Both of these mechanisms can introduce a variable amount of latency as they both depend on the rate of arrival of data (e.g., the packed buffer will not be written until enough data arrives to fill it). One way of balancing this is to employ a timer, used to limit the amount of time a packed buffer is held. If the buffer is full prior to the timer expiring, the data is written. However, when the timer expires the buffer will be written regardless of the amount of data it contains. This can help limit latency by specifying a limit to the time data is held (via use of the timer).

FUNCTION NAME	DESCRIPTION
<code>rsslReactorPackBuffer</code>	<p>Packs the contents of a passed-in RsslBuffer and returns a new RsslBuffer to continue packing new data into. For a buffer to allow packing, it must be requested from rsslReactorGetBuffer as 'packable' and cannot exceed the maxFragmentSize. The returned buffer provides a data pointer for populating and the length conveys number of bytes available in the buffer.</p> <p>An application can use the RsslBuffer.length to determine the amount of space available to continue packing buffers into. After each buffer is populated, the length should be set to reflect the actual number of bytes contained in the buffer. This will ensure that only the necessary space is reserved while packing.</p> <p>rsslReactorPackBuffer return values are defined in Section 6.8.2.11.</p> <p>This function calls the rsslPackBuffer function as described in the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>

Table 64: `rsslReactorPackBuffer` Function

6.8.2.11 `rsslReactorPackBuffer` Return Values

The following table defines return and error code values that can occur when using **rsslReactorPackBuffer**.

RETURN CODE	DESCRIPTION
Valid buffer returned Success Case	An RsslBuffer is returned to the user. The RsslBuffer.length indicates the number of bytes available to populate and the RsslBuffer.data provides a starting location for population.
NULL buffer returned Error Code: <code>RSSL_RET_FAILURE</code>	NULL is returned to the user. This value indicates that some type of general failure has occurred. The RsslReactorChannel should be closed.
NULL buffer returned Error Code: <code>RSSL_RET_INIT_NOT_INITIALIZED</code>	Indicates that the underlying RSSL Transport has not been initialized. See the RsslErrorInfo content for more details.

Table 65: `rsslReactorPackBuffer` Return Values

6.8.2.12 Example: `rsslReactorGetBuffer`, `rsslReactorPackBuffer`, and `rsslReactorSubmit`

The following example shows typical use of `rsslReactorGetBuffer`, `rsslReactorPackBuffer`, and `rsslReactorSubmit`.

```
RsslBuffer *pMsgBuffer;
RsslReactorSubmitOptions submitOpts;
RsslEncodeIterator encodeIter;

/* get a packable buffer */
pMsgBuffer = rsslReactorGetBuffer(pReactorChannel, 1024, RSSL_TRUE, &rsslErrorInfo);

rsslClearEncodeIterator(&encodeIter);
rsslSetEncodeIteratorRWFVersion(&encodeIter, pReactorChannel->majorVersion, pReactorChannel-
    >minorVersion);
rsslSetEncodeIteratorBuffer(&encodeIter, pMsgBuffer);
encodeMsgIntoBuffer(&encodeIter, pMsgBuffer);

/* pack first encoded message into buffer */
pMsgBuffer->length = rsslGetEncodedBufferLength(&encodeIter);
pMsgBuffer = rsslReactorPackBuffer(pReactorChannel, pMsgBuffer, &rsslErrorInfo);

rsslClearEncodeIterator(&encodeIter);
rsslSetEncodeIteratorRWFVersion(&encodeIter, pReactorChannel->majorVersion, pReactorChannel-
    >minorVersion);
rsslSetEncodeIteratorBuffer(&encodeIter, pMsgBuffer);
encodeMsgIntoBuffer(&encodeIter, pMsgBuffer);

/* pack second encoded message into buffer */
pMsgBuffer->length = rsslGetEncodedBufferLength(&encodeIter);
pMsgBuffer = rsslReactorPackBuffer(pReactorChannel, pMsgBuffer, &rsslErrorInfo);

rsslClearEncodeIterator(&encodeIter);
rsslSetEncodeIteratorRWFVersion(&encodeIter, pReactorChannel->majorVersion, pReactorChannel-
    >minorVersion);
rsslSetEncodeIteratorBuffer(&encodeIter, pMsgBuffer);

/* now write packed buffer by passing third buffer to rsslSubmit */
encodeMsgIntoBuffer(&encodeIter, pMsgBuffer);
pMsgBuffer->length = rsslGetEncodedBufferLength(&encodeIter);

rsslClearReactorSubmitOptions(&submitOpts);
ret = rsslReactorSubmit(pReactor, pReactorChannel, pMsgBuffer, &submitOpts, &rsslErrorInfo);
```

Code Example 13: Message Packing using `rsslReactorPackBuffer`

6.9 Creating and Using Tunnel Streams

The Reactor allows users to create and use special tunnel streams. A tunnel stream is a private stream with additional behaviors, such as end-to-end line of sight for authentication and guaranteed delivery. Tunnel streams are founded on the private streams concept, and the Enterprise Transport API establishes them between consumer and provider endpoints (passing through any intermediate components, such as LSEG Real-Time Distribution System or a LSEG Real-Time Edge Device).

When creating a tunnel, the consumer indicates any additional behaviors to enforce, which is exchanged with the provider application end point. The provider end-point acknowledges creation of the stream as well as the behaviors that it will enforce on the stream. After the stream is established, the consumer can exchange any content it wants, though the tunnel stream will enforce behaviors on the transmitted content as negotiated with the provider.

A tunnel stream allows for multiple substreams to exist, where substreams follow from the same general stream concept, except that they flow and coexist within the confines of a tunnel stream.

In the following diagram, the orange cylinder represents a tunnel stream that connects the consumer application to the provider application. Notice that the tunnel stream passes directly through intermediate components: the tunnel stream has end-to-end line of sight so that the provider and consumer effectively talk to one another directly, though they traverse multiple devices in the system. Each black line flowing through the cylinder represents a different substream, where each substream transmits its own independent stream of information. Each substream could communicate different market content; for example one could be a Time Series request while another could be a request for Market Price content.

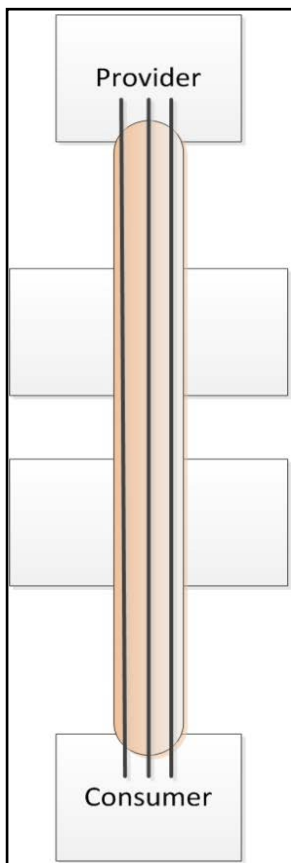


Figure 8. Tunnel Stream Illustration

6.9.1 Authenticating a Tunnel Stream

Providers might require the consumer to authenticate itself when establishing the tunnel stream. The type of authentication, if any, is given by the **RsslClassOfService.authentication.type**. For more information on class or service, refer to Section 6.9.3.

The **RsslClassOfService.authentication.type** may be set to **RDM_COS_AU_OMM_LOGIN**. When an OMM consumer expects this type of authentication, it should set an **RsslRDMLoginRequest** message on the **RsslTunnelStreamOpenOptions.pAuthLoginRequest** member. If the OMM consumer application does not provide it, the API will use the login request provided on the **RsslReactorOMMConsumerRole.pLoginRequest** when the consumer connected (refer to Section 6.3.2). The consumer must provide one of these for authentication of this type.

The login request will be sent to the provider. When the provider sends a Login response to complete the authentication, the **RsslTunnelStreamStatusEvent** event given to the consumer will include an **RsslTunnelStreamAuthInfo** structure with more details. OMM provider applications will see the login request as a normal message within the **RsslTunnelStream** and should respond with a login response message via **rsslTunnelStreamSubmit** or **rsslTunnelStreamSubmitMsg**.

Other types of authentication might be specified, but must be performed by both the provider and consumer applications by submitting normal **RsslTunnelStream** messages via **rsslTunnelStreamSubmit** or **rsslTunnelStreamSubmitMsg**.

The **RsslTunnelStreamAuthInfo** structure contains the following member:

MEMBER	DESCRIPTION
pLoginMsg	The Login message sent by the tunnel stream's provider application, which resulted in this event.

Table 66: RsslTunnelStreamAuthInfo Structure Members

6.9.2 Opening a Tunnel Stream

The user can create one or more tunnel streams and associate them with any **RsslReactorChannel**, which opens the private stream connection and negotiates any specified behaviors. Prior to opening a tunnel stream, you must implement the **RsslTunnelStreamStatusEventCallback**, which is described in Section 6.9.4.

6.9.2.1 rsslReactorOpenTunnelStream Method

METHOD NAME	DESCRIPTION
rsslReactorOpenTunnelStream	Begins the establishment of a tunnel stream. The RsslTunnelStream is returned via the RsslTunnelStreamStatusEventCallback as specified on the RsslTunnelStreamOpenOptions . For more details, refer to Section 6.9.2.2.

Table 67: rsslReactorOpenTunnelStream Method

6.9.2.2 RsslTunnelStreamOpenOptions

The **RsslTunnelStreamOpenOptions** contain event handler associations and options for use in creating a tunnel stream.

CLASS MEMBER	DESCRIPTION
domainType	Indicates the domain for which the tunnel stream is established. Set this to the domain specified on the service on which the Enterprise Transport API opens the tunnel stream.
streamId	Indicates the stream ID to use for the tunnel stream. Though substreams will flow within this stream ID, each will have their own independent stream ID. For example, a tunnel stream can have an ID of 10. If a substream is opened to retrieve TRI data, the substream can have a stream ID of 5, though it is encapsulated in the tunnel stream whose stream ID is 10.
serviceId	Indicates the service ID of the service on which you open the tunnel stream.
userSpecPtr	Indicates a user-specified object passed in via these options and then associated with the RsslTunnelStream .
statusEventCallback	Specifies an instance of the callback for RsslTunnelStreamStatusEvents , which provides the RsslTunnelStream on initial connection, and after the tunnel stream is established, communicates the tunnel stream's state information. For further details, refer to Section 6.9.4.
queueMsgCallback	Specifies the instance of the callback used to handle Queue Messages received on this RsslTunnelStream . <ul style="list-style-type: none"> For details on the RsslTunnelStreamMsgCallback, refer to Section 6.9.4. For details on various Queue Messages, refer to Section 8.6.
defaultMsgCallback	Specifies the instance of the callback that handles all other content received on this RsslTunnelStream . For further details, refer to Section 6.9.4.
name	Specifies the tunnel stream name, which is provided to the remote application. name cannot be longer than 255 characters.
responseTimeout	Sets the duration (in seconds) to wait for a provider to respond to a tunnel stream open request. If the provider does not respond in time, an RsslTunnelStreamStatusEvent is sent to the application to indicate that the tunnel stream was not opened.
guaranteedOutputBuffers	Sets the number of guaranteed output buffers available for the tunnel stream.
pAuthLoginRequest	Specifies the RsslRDMLLoginRequest to send if RsslClassOfService.authentication.type is set to RDM_COS_AU_OMM_LOGIN . If absent, the API uses the login request provided on the RsslReactorOMMConsumerRole.pLoginRequest .
classOfService	The class of service of the tunnel stream to be opened. For further details on RsslClassOfService , refer to Section 6.9.3.

Table 68: RsslTunnelStreamOpenOptions

6.9.3 Negotiating Stream Behaviors: Class of Service

RsslClassOfService is used to negotiate **RsslTunnelStream** behaviors. Negotiated behaviors are divided into five categories: common, authentication, flow control, data integrity, and guarantee.

- When a consumer application calls **rsslReactorOpenTunnelStream**, it sets the **RsslTunnelStreamOpenOptions.classOfService** members to manage and control tunnel stream behaviors. The consumer passes these settings to the connected provider.
- When a provider application receives an **RsslTunnelStreamRequestEvent**, the provider calls **rsslTunnelStreamRequestGetCos** to retrieve the behaviors requested by the consumer.

After tunnel stream negotiation is complete, the provider and consumer each receive an **RsslTunnelStreamStatusEvent** where each can view the negotiated behaviors on the **RsslTunnelStream** structure.

NOTE: Do not modify the **RsslClassOfService** member of the **RsslTunnelStream**.

The enumerations given for members described in this section can be found in **rsslIRDM.h**.

6.9.3.1 ClassOfService Common Member

Common elements describe options related to the exchange of messages, such as the maximum message size and desired exchange protocol.

MEMBER	DEFAULT	RANGE/ ENUMERATIONS	DESCRIPTION
maxFragmentSize	6144	1 – 2,147,483,647	The maximum size of message fragments exchanged on the tunnel stream. This value is set only by providers when accepting a tunnel stream.
maxMsgSize	614400	1 – 2,147,483,647	The maximum size of messages exchanged on the tunnel stream. This value is set only by providers when accepting a tunnel stream.
protocolMajorVersion	RSSL_RWF_MAJOR_VERSION	0 – 255	The major version of the protocol specified by protocolType .
protocolMinorVersion	RSSL_RWF_MINOR_VERSION	0 – 255	The minor version of the protocol specified by protocolType .
protocolType	RSSL_RWF_PROTOCOL_TYPE	0 – 255	Identifies the protocol of the messages exchanged on the tunnel stream.

Table 69: RsslClassOfService.common Structure Members

6.9.3.2 ClassOfService Authentication Member

The authentication member contains options to authenticate a consumer to the corresponding provider.

MEMBER	DEFAULT	RANGE/ ENUMERATIONS	DESCRIPTION
type	RDM_COS_AU_NOT_REQUIRED	RDM_COS_AU_NOT_REQUIRED == 0, RDM_COS_AU_OMM_LOGIN == 1	Indicates the type of authentication, if any, to perform on the tunnel stream. For further details on authentication, refer to Section 6.9.1.

Table 70: RsslClassOfService.authentication Structure Members

6.9.3.3 ClassOfService Flow Control Members

The flow control member contains options related to flow control, such as the type and the allowed window of outstanding data.

MEMBER	DEFAULT	RANGE/ ENUMERATIONS	DESCRIPTION
type	RDM_COS_FC_NONE	RDM_COS_FC_NONE == 0, RDM_COS_FC_BIDIRECTIONAL == 1	Indicates the type of flow control (if any) to apply to the tunnel stream.
recvWindowSize	-1	0 – 2,147,483,647	Sets the amount of data (in bytes) that the remote peer can send to the application over a reliable tunnel stream. If type is set to RDM_COS_FC_NONE , this parameter has no effect. -1 indicates that the application wants to use the default value for the negotiated flow control type. In this case, if type is set to RDM_COS_FC_BIDIRECTIONAL , the default is 12288 .
sendWindowSize	None	0 – 2,147,483,647	Indicates the amount of data (in bytes) the application can send to the remote peer on a reliable tunnel stream. This value is provided on the RsslTunnelStream object and does not need to be set when opening or accepting a tunnel stream. This value is retrieved from the remote end and is informational, as flow control is performed by the API. When room is available in the window, the API transmits more content as submitted by the application. If type is set to RDM_COS_FC_NONE , this parameter has no effect.

Table 71: RsslClassOfService.flowControl Structure Members

6.9.3.4 ClassOfService Data Integrity Member

The data integrity member contains options related to the reliability of content exchanged over the tunnel stream.

MEMBER	DEFAULT	RANGE	DESCRIPTION
type	RDM_COS_DI_BEST_EFFORT	RDM_COS_DI_BEST_EFFORT == 0, RDM_COS_DI_RELIABLE == 1	<p>Sets the level of reliability for message transmission on the tunnel stream. If set to RDM_COS_DI_RELIABLE, data is retransmitted as needed over the tunnel stream to ensure that all messages are delivered in the correct order.</p> <p>NOTE: At this time, RDM_COS_DI_RELIABLE is the only supported option.</p>

Table 72: `RsslClassOfService.dataIntegrity` Structure Members

6.9.3.5 ClassOfService Guarantee Members

The guarantee member contains options related to the guarantee of content submitted over the tunnel stream.

Consumer applications performing Queue Messaging to a Queue Provider should set the `ClassOfService.guarantee.type` to **RDM_COS_GU_PERSISTENT_QUEUE**.

MEMBER	DEFAULT	RANGE	DESCRIPTION
type	RDM_COS_GU_NONE	RDM_COS_GU_NONE == 0, RDM_COS_GU_PERSISTENT_QUEUE == 1	<p>Indicates the level of guarantee that will be performed on this stream.</p> <p>RDM_COS_GU_PERSISTENT_QUEUE is not supported for provider applications.</p> <p>NOTE: If type is set to RDM_COS_GU_PERSISTENT_QUEUE for a consumer application, the data integrity type must also be set to RDM_COS_DI_RELIABLE and the flow control type to RDM_COS_FC_BIDIRECTIONAL.</p>
persistLocally	RSSL_TRUE	RSSL_FALSE, RSSL_TRUE	<p>Indicates whether messages are persisted locally on the tunnel stream.</p> <p>When type is RDM_COS_GU_NONE, this member has no effect.</p>
persistenceFilePath	NULL	n/a	<p>File path where files containing persistent messages may be stored.</p> <p>If set to NULL, the current working directory is used.</p> <p>When type is RDM_COS_GU_NONE, or when persistLocally is set to RSSL_FALSE, this member has no effect.</p>

Table 73: `RsslClassOfService.guarantee` Structure Members

6.9.4 Tunnel Stream Callback Functions and Event Types

Various tunnel stream callbacks return their information via specific event objects. The following table defines these events.

EVENT	EVENT DESCRIPTION	CLASS MEMBER	CLASS MEMBER DESCRIPTION
RsslTunnelStreamStatusEvent	This event presents the tunnel stream and its status.	pReactorChannel	A pointer to the RsslReactorChannelTunnelStream with which this tunnel stream is associated.
		pState	Indicates status information associated with the RsslTunnelStream . For example: <ul style="list-style-type: none"> A state of OPEN and OK indicates that the tunnel stream is established and content should be flowing as expected. A state of CLOSED_RECOVER or SUSPECT indicates that the connection or tunnel stream might be lost. However, if performing guaranteed messaging, content might be persisted by the reactor and communicated upon recovery of the tunnel stream.
		pRsslMsg	A pointer to an RsslMsg structure.
		pAuthInfo	If the event was produced by an authentication message, pAuthInfo is populated by an RsslTunnelStreamAuthInfo structure. For more information, refer to Section 6.9.1.
RsslTunnelStreamMsgEvent	This event presents content received on the RsslTunnelStream . If a more specific handler (i.e., RsslTunnelStreamQueueMsgCallback) is also configured, messages of that type will go to their specific handler.	pReactorChannel	A pointer to the RsslReactorChannelTunnelStream with which this tunnel stream is associated.
		pRsslMsg	A pointer to an RsslMsg structure, used to deliver any OMM content or opaque content.
		pErrorInfo	Used to convey error information, when applicable.
RsslTunnelStreamQueueMsgEvent	This event presents any queue message content received on the RsslTunnelStream .	base	An RsslTunnelStreamMsgEvent . Refer to Section 6.9.4.2.
		pQueueMsg	A pointer to a Queue Message containing OMM content or opaque content exchanged with a Queue Provider. Refer to subsequent chapters for information about Queue Messages.

Table 74: Tunnel Stream Callback Event Types

6.9.4.1 Tunnel Stream Callback Functions

The **RsslTunnelStream** delivers events via the following user-implemented callback functions. These callback functions return event objects as defined in Section 6.9.4.2. Each callback returns the **RsslTunnelStream** on which the event occurred along with the event itself.

CALLBACK FUNCTION	DESCRIPTION
RsslTunnelStreamStatusEventCallback	Communicates status information about the tunnel stream. Additionally, this callback delivers the RsslTunnelStream object after the enhanced private stream is established. This callback provides an RsslTunnelStreamStatusEvent to the application. Details about this event are available in Section 6.9.4.2.
RsslTunnelStreamDefaultMsgCallback	Similar to the ReactorChannel 's defaultMsgCallback , content received by the tunnel stream are returned via this callback if it is not handled by a more specific content handler, such as the RsslTunnelStreamQueueMsgCallback . This callback provides an RsslTunnelStreamMsgEvent to the application. Details about this event are available in Section 6.9.4.2.
RsslTunnelStreamQueueMsgCallback	Any queue messages are delivered via this callback and presented to the user in their native queue message formats. If unspecified, queue messages are delivered via the RsslTunnelStreamDefaultMsgCallback ; however they are not presented in a queue message format. This callback provides a RsslTunnelStreamQueueMsgEvent to the application. Details about this event are available in Section 6.9.4.2.

Table 75: Tunnel Stream Callback Functions

6.9.4.2 Tunnel Stream Callback Event Types

Various tunnel stream callbacks return their information via specific event objects. The following table defines these events.

EVENT	EVENT DESCRIPTION	CLASS MEMBER	CLASS MEMBER DESCRIPTION
RsslTunnelStreamStatusEvent	This event presents the tunnel stream and its status.	pReactorChannel	A pointer to the RsslReactorChannelTunnelStream with which this tunnel stream is associated.
		pState	Indicates status information associated with the RsslTunnelStream . For example: <ul style="list-style-type: none"> A state of OPEN and OK indicates that the tunnel stream is established and content should be flowing as expected. A state of CLOSED_RECOVER or SUSPECT indicates that the connection or tunnel stream might be lost. However, if performing guaranteed messaging, content might be persisted by the reactor and communicated upon recovery of the tunnel stream.
		pRsslMsg	A pointer to an RsslMsg structure.
		pAuthInfo	If the event was produced by an authentication message, pAuthInfo is populated by an RsslTunnelStreamAuthInfo structure. For more information, refer to Section 6.9.1.
RsslTunnelStreamMsgEvent	This event presents content received on the RsslTunnelStream . If a more specific handler (i.e., RsslTunnelStreamQueueMsgCallback) is also configured, messages of that type will go to their specific handler.	pReactorChannel	A pointer to the RsslReactorChannelTunnelStream with which this tunnel stream is associated.
		pRsslMsg	A pointer to an RsslMsg structure, used to deliver any OMM content or opaque content.
		pErrorInfo	Used to convey error information, when applicable.
RsslTunnelStreamQueueMsgEvent	This event presents any queue message content received on the RsslTunnelStream .	base	An RsslTunnelStreamMsgEvent . Refer to Section 6.9.4.2.
		pQueueMsg	A pointer to a Queue Message containing OMM content or opaque content exchanged with a Queue Provider. Refer to subsequent chapters for information about Queue Messages.

Table 76: Tunnel Stream Callback Event Types

6.9.5 Opening a Tunnel Stream Code Sample

The following code sample illustrates how to open a tunnel stream. The example assumes that a Reactor and ReactorChannel are already open and properly established.

```
// Basic sample for event handlers

// RsslTunnelStreamStatusEventCallback
RsslReactorCallbackRet tunnelStreamStatusEventCallback(RsslTunnelStream
    *pTunnelStream, RsslTunnelStreamStatusEvent *pEvent)
{
    printf("Status of Tunnel Stream %d is %d:%d\n", pTunnelStream->streamId, pEvent->pState-
        >streamState, pEvent->pState->dataState);
    return RSSL_RC_CRET_SUCCESS;
}

// RsslTunnelStreamDefaultMsgCallback
RsslReactorCallbackRet tunnelStreamDefaultMsgCallback(RsslTunnelStream *pTunnelStream,
    RsslTunnelStreamMsgEvent *pEvent)
{
    printf("Received content on Tunnel Stream %d\n", pTunnelStream->streamId);
    return RSSL_RC_CRET_SUCCESS;
}

// RsslTunnelStreamQueueMsgCallback
RsslReactorCallbackRet tunnelStreamQueueMsgCallback(RsslTunnelStream *pTunnelStream,
    RsslTunnelStreamQueueMsgEvent *pEvent)
{
    printf("Received Queue Message on Tunnel Stream %d\n", pTunnelStream->streamId);
    return RSSL_RC_CRET_SUCCESS;
}

int openTunnelStream()
{
    RsslTunnelStreamOpenOptions _openOptions;
    RsslErrorInfo _errorInfo;

    rsslClearTunnelStreamOpenOptions(&_openOptions);

    // populate the options and enable guaranteed delivery for communication with a Queue Provider
    _openOptions.classOfService.guarantee.type = RDM_COS_GU_PERSISTENT_QUEUE;
    _openOptions.classOfService.dataIntegrity = RDM_COS_DI_RELIABLE;
    _openOptions.classOfService.flowControl = RDM_COS_FC_BIDIRECTIONAL;
    _openOptions.classOfService.guarantee.persistLocally = RSSL_TRUE;
    _openOptions.streamId = TUNNEL_STREAM_ID;
    _openOptions.domainType = RSSL_DMT_QUEUE_MESSAGING;
    _openOptions.serviceId = QUEUE_MESSAGING_SERVICE_ID;
    // specify the event handlers
    _openOptions.statusEventCallback = tunnelStreamStatusEventCallback;
    _openOptions.defaultMsgCallback = tunnelStreamDefaultMsgCallback;
```

```

_openOptions.queueMsgCallback = tunnelStreamQueueMsgCallback;

if ((rsslReactorOpenTunnelStream(_pReactorChannel, &_openOptions, &_errorInfo)) !=
    RSSL_RET_SUCCESS)
{
    printf("rsslReactorOpenTunnelStream failed!");
    return RSSL_RET_FAILURE;
}

printf("rsslReactorOpenTunnelStream succeeded!");
return RSSL_RET_SUCCESS;
}

```

Code Example 14: Opening a Tunnel Stream

6.9.6 Accepting Tunnel Streams

OMM provider applications can accept tunnel streams provided on an **RsslReactorChannel** (enabled by specifying a **RsslTunnelStreamListenerCallback** on the **RsslReactorOMMProviderRole**).

When a consumer opens a tunnel stream, the **RsslTunnelStreamListenerCallback** receives an **RsslTunnelStreamRequestEvent**. At this point, the provider should call **rsslTunnelStreamRequestGetCos** to retrieve the **RsslClassOfService** requested by the tunnel stream and ensure that the parameters indicated by the members of that class of service match what the provider allows. The provider can also check the **RsslTunnelStreamRequestEvent.classOfServiceFilter** to determine which behaviors the consumer supports. For more information on this filter, refer to Section 6.9.6.1.

- To accept a tunnel stream, the provider must call **rsslReactorAcceptTunnelStream** with the given **RsslTunnelStreamRequestEvent**. Further events regarding the accepted stream are provided in the specified **RsslReactorAcceptTunnelStreamOptions.statusEventCallback**.
- To reject a tunnel stream, the provider calls **rsslReactorRejectTunnelStream** with the given **RsslTunnelStreamRequestEvent**. No further events are received for that tunnel stream.

Queue messaging (an **RsslClassOfService.guarantee.type** setting of **RDM_COS_GU_PERSISTENT_QUEUE**) is not supported for provider applications.

The API automatically rejects tunnel streams that contain invalid information. When this happens, the provider application receives warnings via an **RsslReactorChannelEvent**. The type will be set to **RSSL_RC_CET_WARNING** and the **RsslErrorInfo** in the event will contain text describing the reason for the rejection.



WARNING! Ensure that the provider application calls **rsslReactorAcceptTunnelStream** or **rsslReactorRejectTunnelStream** before returning from the **RsslTunnelStreamListenerCallback**. If not, the provider application will receive a warning via an **RsslReactorChannelEvent** similar to the above, and the stream will be automatically rejected.

6.9.6.1 Reactor Tunnel Stream Listener Callback and Tunnel Stream Request Event

Providers that want to handle tunnel streams from connected consumers can specify a **RsslTunnelStreamListenerCallback**. This callback informs the provider application of any consumer tunnel stream requests.

The provider can specify this callback on the **RsslReactorOMMProviderRole**, which has the following signature:

```
RsslTunnelStreamListenerCallback(RsslTunnelStreamRequestEvent*, RsslErrorInfo*)
```

For more information on the **RsslReactorOMMProviderRole**, refer to Section 6.3.3.

An **RsslTunnelStreamRequestEvent** is returned to the application via the **RsslTunnelStreamListenerCallback**.

STRUCTURE MEMBER	DESCRIPTION
pReactorChannel	Specifies the RsslReactorChannel on which the event was received.
streamId	Specifies the stream ID of the requested tunnel stream.
domainType	Specifies the domain type of the requested tunnel stream.
serviceId	Specifies the service ID of the requested tunnel stream.
name	Specifies the name of the requested tunnel stream.
classOfServiceFilter	Sets a filter that indicates which RsslClassOfService members are present. The provider can use this filter to determine whether behaviors are supported by the consumer and if needed, reject the tunnel stream before calling rsslTunnelStreamRequestGetCos to get the full RsslClassOfService . For enumerations of the flags present in this filter, refer to RsslTunnelStreamCoSFilterFlags in rsslRDM.h .

Table 77: **RsslTunnelStreamRequestEvent** Structure Members

6.9.6.2 rsslReactorAcceptTunnelStream Function

FUNCTION NAME	DESCRIPTION
rsslReactorAcceptTunnelStream	Accepts a tunnel stream requested by a consumer. The RsslTunnelStream is returned in the RsslTunnelStreamStatusEventCallback specified on the RsslReactorAcceptTunnelStreamOptions . For more information, refer to Section 6.9.6.3.

Table 78: **rsslReactorAcceptTunnelStream** Function

6.9.6.3 RsslReactorAcceptTunnelStreamOptions

OPTION	DESCRIPTION
statusEventCallback	Specifies the instance of the callback for RsslTunnelStreamStatusEvents , which provides the RsslTunnelStream on initial connection and then communicates state information about the tunnel afterwards. For details on the RsslTunnelStreamStatusEventCallback , refer to Section 6.9.4.1.
defaultMsgCallback	Specifies the instance of the callback used to handle all other content received on this RsslTunnelStream . For details on RsslTunnelStreamDefaultMsgCallback , refer to Section 6.9.4.1.
userSpecPtr	Specifies a user-defined pointer passed in via these options and then associated with the RsslTunnelStream .
classOfService	Specifies an RsslClassOfService with members indicating behaviors that the application wants to apply to the RsslTunnelStream . For more information on class of service, refer to Section 6.9.3.
guaranteedOutputBuffers	Sets the number of pooled buffers available to the application when writing content to RsslTunnelStream .

Table 79: RsslReactorAcceptTunnelStreamOptions Options

6.9.6.4 rsslReactorRejectTunnelStream Function

FUNCTION NAME	DESCRIPTION
rsslReactorRejectTunnelStream	Rejects a tunnel stream requested by a consumer. No further events will be received for this tunnel stream. For more information, refer to Section 6.9.6.5.

Table 80: rsslReactorRejectTunnelStream Function

6.9.6.5 RsslReactorRejectTunnelStreamOptions

OPTION	DESCRIPTION
state	An RsslState to send to the consumer. The application can use the state.streamState , state.dataState , and state.text to indicate the nature of the rejection.
pCos	An optional RsslClassOfService to send to the consumer. If rejecting the stream due to a problem with the RsslClassOfService parameters from the RsslTunnelStreamRequestEvent , the provider application should populate this with the associated parameters.

Table 81: RsslReactorRejectTunnelStreamOptions Options

6.9.6.6 Accepting a Tunnel Stream Code Sample

The following code illustrates how to accept a tunnel stream requested by a consumer. The example presumes that a Reactor and Reactor Channel are already open and properly established.

```
RsslReactorCallbackRet tunnelStreamListenerCallback(RsslTunnelStreamRequestEvent *pEvent,
    RsslErrorInfo *pErrorInfo)
{
    RsslErrorInfo errorInfo;
    RsslRet ret;
    RsslClassOfService cos;
    RsslReactorAcceptTunnelStreamOptions acceptOpts;
    ret = rsslTunnelStreamRequestGetCos(pEvent, &cos, &errorInfo);

    /* Now presuming that the application wishes to accept the tunnel stream. */
    rsslClearReactorAcceptTunnelStreamOptions(&acceptOpts);
    acceptOpts.statusEventCallback = tunnelStreamStatusEventCallback;
    acceptOpts.defaultMsgCallback = tunnelStreamDefaultMsgCallback;

    /* Set desired ClassOfService options. */
    /* For this sample, set authentication to match consumer. */
    acceptOpts.classOfService.authentication.type = cos.authentication.type;
    acceptOpts.classOfService.flowControl.type = RDM_COS_FC_BIDIRECTIONAL;
    acceptOpts.classOfService.dataIntegrity.type = RDM_COS_DI_RELIABLE;
    /* ... (set additional members, based on what is desired by the provider) */

    ret = rsslReactorAcceptTunnelStream(pEvent, &acceptOpts, &errorInfo);

    return RSSL_RC_CRET_SUCCESS;
}
```

Code Example 15: Accepting a Tunnel Stream Code Example

6.9.6.7 Rejecting a Tunnel Stream Code Sample

The following code illustrates how to reject a tunnel stream requested by a consumer. The example presumes that a Reactor and Reactor Channel are already open and properly established.

```
RsslReactorCallbackRet tunnelStreamListenerCallback(RsslTunnelStreamRequestEvent *pEvent,
    RsslErrorInfo *pErrorInfo)
{
    RsslErrorInfo errorInfo;
    RsslRet ret;
    RsslClassOfService cos;

    ret = rsslTunnelStreamRequestGetCos(pEvent, &cos, &errorInfo);

    /* Now presuming that the application wishes to reject the tunnel stream
     * Because it only communicates using the RWF protocol type. */

    if (cos.common.protocolType != RSSL_RWF_PROTOCOL_TYPE)
    {
        RsslReactorRejectTunnelStreamOptions rejectOpts;
        RsslClassOfService expectedCos;
        rsslClearReactorRejectTunnelStreamOptions(&rejectOpts);

        rejectOpts.state.streamState = RSSL_STREAM_CLOSED;
        rejectOpts.state.dataState = RSSL_DATA_SUSPECT;
        rejectOpts.state.text.data = "This provider only communicates using the RWF protocol.";
        rejectOpts.state.text.length = (RsslUInt32)strlen(rejectOpts.state.text.data);

        /* Set what the class of service is expected to be. */
        rsslClearClassOfService(&expectedCos);
        expectedCos.common.protocolType = RSSL_RWF_PROTOCOL_TYPE;
        expectedCos.common.protocolMajorVersion = RSSL_RWF_MAJOR_VERSION;
        expectedCos.common.protocolMinorVersion = RSSL_RWF_MINOR_VERSION;
        expectedCos.authentication.type = RDM_COS_AU_NOT_REQUIRED;
        expectedCos.flowControl.type = RDM_COS_FC_BIDIRECTIONAL;
        expectedCos.dataIntegrity.type = RDM_COS_DI_RELIABLE;
        /* ... (set additional members, based on what is desired by the provider) */

        rejectOpts.pCos = &expectedCos;

        ret = rsslReactorRejectTunnelStream(pEvent, &rejectOpts, &errorInfo);
    }

    return RSSL_RC_CRET_SUCCESS;
}
```

Code Example 16: Rejecting a Tunnel Stream Code Example

6.9.7 Receiving Content on a TunnelStream

Invoking the **RsslReactorChannel.dispatch** method reads and processes inbound content, where any information received on this **RsslTunnelStream** will be delivered to the application via the tunnel stream callback methods specified via **rsslReactorOpenTunnelStream** or **rsslReactorAcceptTunnelStream**.

Dispatching this content works in the same manner as dispatching any other content on the reactor.

- Tunnel stream callback methods are described in Section 6.9.4.
- Tunnel stream callback methods deliver the events described in Section 6.9.4.2.

6.9.8 Sending Content on a TunnelStream

When you send content on an **RsslTunnelStream**: get a buffer from the **RsslTunnelStream**, encode your content into the buffer, and then use the **rsslTunnelStreamSubmitMsg** method to push the content out over the **RsslTunnelStream**. By obtaining a buffer from the **RsslTunnelStream**, the reactor can then properly handle any negotiated behaviors, making this functionality nearly transparent.

6.9.8.1 Tunnel Stream Buffer Methods

METHOD NAME	DESCRIPTION
rsslTunnelStreamGetBuffer	Obtains a buffer from the RsslTunnelStream . To properly enforce negotiated behaviors on content in the buffer, the Enterprise Transport API associates the buffer with the tunnel stream from which it is obtained.
rsslTunnelStreamGetInfo	Gets information about the Tunnel Stream by returning the RsslTunnelStreamInfo structure. For details on RsslTunnelStreamInfo methods, refer to Section 6.9.8.5.
rsslTunnelStreamReleaseBuffer	Releases a buffer back to the RsslTunnelStream from which it came. You should release any buffer that you do not submit. Releasing the buffer ensures it is properly recycled and can be reused. NOTE: If you submit a buffer properly, you do not need to release it, because the submit method automatically releases it after sending the content on the RsslTunnelStream .

Table 82: Tunnel Stream Buffer Methods

6.9.8.2 Tunnel Stream Submit

The submit method is used to write content to the **RsslTunnelStream**. This method also enforces any specified behaviors on submitted content (e.g., if guaranteed messaging is specified, this content follows all configured persistence options).

METHOD NAME	DESCRIPTION
rsslTunnelStreamSubmitMsg	Allows the user to pass in RDM message content, including Queue Messages, that will be processed and sent over the RsslTunnelStream . This method has additional options that can be specified via the RsslTunnelStreamSubmitOptions . Currently, the only available members of the option structure allow the user to pass in an RDM message or an RsslMsg structure containing their content.
rsslTunnelStreamSubmit	Allows the user to pass in a buffer populated with content that will be processed and sent over the RsslTunnelStream .

Table 83: Tunnel Stream Submit Method

6.9.8.3 RsslTunnelStreamSubmitOption

When calling `rsslTunnelStreamSubmitMsg`, you can use `RsslTunnelStreamSubmitOptions` to provide the `containerType` option.

MEMBER	DESCRIPTION
<code>containerType</code>	<p>Specifies the type of data in the buffer being submitted.</p> <p>For example:</p> <ul style="list-style-type: none"> If the submitted buffer contains an <code>RsslMsg</code>, set <code>containerType</code> <code>RSSL_DT_MSG</code>. If sending non-RWF data, set <code>containerType</code> to a non-RWF type, such as <code>RSSL_DT_OPAQUE</code>. <p>For more information on possible container types, refer to the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>

Table 84: `RsslTunnelStreamSubmitOptions` Structure Members

6.9.8.4 RsslTunnelStreamSubmitMsgOptions

When calling `rsslTunnelStreamSubmitMsg`, you can use `RsslTunnelStreamSubmitMsgOptions` to provide options the following options:

MEMBER	DESCRIPTION
<code>pRsslMsg</code>	Specifies an <code>RsslMsg</code> populated by the application, which the API encodes and sends over the <code>RsslTunnelStream</code> ; mutually exclusive with <code>pRDMMsg</code> .
<code>pRDMMsg</code>	Specifies an <code>RsslRDMMsg</code> populated by the application, which the API encodes and sends over the <code>RsslTunnelStream</code> ; mutually exclusive with <code>pRsslMsg</code> .

Table 85: `RsslTunnelStreamSubmitMsgOptions` Structure Members

6.9.8.5 RsslTunnelStreamInfo Structure Member

The following table describes values available when using the `rsslTunnelStreamGetInfo` method (for details, refer to Section 6.9.8.1). This information is returned as part of the `RsslTunnelStreamInfo` structure.

STRUCTURE MEMBER	DESCRIPTION
<code>buffersUsed</code>	Returns the total number of buffers used by the Tunnel Stream for a user application.

Table 86: `RsslTunnelStreamInfo` Structure Members

6.9.8.6 Submitting Content on a Tunnel Stream Code Sample

The following code sample is a basic example of writing opaque content to a tunnel stream. This can be combined with the `QueueData` message samples in subsequent chapters to send content to a Queue Provider.

```
int submitMessage()
{
    RsslErrorInfo _errorInfo;
    RsslBuffer *pBuffer;
    RsslTunnelStreamGetBufferOptions _getBufferOpts;
    RsslTunnelStreamSubmitOptions _submitOpts;

    // gets a buffer of 50 bytes to put content into.
    rsslClearTunnelStreamGetBufferOptions(&_getBufferOpts);
    _getBufferOpts.size = 50;
    pBuffer = rsslTunnelStreamGetBuffer(pTunnelStream, &_getBufferOpts, _errorInfo);

    // put generic content into the buffer
    pBuffer->data = "Hello World!";
    pBuffer->length = 12;

    rsslClearTunnelStreamSubmitOptions(&_submitOpts);
    _submitOpts.containerType = RSSL_DT_OPAQUE;
    if ((rsslTunnelStreamSubmit(pTunnelStream, pBuffer, &_submitOpts, &_errorInfo)) !=
        RSSL_RET_SUCCESS)
    {
        printf("Content submission failed!");
        // Because submission failed, we need to return the buffer to the tunnel stream
        rsslTunnelStreamReleaseBuffer(&_buffer, &_errorInfo);

        return RSSL_RET_FAILURE;
    }

    printf("Content submission succeeded!");
    // Thanks to successful submission, we do not need to release the buffer because the Reactor will.
    return RSSL_RET_SUCCESS;
}
```

Code Example 17: Submitting Content on a Tunnel Stream

6.9.8.7 Closing a Tunnel Stream

When an application has completed its use of an `RsslTunnelStream`, it can be closed.

METHOD NAME	DESCRIPTION
<code>rsslReactorCloseTunnelStream</code>	Closes a tunnel stream. Once closed, any content stored for guaranteed messaging or reliable delivery will be cleaned up.

Table 87: `rsslReactorCloseTunnelStream` Method

6.9.8.8 RsslTunnelStreamCloseOptions

When calling `rsslTunnelStreamClose`, you can use `RsslTunnelStreamCloseOptions` to provide the `finalStatusEvent` option.

MEMBER	DESCRIPTION
<code>finalStatusEvent</code>	Indicates that the application wants to receive a final <code>RsslTunnelStreamStatusEvent</code> whenever the tunnel stream closes. If set to <code>RSSL_TRUE</code> , the tunnel stream is cleaned up after the application receives the final <code>RsslTunnelStreamStatusEvent</code> .

Table 88: RsslTunnelStreamCloseOptions Structure Members

6.9.8.9 Closing a Tunnel Stream Code Sample

The following code sample illustrates how to close a tunnel stream.

```
int closeTunnelStream()
{
    RsslTunnelStreamCloseOptions _closeOpts;

    rsslClearTunnelStreamCloseOptions(&_closeOpts);
    _closeOpts.finalStatusEvent = RSSL_TRUE;

    if ((rsslReactorCloseTunnelStream(pTunnelStream, &_closeOpts, &_errorInfo)) != RSSL_RET_SUCCESS)
    {
        printf("Closing tunnel stream failed!");
        return RSSL_RET_FAILURE;
    }

    printf("Tunnel Stream closed successfully.");
    return RSSL_RET_SUCCESS;
}
```

Code Example 18: Closing a Tunnel Stream

6.10 Cloud Connectivity

For details on workflows and routines associated with connecting to the cloud, refer to Chapter 7.

6.10.1 rsslReactorQueryServiceDiscovery

You use the **rsslReactorQueryServiceDiscovery** method to query service endpoints from the Real-Time - Optimized service discovery.

6.10.1.1 rsslReactorQueryServiceDiscovery Method

METHOD	DESCRIPTION
rsslReactorQueryServiceDiscovery	<p>Uses the passed-in RsslReactor to query service endpoints from the Real-Time Optimized service according to the rsslReactorQueryServiceDiscoveryOptions that you specify (listed in Section 6.10.1.2).</p> <p>Error handling is managed by the RsslErrorInfo structure.</p> <p>NOTE: rsslReactorQueryServiceDiscovery tries to use the existing access token from the centralized token management process (as described in section Section 7.4.5) for the passed-in username (if any). Otherwise, rsslReactorQueryDiscovery retrieves a new access token by sending a token request using the username and password parameters.</p> <p>NOTE: The proxy settings specified in RsslReactorServiceDiscoveryOptions (proxyHostName, proxyPort, proxyUserName, proxyPasswd, proxyDomain) have lower precedence than the proxy settings specified in RsslCreateReactorOptions.restProxyOptions. For further information, refer to Section 6.2.1.2.</p>

Table 89: rsslReactorQueryServiceDiscovery Method

6.10.1.2 RsslReactorServiceDiscoveryOptions

MEMBER	DESCRIPTION
audience	Optional and only used with Version 2 OAuth ClientCredentials with JWT logins. An RsslRsslBuffer specifies the JWT's audience claim field. By default, the Enterprise Transport API uses https://login.ciam.refinitiv.com/as/token.oauth2
clientId	Required. An RsslBuffer that specifies a unique ID defined for an application making a request to the token service.
clientJWK	Required for Version 2 OAuth ClientCredentials with JWT logins. An RsslBuffer that contains the JWK-formatted private key associated with the Service Account. For further information, refer to the clientSecret member in this section.
clientSecret	An RsslBuffer that specifies the client secret (if one exists) used by the OAuth client to authenticate to the authorization Server.
dataFormat	Optional. An enumeration that specifies the desired data format to use when retrieving service endpoints from the service discovery. For available values, refer to Section 6.10.1.4.
password	Required. An RsslBuffer that specifies a password for authorization with the token service.
proxyHostName	Optional. An RsslBuffer that specifies a proxy server hostname.
proxyPort	Optional. An RsslBuffer that specifies a proxy server port.
proxyUserName	Optional. An RsslBuffer that specifies a username to perform authorization with a proxy server.

Table 90: RsslReactorServiceDiscoveryOptions Structure Members

MEMBER	DESCRIPTION
proxyPasswd	Optional. An RsslBuffer that specifies a password to perform authorization with a proxy server.
proxyDomain	Optional. An RsslBuffer that specifies the proxy domain of the user to authenticate. Required for NTLM or for Negotiate/Kerberos or for Kerberos authentication protocols.
pServiceEndpointEventCallback	A callback function that receives RsslReactorServiceEndpointEvents . Applications can take service endpoint information from the callback to get an endpoint and establish a connection to the service.
restBlocking	Optional. Specifies whether to send REST blocking for authentication and service discovery requests. The default value is RSSL_TRUE . Setting this parameter to RSSL_FALSE in order to receive the RsslReactorServiceEndpointEvent via the rsslReactorDispatch() function call.
takeExclusiveSignOnControl	Sets whether to use exclusive sign-on control, which forces other applications to sign-out if using the same credentials.
tokenScope	An RsslBuffer that specifies an optional token scope to limit the scope of the generated token from the token service.
transport	Optional. An enumeration that specifies the desired transport protocol to retrieve service endpoints from the service discovery. For available values, refer to Section 6.10.1.3.
userName	Required . An RsslBuffer that specifies a user name for authorization with the token service.
userSpecPtr	Optional. A user-specified pointer which is set on the RsslReactorServiceEndpointEvent . Also refer to Section 6.10.1.5.

Table 90: RsslReactorServiceDiscoveryOptions Structure Members (Continued)

6.10.1.3 RsslReactorDiscoveryTransportProtocol Enumerations

ENUMERATED NAME	DESCRIPTION
RSSLRSSL_RD_TP_INIT = 0	Specifies that the transport's protocol is unknown.
RSSLRSSL_RD_TP_TCP = 1	Specifies that the service discovery should use the TCP transport protocol.
RSSLRSSL_RD_TP_WEBSOCKET = 2	Specifies that the service discovery should use the Websocket transport protocol.

Table 91: RsslReactorDiscoveryTransportProtocol Enumerations

6.10.1.4 RsslReactorDiscoveryDataFormatProtocol Enumerations

ENUMERATED NAME	DESCRIPTION
RSSLRSSL_RD_DP_INIT = 0	Specifies that the transport's data format is unknown.
RSSLRSSL_RD_DP_RWF = 1	Specifies that the service discovery should use the RWF data format.
RSSLRSSL_RD_DP_JSON2 = 2	Specifies that the service discovery should use the tr_json2 data format

Table 92: RsslReactorDiscoveryDataFormatProtocol Enumerations

6.10.1.5 RsslReactorServiceEndpointEvent

MEMBER	DESCRIPTION
pErrorInfo	Returns any information about the error that occurred with the Delivery Platform token service and service discovery. Error information includes its location in the source code.
serviceEndpointInfoCount	Specifies the number of service endpoints in serviceEndpointInfoList .
serviceEndpointInfoList	Lists the service endpoints associated with this event. See also Section 6.10.1.6.
userSpecPtr	Optional. A user-specified pointer associated with this RsslReactorServiceEndpointEvent .
statusCode	Represents the HTTP response status code.

Table 93: RsslReactorServiceEndpointEvent Structure Members

6.10.1.6 RsslReactorServiceEndpointInfo

RsslReactorServiceEndpointEvent represents service endpoint information.

MEMBER	DESCRIPTION
dataFormatList	An RsslBuffer that contains a list of data formats used by the transport.
dataFormatCount	Specifies the number of data formats in dataFormatList .
endPoint	An RsslBuffer that specifies the domain name of the service access endpoint.
locationList	An RsslBuffer object that specifies a list of service locations.
locationCount	Specifies the number of locations in locationList .
port	An RsslBuffer that specifies the port number used to establish connection.
provider	An RsslBuffer that specifies a public cloud provider.
transport	An RsslBuffer that specifies the transport type used to access the service.

Table 94: RsslReactorServiceEndpointEvent Structure Members

6.10.2 OAuth Credential Management

6.10.2.1 RsslReactorOAuthCredential Structure

You use the **RsslReactorOAuthCredential** structure to certify OAuth user credentials when connecting to the cloud.

RsslReactorOAuthCredential includes the following members:

MEMBER	DESCRIPTION
audience	Optional and only used with Version 2 OAuth ClientCredentials with JWT logins. An RsslBuffer that specifies the JWT's audience claim field. By default, the Enterprise Transport API uses https://login.ciam.refinitiv.com/as/token.oauth2
clientId	Required. An RsslBuffer that specifies an authentication parameter. <ul style="list-style-type: none"> Version 1 authentication: Client ID usage with OAuth Password Credentials. For further information, refer to Section 7.4. Version 2 authentication: a unique ID defined for the application that makes the request and is provisioned as part of a service account for the login. For further information, refer to Section 7.5.
clientJWK	Required for Version 2 OAuth ClientCredentials with JWT logins. An RsslBuffer that contains the JWK-formatted private key is associated with the Service Account. For further information, refer to Section 7.5.
clientSecret	Required for Version 2 OAuth ClientCredentials logins. An RsslBuffer that specifies the Service Account "secret" for authentication. For further information, refer to Section 7.5.
password	Required for Version 1 OAuth Password Credentials logins. An RsslBuffer that specifies the password used in tandem with the userName to obtain the access token.
peactorOAuthCredentialEventCallback	A callback function that receives the RssleactorOAuthCredentialEvent to specify the password and/or clientSecret . If peactorOAuthCredentialEventCallback is specified, the Value Added Components Reactor does not store the password or clientSecret . In which case, the application must supply the password whenever receiving a new refresh token. For details on this process, refer to Section 7.4.2.
takeExclusiveSignOnControl	Optional and only used with Version 1 Password Credentials logins. A bool that, if set to true, forces sign-out of any other applications using the same credentials. By default, this is set to true.
tokenScope	An RsslBuffer that specifies the user's resource scope that defines the type of data the user accesses in the cloud. For further details on token scopes, refer to the Delivery Platform APIs tutorial called <i>Authorization - All about tokens</i> in the Developer Community Portal . By default, the Enterprise Transport API uses the scope: trapi.streaming.pricing.read .
userName	Required for Version 1 OAuth Password Credentials logins. An RsslBuffer that specifies the user name used to obtain the access token from the Delivery Platform.

Table 95: RsslReactorOAuthCredential Structure Members

Whenever the Enterprise Transport API needs a new refresh token, it needs to again supply the username, Client ID, and password. But the Enterprise Transport API stores only the username and Client ID, not the password. To obtain the password (and if available, the client secret), the Enterprise Transport API sends the **RsslReactorOAuthCredentialEvent** callback to the application.

MEMBER	DESCRIPTION
RsslReactorChannel	Returns the channel associated with the event.
RsslReactorOAuthCredentialRenewal	Returns a structure with OAuth credentials for renewal authentication with the Delivery Platform.

Table 96: RsslReactorOAuthCredentialEvent Structure Members

6.10.2.2 RsslReactorOAuthCredentialRenewal

MEMBER	DESCRIPTION
audience	Optional and only used with Version 2 OAuth ClientCredentials with JWT logins. An RsslBuffer that specifies the JWT's audience claim field. By default, the Enterprise Transport API uses https://login.ciam.refinitiv.com/as/token/oauth2
clientId	Required. An RsslBuffer that specifies an authentication parameter. <ul style="list-style-type: none"> Version 1 authentication: Client ID usage with OAuth Password Credentials. For further information, refer to Section 7.4. Version 2 authentication: a unique ID defined for the application that makes the request and is provisioned as part of a service account for the login. For further information, refer to Section 7.5.
clientJWK	Required for Version 2 OAuth ClientCredentials with JWT logins. An RsslBuffer that contains the JWK-formatted private key is associated with the Service Account. For further information, refer to Section 7.5.
clientSecret	Required for Version 2 OAuth ClientCredentials logins. An RsslBuffer that specifies the Service Account "secret" for authentication. For further information, refer to Section 7.5.
newPassword	Conditional. An RsslBuffer that specifies the new password when changing the password associated with the specified userName . Include newPassword only when the application wants to change its password, in which case both the current (password) and new password (newPassword) are required.
password	Required for Version 1 OAuth Password Credentials logins. An RsslBuffer that specifies the password, which is sent with the userName to get an access token and a refresh token.
tokenScope	An RsslBuffer that specifies the scope of the generated token.
userName	Conditional. An RsslBuffer that specifies the user name that the Enterprise Transport API sends to the Delivery Platform token service. The RsslReactorOAuthCredentialEventCallback also uses userName when returning sensitive information. Required for Version 1 OAuth Password Credentials logins, except when specifying sensitive information in the RsslReactorOAuthCredentialEventCallback .

Table 97: RsslReactorOAuthCredentialRenewal Members

6.10.2.3 rsslReactorSubmitOAuthCredentialRenewal Method

MEMBERS	DESCRIPTION
rsslReactorSubmitOAuthCredentialRenewal	<p>Uses the passed-in RsslReactor and RsslReactorOAuthCredentialRenewal to submit the application's password (and client secret if available) to the Delivery Platform token service. An application can also use this method to change its password.</p> <p>For a list of options you can use with rsslReactorSubmitOAuthCredentialRenewal, refer to Section 6.10.2.4.</p> <p>If you call this method outside of the RsslReactorOAuthCredentialEventCallback, you should also include pAuthTokenEventCallback to receive a result response.</p> <p>Error handling is managed by the RsslErrorInfo structure.</p> <p>NOTE: The proxy settings specified in RsslReactorOAuthCredentialRenewalOptions (proxyHostName, proxyPort, proxyUserName, proxyPasswd, proxyDomain) have lower precedence than the proxy settings specified in RsslCreateReactorOptions.restProxyOptions. For further information, refer to Section 6.2.1.2.</p>

Table 98: rsslReactorSubmitOAuthCredentialRenewal

6.10.2.4 rsslReactorSubmitOAuthCredentialRenewal Options

OPTION	DESCRIPTION
pAuthTokenEventCallback	<p>A callback function that receives RsslReactorAuthTokenEvents. The Reactor requests a token for the Consumer (i.e., disabling watchlist) and NiProvider applications to send login requests and reissues with the token.</p> <p>pAuthTokenEventCallback is needed only when changing a password without a channel in order to get a response from the request. The application does not have to send a login reissue in this case.</p>
proxyDomain	An RsslBuffer that specifies the domain for authenticated proxies.
proxyHostName	An RsslBuffer that specifies the proxy's host name.
proxyPasswd	An RsslBuffer that specifies the password for authenticated proxies.
proxyPort	An RsslBuffer that specifies the proxy's port.
proxyUserName	An RsslBuffer that specifies the username for authenticated proxies.
renewalMode	A RsslReactorOAuthCredentialRenewalMode that specifies the mode in which the Enterprise Transport API submits OAuth credential renewals. For available ENUMs and their descriptions, refer to Section 6.10.2.5.
userName	Required. An RsslBuffer that specifies a user name for authorization with the token service.

Table 99: rsslReactorSubmitOAuthCredentialRenewal Options

6.10.2.5 RsslReactorOAuthCredentialRenewalMode Enums

MODE	DESCRIPTION
RSSL_ROC_RT_RENEW_TOKEN_WITH_PASSWORD	Use this renewal mode when normally submitting a password to obtain an access and refresh token.
RSSL_ROC_RT_RENEW_TOKEN_WITH_PASSWORD_CHANGE	Use this renewal mode when normally submitting a password to obtain an access and refresh token.

Table 100: RsslReactorOAuthCredentialRenewalMode Enums

6.11 JSON to RWF Protocol Conversion for WebSocket Support

For the consumer to support WebSocket connection requests, you must initialize the consumer and define `wsOpts.protocols`, based on the types of protocol you want to support, as follows:

- To support WebSocket connections for only the RWF sub-protocol, use `RsslConnectOptions.wsOpts.protocols = "rssl.rwf"`.
 - This is the only requirement. For an example of setting `wsOpts`, refer to Section 6.11.1.1.
 - For remaining tasks for managing the Consumer, refer to Chapter 3, Building an OMM Consumer.
- To support WebSocket connection requests for JSON2 and RWF sub-protocols (either alone, or in tandem with RWF), use `RsslConnectOptions.wsOpts.protocols = "rssl.rwf, rssl.json.v2, tr_json2"`.

For details on `wsOpts`, refer to the *Enterprise Transport API C Edition Developer Guide*.

To support WebSocket connections in JSON (either alone, or in tandem with RWF), after initializing the Interactive Provider, you must then perform the following:

- Connect the Consumer over an encrypted WebSocket. For an example, refer to Section 6.11.1.2.
- Define the Reactor JSON Converter event callback by using the `jsonConversionEventCallback` function. For an example, refer to Section 6.11.1.3.
- Define the `ServiceName` to `ServiceId` callback (using the `serviceNameToIdCallback` function). For an example, refer to Section 6.11.1.3.
- Clear and populate a JSON Converter options structure using the callbacks from the preceding tasks. For an example, refer to Section 6.11.1.4.
- Initialize the Reactor JSON Converter using the `rsslReactorInitJsonConverter` function passing in the JSON converter options `RsslReactorJsonConverterOptions`. For an example, refer to Section 6.11.2.4.
- For remaining tasks on managing the Consumer, refer to Chapter 3, Building an OMM Consumer.

6.11.1 Consumer and WebSocket Support

6.11.1.1 Example: Initializing the Consumer to Support WebSocket Connections for RWF Only

```
if (strstr(argv[i], "-webSocket") != 0)
{
    pCommand->cInfo.rsslConnectOptions.connectionType = RSSL_CONN_TYPE_WEBSOCKET; // non-encrypted
                                         connection
    pCommand->cInfo.rsslConnectOptions.wsOpts.protocols = protocolList; // Define the supported list
                                         of sub-protocols, string consisting of a white space or comma
                                         delineated list
}
```

6.11.1.2 Example: Connecting the Consumer over an Encrypted WebSocket

```
else if (strcmp("-encryptedWebSocket", argv[i]) == 0)
{
    pCommand->cInfo.rsslConnectOptions.encryptionOpts.encryptedProtocol = RSSL_CONN_TYPE_WEBSOCKET;
    pCommand->cInfo.rsslConnectOptions.wsOpts.protocols = protocolList;
}
```


6.11.1.3 Example: Defining Reactor JSON Converter Event Callback and ServiceName to ServiceId Callback

```

RsslReactorCallbackRet jsonConversionEventCallback(RsslReactor *pReactor, RsslReactorChannel
    *pReactorChannel, RsslReactorJsonConversionEvent *pEvent)
{
    if (pEvent->pError)
    {
        printf("Error Id: %d, Text: %s\n", pEvent->pError->rsslError.rsslErrorId, pEvent->pError-
            >rsslError.text);
    }

    return RSSL_RC_CRET_SUCCESS;
}

RsslRet serviceNameToIdCallback(RsslReactor *pReactor, RsslBuffer* pServiceName, RsslUInt16*
    pServiceId, RsslReactorServiceNameToIdEvent* pEvent)
{
    ChannelCommand *pCommand;
    int i = 0;

    for (i = 0; i < channelCommandCount; i++)
    {
        pCommand = &chanCommands[i];

        if (pCommand->serviceNameFound)
        {
            if (strncmp(&pCommand->serviceName[0], pServiceName->data, pServiceName->length) == 0)
            {
                *pServiceId = (RsslUInt16)pCommand->serviceId;
                return RSSL_RET_SUCCESS;
            }
        }
    }

    return RSSL_RET_FAILURE;
}

```

6.11.1.4 Example: Clear and Populate a JSON Converter Options Structure

```
RsslReactorJsonConverterOptions jsonConverterOptions;    // stack allocated
    RsslReactorJsonConverterOptions structure

rsslClearReactorOAuthCredential(&oAuthCredential);
rsslClearReactorJsonConverterOptions(&jsonConverterOptions);    // Clear allocated structure

jsonConverterOptions.pDictionary = &(chanCommands[0].dictionary);
jsonConverterOptions.pServiceNameToIdCallback = serviceNameToIdCallback;
jsonConverterOptions.pJsonConversionEventCallback = jsonConversionEventCallback;
```

6.11.1.5 Example: Initialize the Reactor JSON Converter

```
if (rsslReactorInitJsonConverter(pReactor, &jsonConverterOptions, &rsslErrorInfo) != RSSL_RET_SUCCESS)
{
    printf("Error initializing RWF/JSON Converter: %s\n", rsslErrorInfo.rsslError.text);
    exit(-1);
}
```

6.11.2 Interactive Provider and WebSocket Support

For the Interactive Provider to support WebSocket connection requests, you must initialize the Interactive Provider and define **wsOpts.protocols**, based on the types of protocol you want to support, as follows:

- To support WebSocket connections for only the RWF sub-protocol, use **RsslBindOptions.wsOpts.protocols = "rssl.rwf"**.
 - This is the only requirement. For an example of setting **wsOpts**, refer to Section 6.11.2.1.
 - For remaining tasks for managing the Interactive Provider, refer to Chapter 4, Building an OMM Interactive Provider.
- To support WebSocket connection requests for both JSON2 and RWF sub-protocols, use **RsslBindOptions.wsOpts.protocols = "rssl.rwf, rssl.json.v2, tr_json2"**.
- To support WebSocket connection requests only JSON2 sub-protocol, use **RsslBindOptions.wsOpts.protocols = "rssl.json.v2, tr_json2"**.

For details on **wsOpts**, refer to the *Enterprise Transport API C Edition Developer Guide*.

To support the JSON protocol (either alone, or in tandem with RWF), after initializing the Interactive Provider, you must then perform the following:

- Define the Reactor JSON Converter event callback by using the **jsonConversionEventCallback** function. For an example, refer to Section 6.11.2.2.
- Define the **ServiceName** to **ServiceId** callback (using the **serviceNameToIdCallback** function). For an example, refer to Section 6.11.2.2.
- Clear and populate a JSON converter options structure using the callbacks from the preceding tasks. For an example, refer to Section 6.11.2.3.
- Initialize the Reactor JSON Converter using the **rsslReactorInitJsonConverter** function passing in the JSON converter options **RsslReactorJsonConverterOptions**. For an example, refer to Section 6.11.2.4.
- For remaining tasks on managing the Interactive Provider, refer to Chapter 4, Building an OMM Interactive Provider.

6.11.2.1 Example: Initializing the Publisher to Support WebSocket Connections for RWF Only

To support WebSocket connections for only the RWF sub-protocol, initialize the **RsslServer** (for details, refer to Section 4.3) with **RsslBindOptions.wsOpts.protocols = rssl.rwf**.

```
RsslBindOptions.serviceName = portNo;
RsslBindOptions.wsOpts.protocols = "rssl.rwf"
RsslBindOptions.majorVersion = RSSL_RWF_MAJOR_VERSION;
RsslBindOptions.minorVersion = RSSL_RWF_MINOR_VERSION;
RsslBindOptions.protocolType = RSSL_RWF_PROTOCOL_TYPE;
RsslBindOptions.connectionType = RSSL_CONN_TYPE_SOCKET;
```

6.11.2.2 Example: Defining Reactor JSON Converter Event Callback and ServiceName to ServiceId Callback

```
RsslReactorCallbackRet jsonConversionEventCallback(RsslReactor *pReactor, RsslReactorChannel
    *pReactorChannel, RsslReactorJsonConversionEvent *pEvent)
{
    if (pEvent->pError)
    {
        printf("Error Id: %d, Text: %s\n", pEvent->pError->rsslError.rsslErrorId, pEvent->pError-
            >rsslError.text);
    }

    return RSSL_RC_CRET_SUCCESS;
}

RsslRet serviceNameToIdCallback(RsslReactor *pReactor, RsslBuffer* pServiceName, RsslUInt16*
    pServiceId, RsslReactorServiceNameToIdEvent* pEvent)
{
    if (strncmp(&serviceName[0], pServiceName->data, pServiceName->length) == 0)
    {
        *pServiceId = (RsslUInt16)getServiceId();
        return RSSL_RET_SUCCESS;
    }

    return RSSL_RET_FAILURE
}
```

6.11.2.3 Example: Clear and Populate a JSON Converter Options Structure with Callback Information

```
RsslReactorJsonConverterOptions jsonConverterOptions; // stack allocated
    RsslReactorJsonConverterOptions structure
time_t nextSendTime;

rsslClearReactorJsonConverterOptions(&jsonConverterOptions); // Clear allocated structure

jsonConverterOptions.pDictionary = getDictionary();
jsonConverterOptions.defaultServiceId = (RsslUInt16)getServiceId();
jsonConverterOptions.pServiceNameToIdCallback = serviceNameToIdCallback;
jsonConverterOptions.pJsonConversionEventCallback = jsonConversionEventCallback;
```

6.11.2.4 Example: Initialize the Reactor JSON Converter Using the rsslReactorInitJsonConverter Function

```
if (rsslReactorInitJsonConverter(pReactor, &jsonConverterOptions, &rsslErrorInfo) != RSSL_RET_SUCCESS)
{
    printf("Error initializing RWF/JSON Converter: %s\n", rsslErrorInfo.rsslError.text);
    cleanUpAndExit();
}
```

6.11.3 RsslReactorJsonConverterOptions Structure

The **RsslReactorJsonConverterOptions** structure includes the following options:

MEMBER	DESCRIPTION
catchUnknownJsonFids	When converting from JSON to RWF, sets the Enterprise Transport API to catch unknown JSON field IDs. catchUnknownJsonFids is an RsslBool and defaults to RSSL_TRUE .
catchUnknownJsonKeys	When converting from JSON to RWF, sets the Enterprise Transport API to catch unknown JSON keys. catchUnknownJsonKeys is an RsslBool and defaults to RSSL_FALSE .
closeChannelFromFailure	An RsslBool that closes the channel if the Reactor fails to parse a JSON message or if the Reactor receives a JSON error message. closeChannelFromFailure defaults to RSSL_TRUE .
defaultServiceId	If both ServiceName and ServiceID are not set, defaultServiceId specifies a default service ID (an RsslUInt16) for requests. defaultServiceId defaults to -1 (i.e., not set). defaultServiceId accepts a valid range of 0 to 65535.
jsonExpandedEnumFields	An RsslBool that expands enumerated values in field entries to their display values for the JSON protocol. jsonExpandedEnumFields defaults to RSSL_FALSE (do not expand the field).

Table 101: RsslReactorJsonConverterOptions Structure Members


MEMBER	DESCRIPTION
outputBufferSize	<p>Sets the size (an RsslUInt32) that the converter allocates for its output buffer. Defaults to 65535.</p> <p> WARNING! If the output buffer is not large enough, JSON/RWF conversion will fail.</p>
pDictionary	<p>Sets the data dictionary (RsslDataDictionary) that Enterprise Transport API uses to initialize the RWF/JSON converter.</p> <p>pDictionary defaults to 0.</p>
pJsonConversionEventCallback	<p>Specifies the callback function (RsslReactorJsonConversionEventCallback) that receives the RsslReactorJsonConversionEvent if the JSON converter fails to convert a message.</p> <p>pJsonConversionEventCallback defaults to 0.</p> <ul style="list-style-type: none"> For further details on RsslReactorJsonConversionEventCallback, refer to Section 6.11.4. For further details on RsslReactorJsonConversionEvent, refer to Section 6.11.7.
pServiceNameToldCallback	<p>Specifies the callback function (RsslReactorServiceNameToIdCallback) that handles conversion of the ServiceName to ServiceId.</p> <p>pServiceNameToIdCallback defaults to 0.</p> <p>For further details on RsslReactorServiceNameToIdCallback, refer to Section 6.11.5.</p>
userSpecPtr	<p>Specifies a user-defined pointer which is retrieved in the callback function.</p> <p>pUserSpec is a void* type and defaults to 0.</p>

Table 101: RsslReactorJsonConverterOptions Structure Members (Continued)

6.11.4 RsslReactorJsonConversionEventCallback Function

The **RsslReactorJsonConversionEventCallback** function communicates conversion information when the JSON converter fails to convert JSON to RWF messages by providing an **RsslReactorJsonConversionEvent** structure to the application. For further details on **RsslReactorJsonConversionEvent**, refer to Section 6.11.7.

6.11.5 RsslReactorServiceNameToldCallback Function

RsslReactorServiceNameToIdCallback calls back to the application to translate a **ServiceName** to **ServiceId** by application by providing an **RsslReactorServiceNameToIdEvent**. For further details on **RsslReactorServiceNameToIdEvent**, refer to Section 6.11.6.

- If **RsslReactorServiceNameToIdCallback** succeeds, it returns **RSSL_RET_SUCCESS**.
- If **RsslReactorServiceNameToIdCallback** fails, it returns **RSSL_RET_FAILURE**.

PARAMETER	DESCRIPTION
pServiceName	Specifies the ServiceName for which the callback looks up the appropriate ID.
pServiceId	Specifies the ServiceId that the callback populates with the translated ID.

Table 102: RsslReactorServiceNameToldCallback Parameters

6.11.6 RsslReactorServiceNameToIdEvent Structure

The **RsslReactorServiceNameToIdEvent** structure includes the following options:

MEMBER	DESCRIPTION
userSpecPtr	Specifies a user-defined pointer provided when specifying the callback for the RsslReactorServiceNameToIdEvent event. userSpecPtr is a void* type and defaults to 0.

Table 103: RsslReactorServiceNameToIdEvent Structure Members

6.11.7 RsslReactorJsonConversionEvent Structure

The **RsslReactorJsonConversionEvent** structure includes the following options:

MEMBER	DESCRIPTION
pUserSpec	Specifies a user-defined pointer provided when specifying the callback for the RsslReactorServiceNameToIdEvent event. pUserSpec is a void* type and defaults to 0.
pError	Contains any error information (RsslErrorInfo) associated with a JSON conversion. pError defaults to 0.

Table 104: RsslReactorJsonConversionEvent Structure Members

6.12 Reactor Utility Functions

The Enterprise Transport API Reactor provides several additional utility functions. These functions can be used to query more detailed information for a specific connection or change certain **RsslReactorChannel** parameters during run-time. These functions are described in Section 6.12.1 - Section 6.12.3.

6.12.1 General Reactor Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rsslReactorGetChannelInfo</code>	Allows the application to query RsslReactorChannel negotiated parameters and settings and retrieve all current settings. This includes maxFragmentSize and negotiated compression information as well as many other values. For a full list of available settings, refer to the RsslReactorChannelInfo structure defined in Section 6.12.2. This function calls the Enterprise Transport API rsslGetChannelInfo function which has its use and return values described in the Enterprise Transport API C Edition <i>Developers Guide</i> .
<code>rsslReactorIoctl</code>	Allows the application to change various settings associated with the RsslReactorChannel . The available options are defined in Section 6.12.3. This function calls the Enterprise Transport API rsslIoctl function which has its use and return values described in the Enterprise Transport API C Edition <i>Developers Guide</i> .

Table 105: Reactor Utility Functions

6.12.2 RsslReactorChannelInfo Structure Members

The following table describes the values available to the user through using the **rsslReactorGetChannelInfo** function. This information is returned as part of the **RsslReactorChannelInfo** structure.

STRUCTURE MEMBER	DESCRIPTION
<code>rsslChannelInfo</code>	Returns the underlying RsslChannel information. This includes maxFragmentSize , number of output buffers, compression information, and more. The RsslChannelInfo function structure is fully described in the Enterprise Transport API C Edition <i>Developers Guide</i> .
<code>rsslPreferredHostInfo</code>	Structure containing the current channel's Preferred Host configuration, as well as the remaining time (in seconds) before the next configured preferred host timer will be triggered. See Section 6.4.2.2.

Table 106: RsslReactorChannelInfo Structure Members

6.12.3 rsslReactorIoctl Option Values

There are currently no **RsslReactor** or **RsslReactorChannel** specific codes for use with the **rsslReactorIoctl**. Reactor-specific codes may be added in the future. The application can still use any of the codes allowed with **rsslReactorIoctl**, which are documented in the Enterprise Transport API C Edition *Developers Guide*.

6.12.4 Reactor Debug Functions

The following table describes the Reactor debug functions.

FUNCTION NAME	DESCRIPTION
<code>rsslReactorGetDebugLevel</code>	Gets the level of the debug information.
<code>rsslReactorSetDebugLevel</code>	Sets the level of the debug information.

FUNCTION NAME	DESCRIPTION
rsslReactorGetDebugInfo	<p>Returns the accumulated debug information as a byte array written from the debug enabling till this function call. Empty debug information array will be returned in case if no debug information was written.</p> <p>If debugging was at some point enabled and this function has not been previously called, the array will contain the previously logged messages. If this function was not called frequently enough or the debug buffer was not set big enough (see Section 6.2.1.4), this may cause incomplete debugging information due to the debug buffer overflow.</p>

7 Consuming Data from the Cloud

7.1 Overview

You can use the Enterprise Transport API to consume data from a cloud-based LSEG Real-Time Advanced Distribution Server. The API interacts with cloud-based servers using the following workflows:

- Credential Management (for details, refer to Section 7.3)
- Service Discovery (for details, refer to Section 7.6)
- Consuming Market Data (for details, refer to Section 7.7)
- Login Reissue (for details, refer to Section 7.4.3)

There are two versions of login credentials for the Delivery Platform:

- Version 1 Authentication also known as “V1 auth”, “OAuthPasswordGrant” or “V1 Password Credentials”: Uses the OAuth2.0 Password grant or Refresh Token grant. Requires a Machine Account consisting of username and password; also requires a client ID generated by the LSEG **AppGenerator**. For details, refer to Section 7.4.
- Version 2 Authentication also known as “V2 auth”, “OAuthClientCredentials” or “V2 Client Credentials”: Uses OAuth2.0 Client Credentials grant to obtain an access token. Requires a Service Account consisting of client ID and client Secret. For details, refer to Section 7.5.

NOTE: Version 2 Authentication is available as an **Early Access** feature to API developers to preview changes required to use this new authentication mechanism. Please note that the ability to setup Service Accounts to use this authentication is forthcoming.

The Enterprise Transport API will determine which authentication version to use based on the inputs. By default, for cloud connections the Enterprise Transport API connects to a server in the **us-east-1** cloud location.

For further details on Real-Time as it functions in the cloud, refer to the *Real-Time — Optimized: Installation and Configuration for Client Use*.

7.2 Encrypted Connections

When connecting to an LSEG Real-Time Advanced Distribution Server in the cloud, you must use an encrypted connection type (for details on connection types, refer to the *ETA C Developer Guide*).

Encrypted connections to the cloud must use an OpenSSL-based connection type (on both Windows and Linux). WinINet is not supported for cloud connectivity.

7.3 Credential Management

By default, the Enterprise Transport API will store all credential information. In order to use secure credential storage, a callback function can be specified by the user. If a callback function is specified, credentials are not stored in API; instead, application is called back whenever credentials are required.

When configuring the **RsslReactorChannel**, if **ReactorOAuthCredential.ReactorOAuthCredentialEventCallback** is specified, the API will call the user back whenever credentials are required. This callback must call **Reactor.submitOAuthCredentialRenewal** to submit the updated credentials.

7.4 Version 1 Authentication Using OAuth Password and Refresh_Token

7.4.1 Client_ID (AppKey)

To connect to Real-Time - Optimized infrastructure, the Enterprise Transport API requires a **Client_ID**, and optionally can include a client secret. **Client_IDs** are generated using **AppGenerator**, which refers to the **Client_ID** as an AppKey. Each user must obtain their unique **Client_ID** using the machine account email sent by LSEG, which includes a link to **AppGenerator**. Keep your **Client_ID** private: do not share **Client_IDs**.

- For further details on generating this ID, refer to the *Real-Time - Optimized: Installation and Configuration for Client Use* document. Each **Client_ID** is unique: do not share it with others.
- For details on how OAuth uses a Client Secret with a Client ID and their relationship, refer to OAuth documentation at: the following URL: <https://www.oauth.com/oauth2-servers/client-registration/client-id-secret/>.

7.4.2 Obtaining Initial Access and Refresh Tokens

To obtain an access token, the RTSDK API sends its username, **Client_ID** (from **RsslReactorOAuthCredential** as described in Section 6.10.2.1), and password (defined in the Login Domain, as described in Section 8.3) in a single message to the Delivery Platform. You must configure these details before executing a connect (for details on the **rsslReactorConnect** function, refer to Section 6.4.1.1).

In response, the Delivery Platform sends an access token, its expiration timeout (by default: 300 seconds), and a refresh token for use in the login reissue process (for details on the expiration timeout and login reissue process, refer to Section 7.4.3). The API must obtain an access token before executing a service discovery or obtaining market data.

The following diagram illustrates the process by which the RTSDK API obtains its tokens:

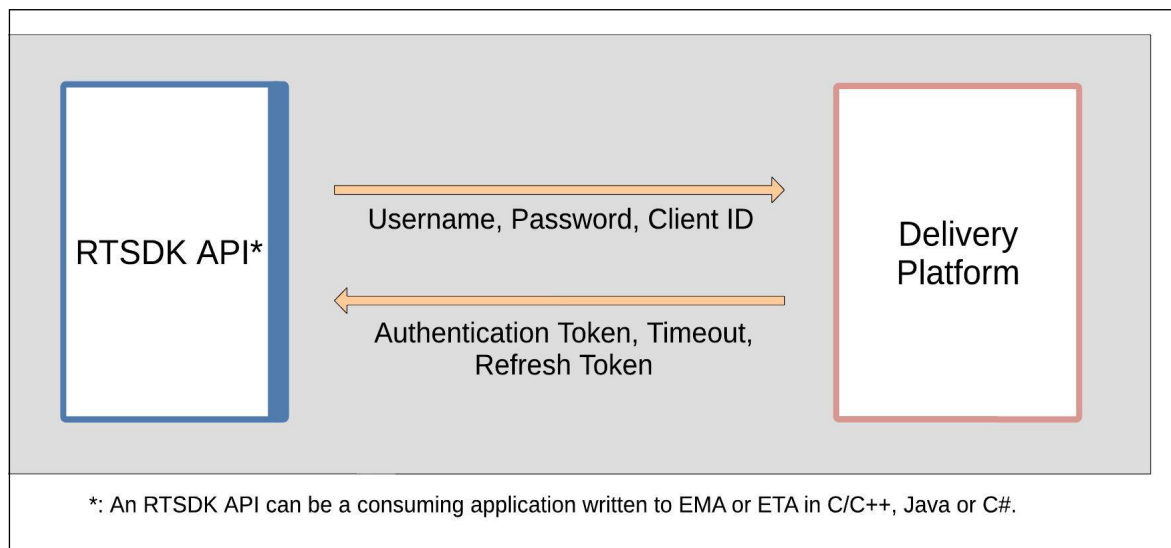


Figure 9. Obtaining an Authentication Token

7.4.3 Refreshing the Access Token and Sending a Login Reissue

In response to the API's token request, the Delivery Platform sends an access token and a refresh token, both with associated expiration timeouts which set the length of time for which the token is valid. If the LSEG Real-Time Advanced Distribution Server does not receive a new access token before the end of the expiration timeout, the LSEG Real-Time Advanced Distribution Server sends a login close status message and closes the connection.

To create a seamless experience for API users, the API sends the refresh token to proactively obtain a new access token prior to the published expiration timeout. The Enterprise Transport API calculates the time at which it requests a new access token by multiplying the token's published timeout by 4/5 (i.e., **0.8**). Thus, if the default is 300 seconds, the API requests a new access token after 240 seconds. You can configure this reissue ratio using `RsslCreateReactorOptions.tokenReissueRatio` (for details, refer to Section 6.2.1.2).

In response to receiving a refresh token, the Delivery Platform sends a new access token with an associated timeout to the API. After receiving the new access token from the Delivery Platform, the API renews its connection by sending a Login Reissue with the new access token to the LSEG Real-Time Advanced Distribution Server. The process of renewing the access token and refreshing the LSEG Real-Time Advanced Distribution Server connection via a Login Reissue continues until the refresh token itself expires (which can take several hours or days). When using a **grant_type** of **refresh_token**, if the value for **expires_in** does not match the **expires_in** received from when the API obtained the **refresh_token** (i.e., when **grant_type** was **password**), this is an indication that the **refresh_token** is about to expire. In this case, the API will obtain a new set of both refresh and access tokens as described in Section 7.4.2.

The login reissue process is illustrated in the following diagram:

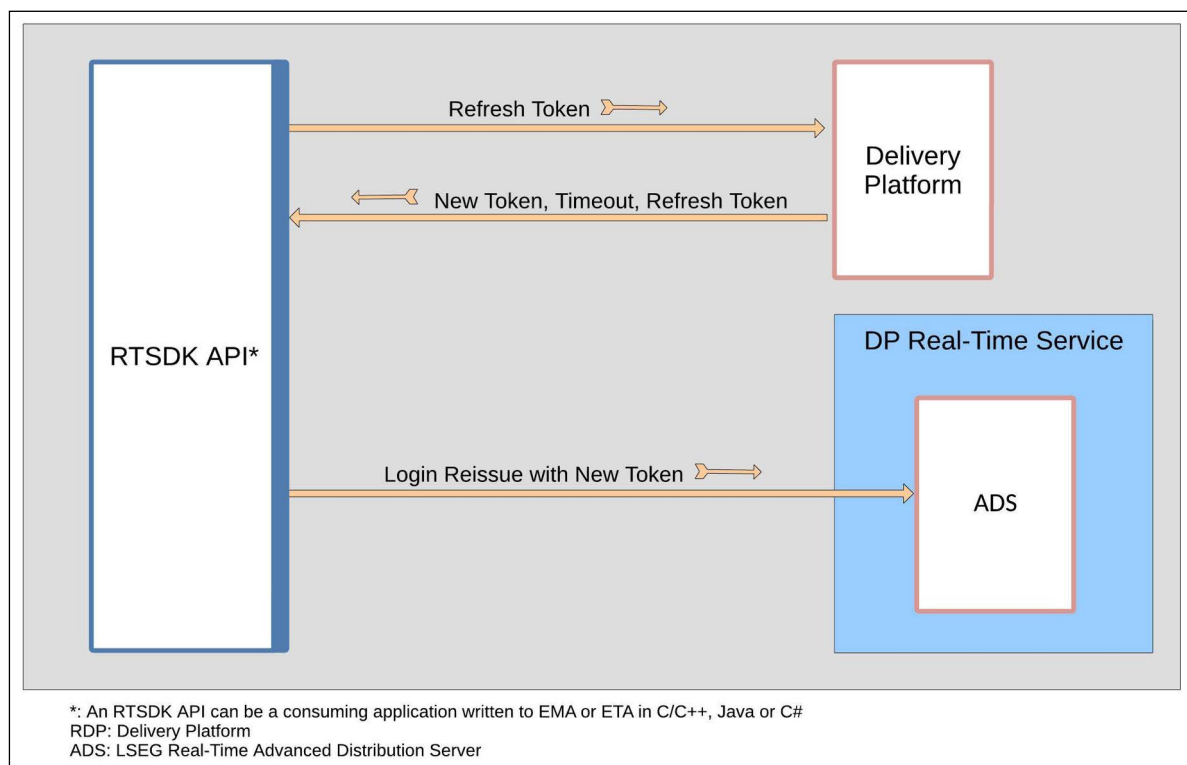


Figure 10. Login Reissue

7.4.4 Managing the Password and Client Secret

For security purposes, you can now configure whether the Enterprise Transport API reactor stores the password and client secret (used with username and Client ID to obtain the access and refresh tokens). By default, the Enterprise Transport API stores them both.

If you configure the Enterprise Transport API reactor to not store the password and client secret, whenever the Enterprise Transport API needs these credentials (i.e., when obtaining an initial access token or new refresh token), the API sends the **RsslReactorOAuthCredentialEvent** callback to the application. For details on the **RsslReactorOAuthCredentialEvent** callback, refer to Section 6.10.2.2.

After receiving the **RsslReactorOAuthCredentialEvent** callback, the application should send an **RsslReactorOAuthCredentialRenewal**, with the needed information, using the **rsslReactorSubmitOAuthCredentialRenewal** method.

- For details on **RsslReactorOAuthCredentialRenewal**, refer to Section 6.10.2.2.
- For details on the **rsslReactorSubmitOAuthCredentialRenewal** method, refer to Section 6.10.2.3.



TIP: The application can use the **rsslReactorSubmitOAuthCredentialRenewal** method to change its password on the fly.

7.4.5 Session Management per User Credential

Prior to Version 3.3.1, the Enterprise Transport API would manage tokens separately across each channel, even when using the same Username, Client ID, and password credentials. So that each channel had a unique pair of access and refresh tokens. API would manage each channel distinct from the others.

As of Version 3.3.1, the Enterprise Transport API connects to the Delivery Platform once and reuses the same access and refresh tokens for all channels. The Enterprise Transport API supports up to, but no more than, 5 channels per OAuth credential set.

7.5 Version 2 Authentication Using OAuth Client Credentials

Version 2 OAuth Client Credentials requires a client ID and client secret, or private JWK for JWT, or a client ID and private client JWK for OAuth Client Credentials with JWT. Version 2 will only generate an Access Token, not both Access and Refresh Token.

Once connected to Real-Time — Optimized RTC, there is no need to renew the Access Token. The login session to the LSEG Real-Time Connector (RTC) will remain valid until the consumer disconnects or is disconnected from Real-Time — Optimized. The API will only re-request an Access Token in the following cases:

- When the consumer disconnects and goes into a reconnection state.
- If the **reactorChannel** stays in reconnection long enough to get close to the expiry time of the Access Token.

Due to the above changes, credentials are managed independently per reactor channel. Channels do not share credentials.

7.5.1 Configuring and Managing Version 2 Credentials

The client ID and client secret or private JWK must be set on the **RsslReactorOAuthCredential** as described in Section 6.10.2.1 of the *Enterprise Transport API C Edition Value Added Developers Guide*. The **RsslReactorI** will handle the credentials the same way as Version 1, with an **IRsslReactorOAuthCredentialEvent** callback for credentials if the user does not wish for the **RsslReactorI** to store them.

7.5.1.1 JWT Credentials Handling

Version 2 OAuth Client Credentials with JWT requires a JWK public/private pair to be generated and registered with LSEG via the Platform Admin UI. The API will use a private JWK to create and sign a JWT request, which will be sent to retrieve an access token. The JWK will be handled by the API the exact same way as a client secret above. For more information about the Platform Admin UI, refer to the Real-Time — Optimized documentation in the LSEG Developers portal.

NOTE: Follow best practices for securely storing and retrieving JWK.

7.5.2 Version 2 OAuth Client Credentials Token Lifespan

Unlike Version 1, Version 2 will only produce a single Access Token, which will be valid for the length of the entire **expires_in** field in the token. This Access Token is used by the API to perform service discovery, and to connect to Real-Time — Optimized.

Once connected, the API does not need to periodically renew a token.

The API will re-request a token on reconnect, and will use that token for all reconnect attempts until a short time prior to expiry. At that time, the API will get a new token for reconnection use.

7.6 Service Discovery

After obtaining a token (for details, refer to Section 7.4.2), the Enterprise Transport API can perform a service discovery against the Delivery Platform to obtain connection details for the Real-Time — Optimized. The Enterprise Transport API C Edition uses the **rsslReactorQueryServiceDiscovery** function to submit a service discovery (refer to Section 6.2.1 for a description of this reactor method).

NOTE: The **rsslReactorQueryServiceDiscovery** function supports both blocking and non-blocking for authentication and service discovery requests. For details, refer to Section 6.10.1.2.

In response to a service discovery, the Delivery Platform returns transport and data format protocols and a list of hosts and associated ports for the requested service(s) (i.e., an LSEG Real-Time Advanced Distribution Server running in the cloud or endpoint). LSEG provides multiple cloud locations based on region, which is significant in how the Enterprise Transport API chooses the IP address and port to use when connecting to the cloud.

From the list sent by the Delivery Platform, the Enterprise Transport API identifies a Real-Time — Optimized endpoint with built-in resiliency whose regional location matches the API's location setting in **RsslReactorConnectInfo** (for details, refer to Section 6.4.1.3). If you do not specify a location, the Enterprise Transport API defaults to the **us-east-1** cloud location. An endpoint with built-in resiliency lists multiple locations in its location field (e.g., **location: [us-east-1a, us-east-1b]**). If multiple endpoints are configured for failover, the Enterprise Transport API chooses to connect to the first endpoint listed.

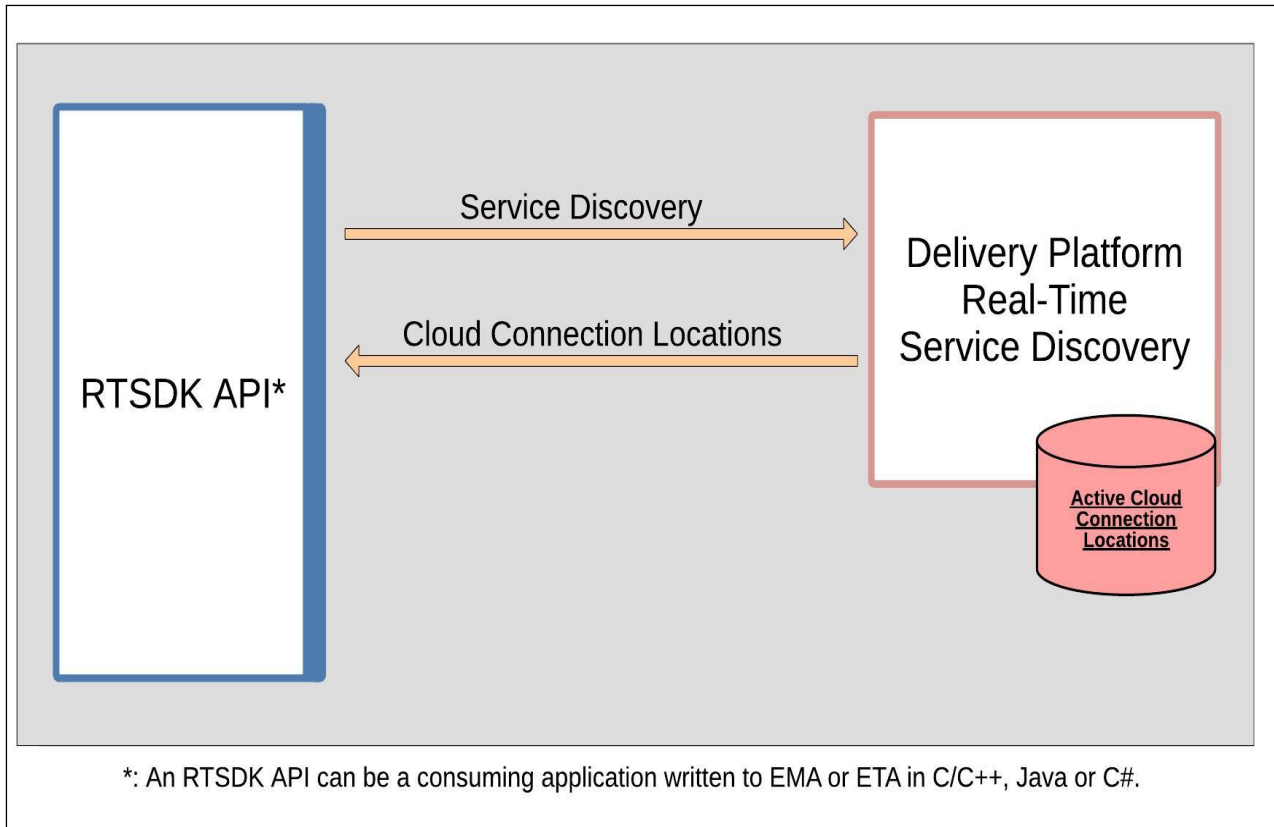
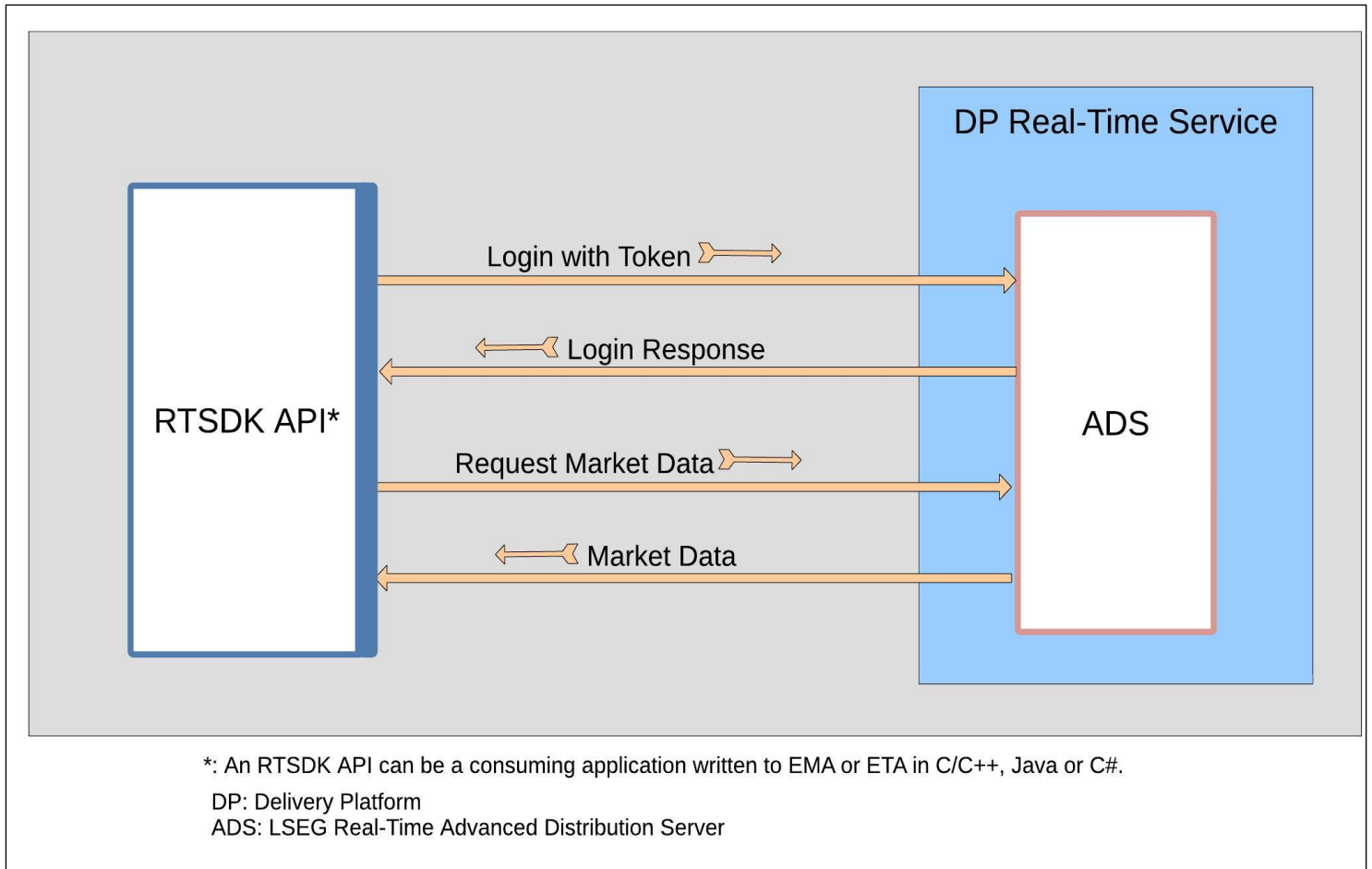


Figure 11. Service Discovery

7.7 Consuming Market Data

After obtaining its login token (for details, refer to Section 7.4.2) and running a service discovery (for details, refer to Section 7.6), the API can connect to the LSEG Real-Time Advanced Distribution Server in the cloud and obtain market data. While consuming market data, the API must periodically renew its token via the login reissue workflow (for details, refer to Section 7.4.3).



7.8 HTTP Error Handling for Reactor Token Reissues

The Enterprise Transport API supports handling for the following HTTP error codes from the API gateway:

- 300 Errors:
 - Perform URL redirect for 301, 302, 307 and 308 error codes
 - Retry the request to the API gateway for all other error codes
- 400 Errors:
 - For Version 1 authentication, retry with username and password for error codes 400 and 401
 - Stop retry the request for error codes 403, 404, 410, and 451
 - Retry the request to the API gateway for all other error codes
- 500 Errors:
 - Retry the request to the API gateway for all error codes

7.9 Cloud Connection Use Cases

You can connect to the cloud and consume data according to the following use cases:

- Start to finish session management using Reactor watchlist (for details, refer to Section 7.9.1)

- Session management using Reactor with watchlist disabled (for details, refer to Section 7.9.2Section 6.8.2)
- Explicit service discovery use case (for details, refer to Section 7.9.3)

7.9.1 Session Management Using Reactor Watchlist

In this use case, when using Enterprise Transport API Reactor with watchlist enabled in conjunction with session management, the API manages the entire connection from start to finish. To enable session management, set **EnableSessionManagement** in **ReactorConnectInfo**.

The API exhibits the following behavior when enabling both session management and watchlist features:

1. Obtains a token and renews access token as needed (according to the details in Section 7.4 and Section 7.5).
2. Performs an implicit service discovery and chooses an endpoint based on specified or defaulted location (us-east-1) to connect to (according to the details in Section 7.6).
3. Consumes market data from either discovered or specified endpoint (according to the details in Section 7.7).
4. Manages login reissues for Version 1 authentication when needed on a cyclical basis (according to the details in Section 7.4.3).

Possible ways to use session management with watchlist enabled include:

- By default, a default location is used to do implicit service discovery as detailed above and a LSEG resilient endpoint is automatically chosen for connectivity as detailed above. See Section 7.5 for details on service discovery.
- Application may specify a location to override default location. An LSEG resilient endpoint is automatically chosen from the specified region.
- Application may also choose to not consider results of automatic implicit service discovery by setting host and port to connect to. In this case, service discovery still occurs, however, results are ignored. Host and port information must be set in **ReactorConnectInfo** instance from **ReactorConnectOptions.ConnectionList** list.

7.9.2 Session Management Using Reactor with Watchlist Disabled

When connecting to an LSEG Real-Time Advanced Distribution Server in the cloud with the watchlist disabled (the default), the API:

- Obtains a token (according to the details in Section 7.4.2)
- If needed, queries service discovery (according to the details in Section 7.6)

If using **pOAuthCredential**, the application manually logs in with the token and manages the login reissues, otherwise the Reactor initially handles the RDM Login request, with the application handling subsequent Login Reissues using renewed access tokens. For details on **pOAuthCredential**, refer to Section 6.3.2.1.

To support this use case, you must configure session management (i.e., in **RsslReactorConnectInfo** objects, set **enableSessionManagement**).

7.9.3 Explicit Service Discovery Use Case

Application has the option to do a service discovery, parse the results, and choose an endpoint to pass into API. Depending on whether or not the watchlist is enabled, API will behave according to use cases already described in this section to manage session:

1. Obtains a token (according to the details in Section 7.4.2).
2. Queries service discovery (according to the details in Section 7.6).

7.10 Logging of Authentication and Service Discovery Interaction

If needed, you can log interactions with the Delivery Platform. To enable logging, use the `RsslCreateReactorOptions: restEnableLog` and `restLogOutputStream` as described in Section 6.2.1.2.

7.10.1 Logged Request Information

With logging turned on in the fashion mentioned in Section 7.10, the Enterprise Transport API writes the following request information in the log:

```
Request:
- Time stamp
- The Name of the class and method that made the request
- Request method
- URI
- Request headers
- Proxy information (if used)
- Body of request as set of pairs parameter_name: parameter_value
```

NOTE: If the request contains parameters `password`, `newPassword`, or `client_secret`, the Enterprise Transport API uses a placeholder instead of the real value of the respective parameter (thus indicating that the value was present).

7.10.2 Logged Response Information

With logging turned on in the fashion mentioned in Section 7.10, the Enterprise Transport API writes the following response information in the log:

```
Response:
- Time stamp
- The Name of the class and method that received the response
- Response status code
- Response headers
- Body of response in string format
```

8 Administration Domain Models Detailed View

8.1 Concepts

Administration Domain Model Representations are RDM-specific representations of OMM administrative domain models. This Value Added Component contains structures that represent messages within the Login, Source Directory, and Dictionary domains (discussed in Table 107). All structures follow the formatting and naming specified in the *Enterprise Transport API C Edition LSEG Domain Model Usage Guide*, so access to content is logical and specific to the content being represented. This component also handles all encoding and decoding functionality for these domain models, so the application needs only to manipulate the message's structure members to send or receive content. Such functionality significantly reduces the amount of code an application needs to interact with OMM devices (i.e., LSEG Real-Time Distribution System infrastructure), and also ensures that encoding/decoding for these domain models follow OMM-specified formatting rules. Applications can use this Value Added Component directly to help with encoding, decoding, and representation of these domain models. When using the Enterprise Transport API Reactor, this component is embedded to manage and present callbacks with a domain-specific representation of content.

Where possible, the members of an Administration Domain Model Representation structure are represented in the structure with the same **RsslDataType** that is specified for the element by the domain model. In cases where multiple elements are part of a more complex container such as an **RsslMap** or **RsslElemList**, the elements are represented with a C-style array with an associated count indicating the number of structures in the array.

The Enterprise Transport API C Edition *LSEG Domain Model Usage Guide* defines and describes all domain-specific behaviors, usage, and details.

DOMAIN	PURPOSE
Dictionary	Provides dictionaries that may be needed when decoding data. Though use of the Dictionary domain is optional, LSEG recommends that provider applications support the domain's use. The Dictionary domain is considered an administrative domain. Many LSEG components require this content and expect it to follow the domain model definition. For further details refer to Section 8.5.
Login	Authenticates users and advertises/requests features that are not specific to a particular domain. Use of and support for this domain is required for all OMM applications. Login is considered an administrative domain. Many LSEG components require this content and expect it to conform to the domain model definition. For further details refer to Section 8.3.
Source Directory	Advertises information about available services and their state, quality of service, and capabilities. This domain also conveys any group status and group merge information. Interactive and non-interactive provider applications require support for this domain. LSEG strongly recommends that consumers request this domain. Source Directory is considered an administrative domain, and many LSEG components expect this content and require it to conform to the domain model definition. For further details, refer to Section 8.4.

Table 107: Domains Representations in the Administration Domain Model Value Added Component

8.2 RDM Message Base

All Administration Domain Model Representation structures contain a common base structure that provides members common to all representations and identifies the specific message.

8.2.1 RSSL RDM Message Base Structure Members

All domain representation structures have several common members used for stream and domain identification. These are available in the **RsslRDMMsgBase** structure, as described in the following table.

STRUCTURE MEMBER	DESCRIPTION
streamId	Required. A unique signed-integer identifier associated with all messages flowing in the stream. <ul style="list-style-type: none"> Positive values indicate a consumer-instantiated stream, typically via a request message. Negative values indicate a provider-instantiated stream, often associated with Non-Interactive Providers.
domainType	Required. Identifies the specific domain message model type. If value is less than 128 , domain is a LSEG-defined domain model. If value is 128 - 255 , domain is a user defined domain model. Domain model definition is decoupled from the API and domain models are typically defined in some type of specification document. You can find more information on LSEG-defined domain models in the Enterprise Transport API C Edition <i>LSEG Domain Model Usage Guide</i> .
rdmMsgType	Required. Identifies the specific representation for a given domain. The currently supported rdmMsgTypes are defined in Section 8.2.2.

Table 108: RsslRDMMsgBase Structure Members

8.2.2 RSSL RDM Message Types

The following table provides a reference mapping between the administrative domain type and the structural representations provided in this component.

DOMAIN TYPE	RDM: MESSAGE TYPE	RDM: MESSAGE STRUCTURE
RSSL_DMT_LOGIN (RsslRDMLLoginMsg) Refer to Section 8.3	RDM_LG_MT_REQUEST	RsslRDMLLoginRequest
	RDM_LG_MT_REFRESH	RsslRDMLLoginRefresh
	RDM_LG_MT_STATUS	RsslRDMLLoginStatus
	RDM_LG_MT_CLOSE	RsslRDMLLoginClose
	RDM_LG_MT_CONSUMER_CONNECTION_STATUS	RsslRDMLLoginConsumerConnectionStatus
	LoginMsgType.RTT	LoginRTT

Table 109: RsslRDMMsg

DOMAIN TYPE	RDM: MESSAGE TYPE	RDM: MESSAGE STRUCTURE
RSSL_DMT_SOURCE (RsslRDMDirectoryMsg) Refer to Section 8.4	RDM_DR_MT_REQUEST	RsslRDMDirectoryRequest
	RDM_DR_MT_REFRESH	RsslRDMDirectoryRefresh
	RDM_DR_MT_UPDATE	RsslRDMDirectoryUpdate
	RDM_DR_MT_STATUS	RsslRDMDirectoryStatus
	RDM_DR_MT_CLOSE	RsslRDMDirectoryClose
	RDM_DR_MT_CONSUMER_STATUS	RsslRDMDirectoryConsumerStatus
RSSL_DMT_DICTIONARY (RsslRDMDictionaryMsg) Refer to Section 8.5	RDM_DC_MT_REQUEST	RsslRDMDictionaryRequest
	RDM_DC_MT_REFRESH	RsslRDMDictionaryRefresh
	RDM_DC_MT_STATUS	RsslRDMDictionaryStatus
	RDM_DC_MT_CLOSE	RsslRDMDictionaryClose

Table 109: RsslRDMMsg(Continued)

8.2.3 RSSL RDM Encoding and Decoding Functions

Encode and decode functionality is provided that can take the **RsslRDMMsg** union. This allows users to encode or decode from a general type that can represent any of the domain messages. Encode and decode functions are also provided for each specific domain type, as documented in the following chapters.

FUNCTION NAME	DESCRIPTION
rsslEncodeRDMMsg	Used to encode any message that the RsslRDMMsg can represent. This function takes the RsslRDMMsg as a parameter.
rsslDecodeRDMMsg	Used to decode any message that the RsslRDMMsg can represent. This function populates the RsslRDMMsg and leverages the Value Added Utility message buffer (refer to Section 11.2).
	NOTE: The decoded message may refer to encoded data from the original RsslMsg . If the message is to be stored, the appropriate copy function for the decoded RsslRDMMsg should be used to create a full copy.

Table 110: RDM Encoding and Decoding Functions

8.3 RDM Login Domain

The Login domain registers a user with the system, after which the user can request¹, post², or provide³ OMM content.

- A consumer application must log into the system before it can request or post content.
- A non-interactive provider (NIP) application must log into the system before providing content. An interactive provider application must handle login requests and provide login response messages, possibly using the Data Access Control System to authenticate users.

Section 8.3.1 - Section 8.3.10 detail the layout and use of each message structure in the Login portion of the Administration Domain Message Component.

-
1. Consumer applications can request content after logging into the system.
 2. Consumer applications can post content (similar to contributions or unmanaged publications) after logging into the system.
 3. Non-interactive provider applications.

8.3.1 RSSL RDM Login Request

A **Login Request** message is encoded and sent by OMM consumer and non-interactive provider applications. This message registers a user with the system. After receiving a successful login response, applications can then begin consuming or providing additional content. An OMM provider can use the login request information to authenticate users with the Data Access Control System.

The **RsslRDMLLoginRequest** represents all members of a login request message and allows for simplified use in OMM applications that leverage RDMs. This structure follows the behavior and layout that is defined in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.3.1.1 RSSL RDM Login Request Structure Members

STRUCTURE MEMBER	DESCRIPTION
allowSuspectData	<p>Optional. If present, a flags value of RDM_LG_RQF_HAS_ALLOW_SUSPECT_DATA should be specified. If absent, a default value of 1 is assumed.</p> <ul style="list-style-type: none"> 1: Indicates that the consumer application allows for suspect streamState information. 0: Indicates that the consumer application prefers any suspect data to result in the stream being closed with an RSSL_STREAM_CLOSED_RECOVER state.
applicationId	<p>Optional. If present, a flags value of RDM_LG_RQF_HAS_APPLICATION_ID should be specified.</p> <p>When populated, should contain the Data Access Control System applicationId. If the server authenticates with the Data Access Control System, the consumer application may be required to pass in a valid application id.</p> <p>If initializing RsslRDMLLoginRequest using rsslInitDefaultRDMLLoginRequest, an applicationId of 256 will be used.</p>
applicationName	<p>Optional. If present, a flags value of RDM_LG_RQF_HAS_APPLICATION_NAME should be specified.</p> <p>When present, the applicationName in the login request identifies the OMM consumer or non-interactive provider.</p> <p>If initializing RsslRDMLLoginRequest using rsslInitDefaultRDMLLoginRequest, applicationName is set to upa.</p>
authenticationExtended	<p>Optional. If present, a flags value of RDM_LG_RQF_HAS_AUTH_EXTENDED should be specified.</p> <p>When populated, authenticationExtended contains additional content that will be passed to the token authenticator as an additional means to verifying a user's identity.</p>
downloadConnectionConfig	<p>Optional. If present, a flags value of RDM_LG_RQF_HAS_DOWNLOAD_CONN_CONFIG should be specified. If absent, a default value of 0 is assumed.</p> <p>Enabling this option allows the application to download information about other providers on the network. You can use such downloaded information to load balance connections across multiple providers.</p> <ul style="list-style-type: none"> 1: Indicates that the user wants to download connection configuration information. 0: Indicates that the user does not want to download connection information.
flags	<p>Required. Indicates presence of optional login request members. For details, refer to Section 8.3.1.2.</p>
instanceId	<p>Optional. If present, a flags value of RDM_LG_RQF_HAS_INSTANCE_ID should be specified.</p> <p>You can use the instanceId to differentiate applications running on the same machine. However, because instanceId is set by the user logging into the system, it does not guarantee uniqueness across different applications on the same machine.</p>

Table 111: RsslRDMLLoginRequest Structure Members

STRUCTURE MEMBER	DESCRIPTION
password	Optional. If present, a flags value of RDM_LG_RQF_HAS_PASSWORD should be specified. When necessary, this should be set to the password for logging into the system. See specific component documentation to determine password requirements and how to obtain one.
position	Optional. If present, a flags value of RDM_LG_RQF_HAS_POSITION should be specified. When populated, should contain the Data Access Control System position . If the server is authenticating with the Data Access Control System, the consumer application might be required to pass in a valid position. If initializing RsslRDMLLoginRequest using rsslInitDefaultRDMLLoginRequest , the IP address of the system the application is running on will be used.
providePermissionExpressions	Optional. If present, a flags value of RDM_LG_RQF_HAS_PROVIDE_PERM_EXPR should be specified. If absent, a default value of 1 is assumed. When 1 , this indicates a consumer wants permission expression information to be sent with responses. Permission expressions allow for items to be proxy permissioned by a consumer via content-based entitlements.
providePermissionProfile	Optional. If present, a flags value of RDM_LG_RQF_HAS_PROVIDE_PERM_PROFILE should be specified. If not present, a default value of 1 is assumed. When 1 , this indicates that a consumer desires the permission profile. The permission profile can be used by an application to perform proxy permissioning.
rdmMsgBase	Required. Contains general message information like streamId and domainType . For more information, refer to Section 8.2.
role	Optional. If present, a flags value of RDM_LG_RQF_HAS_ROLE should be specified. If absent, a default value of RDM_LOGIN_ROLE_CONS is assumed. Indicates the role of the application logging onto the system. <ul style="list-style-type: none"> 0: RDM_LOGIN_ROLE_CONS, indicates application is a consumer. 1: RDM_LOGIN_ROLE_PROV, indicates application is a provider.
singleOpen	Optional. If present, a flags value of RDM_LG_RQF_HAS_SINGLE_OPEN should be specified. If absent, a default value of 1 is assumed. <ul style="list-style-type: none"> 1: Indicates the consumer application wants the provider to drive stream recovery. 0: Indicates that the consumer application will drive stream recovery.
supportProviderDictionaryDownload	Optional. If present, a flags value of RDM_LG_RQF_HAS_SUPPORT_PROV_DIC_DOWNLOAD should be specified. If absent, a default value of 0 is assumed. Indicates whether the LSEG Real-Time Advanced Distribution Hub supports the Provider Dictionary Download feature, which allows the application to request RWFFId and RWFEnum dictionaries from an LSEG Real-Time Advanced Distribution Hub. <ul style="list-style-type: none"> 1: The LSEG Real-Time Advanced Distribution Hub supports the Provider Dictionary Download feature. 0: The LSEG Real-Time Advanced Distribution Hub does not support the Provider Dictionary Download feature. For details on the Provider Dictionary Download feature, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .

Table 111: RsslRDMLLoginRequest Structure Members (Continued)

STRUCTURE MEMBER	DESCRIPTION
userName	Required. Populate this member with the username, email address, or user token based on the userNameType specification. If you initialize RsslRDMLLoginRequest using rsslInitDefaultRDMLLoginRequest , it uses the name of the user currently logged into the system on which the application runs.
userNameType	Optional. If present, a flags value of RDM_LG_RQF_HAS_USERNAME_TYPE should be specified. If absent, a default value of RDM_LOGIN_USER_NAME is assumed. Possible values: <ul style="list-style-type: none"> RDM_LOGIN_USER_NAME == 1 RDM_LOGIN_USER_EMAIL_ADDRESS == 2 RDM_LOGIN_USER_TOKEN == 3 RDM_LOGIN_USER_COOKIE == 4 RDM_LOGIN_USER_AUTHN_TOKEN==5 A type of RDM_LOGIN_USER_NAME typically corresponds to a Data Access Control System user name and can to authenticate and permission a user. RDM_LOGIN_USER_TOKEN is specified when using the AAA ('triple A') API. The user token is retrieved from the Authentication Manager application. To validate users, a provider application passes this user token to the AAA Gateway. This type of token periodically changes: when it changes, an application can send a login reissue to pass information upstream. For more information, refer to documentation specific to the AAA API. RDM_LOGIN_USER_AUTHN_TOKEN is specified when using LSEG Real-Time Distribution System Authentication. The authentication token should be specified in the userName member. This type of token can periodically change: when it changes, an application can send a login reissue to pass information upstream. For more information, refer to the <i>LSEG Real-Time Distribution System Authentication User Manual</i> . ^a

Table 111: RsslRDMLLoginRequest Structure Members (Continued)

a. For further details on LSEG Real-Time Distribution System Authentication, refer to the *LSEG Real-Time Distribution System Authentication User Manual*, accessible on [MyAccount](#) in the Data Access Control System product documentation set.

8.3.1.2 RSSL RDM Login Request Flag Enumeration Values

FLAG ENUMERATION	MEANING
RDM_LG_RQF_HAS_ALLOW_SUSPECT_DATA	Indicates the presence of allowSuspectData . If not present, a value of 1 should be assumed.
RDM_LG_RQF_HAS_APPLICATION_ID	Indicates the presence of applicationId .
RDM_LG_RQF_HAS_APPLICATION_NAME	Indicates the presence of applicationName .
RDM_LG_RQF_HAS_AUTHN_EXTENDED	Indicates the presence of authenticationExtended .
RDM_LG_RQF_HAS_DOWNLOAD_CONN_CONFIG	Indicates the presence of downloadConnectionConfig . If absent, a value of 0 should be assumed.
RDM_LG_RQF_HAS_INSTANCE_ID	Indicates the presence of instanceId .
RDM_LG_RQF_HAS_PASSWORD	Indicates the presence of password .
RDM_LG_RQF_HAS_POSITION	Indicates the presence of position .
RDM_LG_RQF_HAS_PROVIDE_PERM_EXPR	Indicates the presence of providePermissionExpressions . If not present, a value of 1 should be assumed.

Table 112: RsslRDMLLoginRequest Flags

FLAG ENUMERATION	MEANING
RDM_LG_RQF_HAS_PROVIDE_PERM_PROFILE	Indicates the presence of providePermissionProfile . If not present, a value of 1 should be assumed.
RDM_LG_RQF_HAS_ROLE	Indicates the presence of role . If absent, a role of RDM_LOGIN_ROLE_CONS is assumed.
RDM_LG_RQF_HAS_SINGLE_OPEN	Indicates the presence of singleOpen . If not present, a value of 1 should be assumed.
RDM_LG_RQF_HAS_SUPPORT_PROV_DIC_DOWNLOAD	Indicates the presence of supportProviderDictionaryDownload . If absent, a value of 0 should be assumed. For more information on Provider Dictionary Download, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
RDM_LG_RQF_HAS_USERNAME_TYPE	Indicates the presence of userNameType . If not present, a userNameType of RDM_LOGIN_USER_NAME should be assumed.
RDM_LG_RQF_NO_REFRESH	Indicates that the consumer application does not require a login refresh for this request. This typically occurs when resuming a stream or changing a AAA token. In some instances, a provider can still deliver a refresh message, however if such a message is not explicitly asked for by the consumer, it is considered unsolicited.
RDM_LG_RQF_PAUSE_ALL	Indicates that the consumer wants to pause all streams associated with the logged in user. For more information on pause and resume behavior, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
RDM_LG_RQF_RTT_SUPPORT	Indicates that the consumer supports the Round Trip Time feature. If this flag is set on a Consumer reactor Channel, the Reactor automatically replies with an Round Trip Time response. On the Login Callback Event, this is indicated by a RSSL_RDM_LG_LME_RTT_RESPONSE_SENT flag.

Table 112: RsslRDMLoginRequest Flags (Continued)

8.3.1.3 RSSL RDM Login Request Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearRDMLoginRequest	Clears an RsslRDMLoginRequest structure. Useful for structure reuse.
rsslInitDefaultRDMLoginRequest	Clears an RsslRDMLoginRequest structure and populates userName , position , applicationId , and applicationName with default values.
rsslCopyRDMLoginRequest	Performs a deep copy of an RsslRDMLoginRequest structure.

Table 113: RsslRDMLoginRequest Utility Functions

8.3.2 RSSL RDM Login Refresh

A **Login Refresh** message is encoded and sent by an OMM interactive provider application and responds to a Login Request message. A login refresh message indicates that the user's Login is accepted. A provider can use information from the login request to authenticate users with the Data Access Control System. After authentication, a refresh message is sent to convey that the login was accepted. If the login is rejected, a login status message should be sent as described in Section 8.3.3.

The **RsslRDMLoginRefresh** represents all members of a login refresh message and allows for simplified use in OMM applications that leverage RDMs. This structure follows the behavior and layout that is defined in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.3.2.1 Login Refresh Structure Members

STRUCTURE MEMBER	DESCRIPTION
allowSuspectData	Optional. If present, flags value of RDM_LG_RFF_HAS_ALLOW_SUSPECT_DATA should be specified. If absent, a default value of 1 is assumed. <ul style="list-style-type: none"> 1: Indicates that the consumer application allows for suspect streamState information. 0: Indicates that the consumer application prefers any suspect data to result in the stream being closed with an RSSL_STREAM_CLOSED_RECOVER state.
applicationId	Optional. If present, flags value of RDM_LG_RFF_HAS_APPLICATION_ID should be specified. When populated, this should match the applicationId contained in the login request.
applicationName	Optional. If present, flags value of RDM_LG_RFF_HAS_APPLICATION_NAME should be specified. When populated, the applicationName in the login refresh identifies the provider.
authenticationErrorCode	Optional. If present, a flags value of RDM_LG_RFF_HAS_AUTHN_ERROR_CODE should be specified. authenticationErrorCode is specific to an LSEG Real-Time Distribution System Authentication environment, where 0 indicates an error-free condition. For further information, refer to the <i>LSEG Real-Time Distribution System Authentication User Manual</i> . ^a
authenticationErrorText	Optional. If present, a flags value of RDM_LG_RFF_HAS_AUTHN_ERROR_TEXT should be specified. authenticationErrorText specifies any error text that accompanies an authenticationErrorCode . For further information, refer to the <i>LSEG Real-Time Distribution System Authentication User Manual</i> . ^a
authenticationExtendedResp	Optional. If present, a flags value of RDM_LG_RFF_HAS_AUTHN_EXTENDED_RESP should be specified. authenticationExtendedResp contains additional, customer-defined data associated with the authentication token sent in the original request. For further information, refer to the <i>LSEG Real-Time Distribution System Authentication User Manual</i> . ^a
authenticationTTReissue	Optional. If present, a flags value of RDM_LG_RFF_HAS_AUTHN_TT_REISSUE should be specified. Indicates when a new authentication token needs to be reissued (in UNIX Epoch time). For more information, refer to the <i>LSEG Real-Time Distribution System Authentication User Manual</i> . ^a
flags	Required. Indicate the presence of optional login refresh members. For details, see Section 8.3.2.2.

Table 114: RsslRDMLoginRefresh Structure Members

STRUCTURE MEMBER	DESCRIPTION
numStandbyServers	<p>Optional. If present, flags value of RDM_LG_RFF_HAS_CONN_CONFIG should be specified and the serverList member should also be specified. If not present, a default value of 0 is assumed.</p> <p>Indicates the number of servers in the serverList that the consumer is expected to use as standby servers when using Warm Standby functionality.</p>
position	<p>Optional. If present, flags value of RDM_LG_RFF_HAS_POSITION should be specified. When populated, this should match the position contained in the login request.</p>
providePermissionProfile	<p>Optional. If present, flags value of RDM_LG_RFF_HAS_PROVIDE_PERM_PROFILE should be specified. If absent, a default value of 1 is assumed.</p> <p>When 1, this indicates that the permission profile is provided. The permission profile can be used by an application to perform proxy permissioning.</p>
providePermissionExpressions	<p>Optional. If present, flags value of RDM_LG_RFF_HAS_PROVIDE_PERM_EXPR should be specified. If absent, a default value of 1 is assumed.</p> <p>When 1, this indicates a provider will provide permission expression information with responses. Permission expressions allow for items to be proxy permissioned by a consumer via content-based entitlements.</p>
rdmMsgBase	Required. Contains general message information like streamId and domainType. (i.e.,
sequenceNumber	<p>Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.</p>
serverCount	<p>Optional. If present, flags value of RDM_LG_RFF_HAS_CONN_CONFIG should be specified and the serverList member should also be specified. If not present, a default value of 0 is assumed.</p> <p>Indicates the number of servers present in the serverList parameter.</p>
serverList	<p>Optional. If present, flags value of RDM_LG_RFF_HAS_CONN_CONFIG should be specified and the serverCount and numStandbyServers members should also be specified.</p> <p>An array of servers that the consumer may connect to when using Warm Standby functionality.</p>
singleOpen	<p>Optional. If present, flags value of RDM_LG_RFF_HAS_SINGLE_OPEN should be specified. If absent, a default value of 1 is assumed.</p> <ul style="list-style-type: none"> • 1: Indicates the consumer application wants the provider to drive stream recovery. • 0: Indicates that the consumer application will drive stream recovery.
state	<p>Required. Indicates the state of the login stream.</p> <p>Defaults to a streamState of RSSL_STREAM_OPEN and a dataState of RSSL_DATA_OK. For more information on RsslState, refer to the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>
supportBatchRequests	<p>Optional. If present, flags value of RDM_LG_RFF_HAS_SUPP_BATCH should be specified. If absent, a default value of 0 is assumed.</p> <p>Indicates whether the provider supports batch functionality. Batch functionality allows a consumer to specify multiple items, all with matching attributes, in the same request message.</p> <ul style="list-style-type: none"> • 1: The provider supports batch requesting. • 0: The provider does not support batch requesting. <p>For more information on batch requesting, refer to the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>

Table 114: RsslRDMLoginRefresh Structure Members(Continued)

STRUCTURE MEMBER	DESCRIPTION
supportEnhancedSymbolList	<p>Optional. If present, a flags value of RDM_LG_RFF_HAS_SUPPORT_ENH_SL should be specified. If absent, a default value of 0x0 is assumed.</p> <p>Advertises, via flags, additional features that the provider supports for the Symbol List domain, such as providing data streams for the items present in a requested Symbol List item.</p> <ul style="list-style-type: none"> • 0x0: The provider does not support any Symbol List enhancements. • 0x1: The provider supports providing Symbol List data streams. <p>For more information on Symbol List behaviors, refer to the Enterprise Transport API C Edition <i>LSEG Domain Model Usage Guide</i>.</p>
supportOMMPost	<p>Optional. If present, flags value of RDM_LG_RFF_HAS_SUPP_POST should be specified. If absent, a default value of 0 is assumed.</p> <p>Indicates whether the provider supports OMM Posting:</p> <ul style="list-style-type: none"> • 1: The provider supports OMM Posting and the user is permissioned. • 0: The provider supports the OMM Post feature, but the user is not permissioned. • If this element is not present, then the server does not support OMM Post feature. <p>For more information on Posting, refer to the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>
supportOptimizedPauseResume	<p>Optional. If present, flags value of RDM_LG_RFF_HAS_SUPP_OPT_PAR should be specified. If not present, a default value of 0 is assumed.</p> <p>Indicates whether the provider supports Optimized Pause and Resume. Optimized Pause and Resume allows for pausing/resuming of individual item streams or pausing all item streams via a pause of the login stream.</p> <ul style="list-style-type: none"> • 1: The server supports optimized pause and resume. • 0: The server does not support optimized pause and resume. <p>For more information on Pause and Resume, refer to the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>
supportProviderDictionaryDownload	<p>Optional. If present, a flags value of RDM_LG_RFF_HAS_SUPPORT_PROV_DIC_DOWNLOAD should be specified. If absent, a default value of 0 is assumed.</p> <p>Indicates whether the LSEG Real-Time Advanced Distribution Hub supports the Provider Dictionary Download feature, which allows a user to request RWFFId and RWFEnum dictionaries.</p> <ul style="list-style-type: none"> • 1: The LSEG Real-Time Advanced Distribution Hub supports the Provider Dictionary Download feature. • 0: The LSEG Real-Time Advanced Distribution Hub does not support the Provider Dictionary Download feature. <p>For more information on Provider Dictionary Download, refer to the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>
supportStandby	<p>Optional. If present, flags value of RDM_LG_RFF_HAS_SUPP_STANDBY should be specified. If absent, a default value of 0 is assumed.</p> <p>Indicates whether the provider supports Warm Standby functionality. If supported, a provider can be told to run as an Active or a Standby server, where the Active will behave as usual. The Standby will respond to item requests only with the message header and will forward any state changing information. When informed that an Active server has failed, the Standby begins sending responses and becomes the 'Active' server.</p> <ul style="list-style-type: none"> • 1: The provider can support a role of Active or Standby in a Warm Standby group. • 0: The provider does not support warm standby functionality.

Table 114: Rss1RDMLginRefresh Structure Members(Continued)

STRUCTURE MEMBER	DESCRIPTION
supportViewRequests	<p>Optional. If present, flags value of RDM_LG_RFF_HAS_SUPP_VIEW should be specified. If absent, a default value of 0 is assumed.</p> <p>Indicates whether the provider supports Dynamic View functionality. A Dynamic View allows a user to request only the specific contents of the response information in which they are interested.</p> <ul style="list-style-type: none"> 1: The provider supports Dynamic View functionality. 0: The provider does not support Dynamic View functionality. <p>For more information on Dynamic View use, refer to the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>
userName	<p>Optional. If present, a flags value of RDM_LG_RFF_HAS_USERNAME should be specified. If populated, this should match the userName contained in the login request.</p>
userNameType	<p>Optional. If present, a flags value of RDM_LG_RFF_HAS_USERNAME_TYPE should be specified. If absent, a default value of RDM_LOGIN_USER_NAME is assumed.</p> <p>Possible values:</p> <ul style="list-style-type: none"> RDM_LOGIN_USER_NAME == 1 RDM_LOGIN_USER_EMAIL_ADDRESS == 2 RDM_LOGIN_USER_TOKEN == 3 RDM_LOGIN_USER_COOKIE==4 RDM_LOGIN_USER_AUTHN_TOKEN==5 <p>A type of RDM_LOGIN_USER_NAME typically corresponds to a Data Access Control System user name and can be used to authenticate and permission a user.</p> <p>RDM_LOGIN_USER_TOKEN is specified when using the AAA ('triple A') API. The user token is retrieved from the Authentication Manager application. To validate users, a provider application passes this user token to the AAA Gateway. This type of token periodically changes: when it changes, an application can send a login reissue to pass information upstream. For more information, refer to documentation specific to the AAA API.</p> <p>RDM_LOGIN_USER_AUTHN_TOKEN is specified when using LSEG Real-Time Distribution System Authentication. The authentication token should be specified in the userName member. This type of token can periodically change: when it changes, an application can send a login reissue to pass information upstream. For more information, refer to the <i>LSEG Real-Time Distribution System Authentication User Manual</i>.^a</p>

Table 114: Rss\RDMLginRefresh Structure Members(Continued)

a. For further details on LSEG Real-Time Distribution System Authentication, refer to the *LSEG Real-Time Distribution System Authentication User Manual*, accessible on [MyAccount](#) in the Data Access Control System product documentation set.

8.3.2.2 Login Refresh Flag Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_LG_RFF_CLEAR_CACHE	Indicates to clear stored payload information associated with the login stream. This might occur if some portion of data is known to be invalid.
RDM_LG_RFF_HAS_ALLOW_SUSPECT_DATA	Indicates the presence of allowSuspectData . If absent, a value of 1 should be assumed.
RDM_LG_RFF_HAS_APPLICATION_ID	Indicates the presence of applicationId .
RDM_LG_RFF_HAS_APPLICATION_NAME	Indicates the presence of applicationName .
RDM_LG_RFF_HAS_AUTHN_ERROR_CODE	Indicates the presence of authenticationErrorCode .

Table 115: Rss\RDMLginRefresh Flags

FLAG ENUMERATION	DESCRIPTION
RDM_LG_RFF_HAS_AUTHN_ERROR_TEXT	Indicates the presence of authenticationErrorText .
RDM_LG_RFF_HAS_AUTHN_EXTENDED_RESP	Indicates the presence of authenticationExtendedResp .
RDM_LG_RFF_HAS_AUTHN_TT_REISSUE	Indicates the presence of authenticationTTReissue .
RDM_LG_RFF_HAS_CONN_CONFIG	Indicates the presence of connection configuration information.
RDM_LG_RFF_HAS_POSITION	Indicates the presence of position .
RDM_LG_RFF_HAS_PROVIDE_PERM_EXPR	Indicates the presence of providePermissionExpressions . If absent, a value of 1 should be assumed.
RDM_LG_RFF_HAS_PROVIDE_PERM_PROFILE	Indicates the presence of providePermissionProfile . If absent, a value of 1 should be assumed.
RDM_LG_RFF_HAS_SEQ_NUM	Indicates the presence of numStandbyServers , serverCount , and serverList .
RDM_LG_RFF_HAS_SINGLE_OPEN	Indicates the presence of singleOpen . If absent, a value of 1 should be assumed.
RDM_LG_RFF_HAS_SUPP_BATCH	Indicates the presence of supportBatchRequests . If absent, a value of 0 should be assumed. For more information on Batch functionality, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
RDM_LG_RFF_HAS_SUPP_POST	Indicates the presence of supportOMMPost . If absent, a value of 0 should be assumed. For more information on Posting, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
RDM_LG_RFF_HAS_SUPPORT_PROV_DIC_DOWNLOAD	Indicates the presence of supportProviderDictionaryDownload . If absent, a value of 0 should be assumed. For more information on Provider Dictionary Download, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
RDM_LG_RFF_HAS_SUPP_OPT_PAR	Indicates the presence of supportOptimizedPauseResume . If absent, a value of 0 should be assumed. For more information on Pause and Resume, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
RDM_LG_RFF_HAS_SUPP_VIEW	Indicates the presence of supportViewRequests . If absent, a value of 0 should be assumed. For more information on View functionality, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
RDM_LG_RFF_HAS_SUPP_STANDBY	Indicates the presence of supportStandby . If absent, a value of 0 should be assumed.
RDM_LG_RFF_HAS_USERNAME	Indicates the presence of userName .
RDM_LG_RFF_HAS_USERNAME_TYPE	Indicates the presence of userNameType . If absent, a userNameType of RDM_LOGIN_USER_NAME should be assumed.
RDM_LG_RFF_SOLICITED	<ul style="list-style-type: none"> If present, this flag indicates that the login refresh is solicited (e.g., it is in response to a request). If this flag is absent, this refresh is unsolicited.

Table 115: Rss1RDMLoginRefresh Flags (Continued)

8.3.2.3 Login Refresh Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMLLoginRefresh</code>	Clears an RsslRDMLLoginRefresh structure. Useful for structure reuse.
<code>rsslCopyRDMLLoginRefresh</code>	Performs a deep copy of an RsslRDMLLoginRefresh structure.

Table 116: RsslRDMLLoginRefresh Utility Functions

8.3.2.4 Server Info Structure Members

STRUCTURE MEMBER	DESCRIPTION
<code>flags</code>	Required. Indicates the presence of optional server information members. For details, refer to Section 8.3.2.5.
<code>hostname</code>	Required. Indicates the server's hostname .
<code>loadFactor</code>	Optional. Indicates the load information for this server. If present, a flags value of RDM_LG_SIF_HAS_LOAD_FACTOR should be specified.
<code>port</code>	Required. Indicates the server's port number for connections.
<code>serverIndex</code>	Required. Provides the index value to this server.
<code>serverType</code>	Optional. Indicates whether this server is an active or standby server. If present, a flags value of RDM_LG_SIF_HAS_TYPE should be specified, populated by RDMLLoginServerTypes .

Table 117: RsslRDMServerInfo Structure Members

8.3.2.5 Server Info Flag Enumeration Values

FLAG ENUMERATION	DESCRIPTION
<code>RDM_LG_SIF_HAS_LOAD_FACTOR</code>	Indicates presence of loadFactor information.
<code>RDM_LG_SIF_HAS_TYPE</code>	Indicates presence of serverType .

Table 118: RsslRDMServerInfo Flags

8.3.2.6 Server Info Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMServerInfo</code>	Clears an RsslRDMServerInfo structure. Useful for structure reuse.

Table 119: RsslRDMServerInfo Utility Functions

8.3.3 RSSL RDM Login Status

OMM provider and non-interactive provider applications use the **Login Status** message to convey state information associated with the login stream. Such state information can indicate that a login stream cannot be established or to inform a consumer of a state change associated with an open login stream.

The login status message can also reject a login request or close an existing login stream. When a status message closes a login stream, any other open streams associated with the user are also closed.

The **RsslRDMLoginStatus** represents all members of a login status message and allows for simplified use in OMM applications that leverage RDMs. This structure follows the behavior and layout defined in the Enterprise Transport API C Edition *LSEG Domain Models Usage Guide*.

8.3.3.1 Login Status Structure Members

STRUCTURE MEMBER	DESCRIPTION
authenticationErrorCode	Optional. If present, a flags value of RDM_LG_STF_HAS_AUTHN_ERROR_CODE should be specified. authenticationErrorCode is specific to deployments using LSEG Real-Time Distribution System Authentication, and specifies an error code. A code of 0 indicates no error condition. For further information, refer to the <i>LSEG Real-Time Distribution System Authentication User Manual</i> . ^a
authenticationErrorText	Optional. If present, a flags value of RDM_LG_STF_HAS_AUTHN_ERROR_TEXT should be specified. Specifies any text associated with the specified authenticationErrorCode . For further information, refer to the <i>LSEG Real-Time Distribution System Authentication User Manual</i> . ^a
flags	Required. Indicates the presence of optional login status members. For details, refer to Section 8.3.3.2.
rdmMsgBase	Required. Contains general message information, such as streamId and domainType .
state	Optional. If present, a flags value of RDM_LG_STF_HAS_STATE should be specified. Indicates the state of the login stream. When rejecting a login the state should be: <ul style="list-style-type: none"> streamState = RSSL_STREAM_CLOSED or RSSL_STREAM_CLOSED_RECOVER dataState = RSSL_DATA_SUSPECT stateCode = RSSL_SC_NOT_ENTITLED For more information on RsslState , refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .

Table 120: RsslRDMLoginStatus Structure Members

STRUCTURE MEMBER	DESCRIPTION
userNameType	<p>Optional. If present, a flags value of RDM_LG_STF_HAS_USERNAME_TYPE should be specified. If absent, a default value of RDM_LOGIN_USER_NAME is assumed.</p> <p>Possible values:</p> <ul style="list-style-type: none"> • RDM_LOGIN_USER_NAME == 1 • RDM_LOGIN_USER_EMAIL_ADDRESS == 2 • RDM_LOGIN_USER_TOKEN == 3 • RDM_LOGIN_USER_COOKIE == 4 • RDM_LOGIN_USER_AUTHN_TOKEN == 5 <p>A type of RDM_LOGIN_USER_NAME typically corresponds to a Data Access Control System user name and can be used to authenticate and permission a user.</p> <p>RDM_LOGIN_USER_TOKEN is specified when using the AAA ('triple A') API. The user token is retrieved from the Authentication Manager application. To validate users, a provider application passes this user token to the AAA Gateway. This type of token periodically changes: when it changes, an application can send a login reissue to pass information upstream. For more information, refer to documentation specific to the AAA API.</p> <p>RDM_LOGIN_USER_AUTHN_TOKEN is specified when using LSEG Real-Time Distribution System Authentication. The authentication token should be specified in the userName member. This type of token can periodically change: when it changes, an application can send a login reissue to pass information upstream. For more information, refer to the <i>LSEG Real-Time Distribution System Authentication User Manual</i>.^a</p>
userName	<p>Optional. If present, a flags value of RDM_LG_STF_HAS_USERNAME should be specified.</p> <p>When populated, this should match the userName in the login request.</p>

Table 120: Rss1RDMLoginStatus Structure Members (Continued)

a. For further details on LSEG Real-Time Distribution System Authentication, refer to the *LSEG Real-Time Distribution System Authentication User Manual*, accessible on [MyAccount](#) in the Data Access Control System product documentation set.

8.3.3.2 Login Status Flag Enumeration Values

FLAG ENUMERATION	MEANING
RDM_LG_STF_CLEAR_CACHE	Indicates whether the receiver of the login status should clear any associated cache information.
RDM_LG_STF_HAS_AUTHN_ERROR_CODE	Indicates the presence of authenticationErrorCode .
RDM_LG_STF_HAS_AUTHN_ERROR_TEXT	Indicates the presence of authenticationErrorText .
RDM_LG_STF_HAS_STATE	Indicates the presence of state . If absent, any previously conveyed state continues to apply.
RDM_LG_STF_HAS_USERNAME	Indicates the presence of userName .
RDM_LG_STF_HAS_USERNAME_TYPE	Indicates the presence of userNameType . If absent a userNameType of RDM_LOGIN_USER_NAME is assumed.

Table 121: Rss1RDMLoginStatus Flags

8.3.3.3 Login Status Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMLLoginStatus</code>	Clears an RsslRDMLLoginStatus structure. Useful for structure reuse.
<code>rsslCopyRDMLLoginStatus</code>	Performs a deep copy of an RsslRDMLLoginStatus structure.

Table 122: **RsslRDMLLoginStatus** Utility Functions

8.3.4 RSSL RDM Login Close

A **Login Close** message is encoded and sent by OMM consumer applications. This message allows a consumer to log out of the system. Closing a login stream is equivalent to a **Close All** type of message, where all open streams are closed (i.e., all streams associated with the user). A provider can log off a user and close all of that user's streams via a login status message, see Section 8.3.3.

8.3.4.1 Login Close Structure Member

STRUCTURE MEMBER	DESCRIPTION
<code>rdmMsgBase</code>	Contains general message information like streamId and domainType .

Table 123: **RsslRDMLLoginClose** Structure Member

8.3.4.2 Login Close Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMLLoginClose</code>	Clears an RsslRDMLLoginClose structure. Useful for structure reuse.
<code>rsslCopyRDMLLoginClose</code>	Performs a deep copy of an RsslRDMLLoginClose structure.

Table 124: **RsslRDMLLoginClose** Utility Functions

8.3.5 RSSL RDM Consumer Connection Status

The **Login Consumer Connection Status** informs an interactive provider of its role in a **Warm Standby** group, either as an **Active** or **Standby** provider. An active provider behaves normally; however a standby provider responds to requests only with a message header (allowing a consumer application to confirm the availability of requested data across active and standby servers), and forwards any state-related messages (i.e., unsolicited refresh messages, status messages). A standby provider aggregates changes to item streams whenever possible. If a provider changes from Standby to Active via this message, all aggregated update messages are passed along. If aggregation is not possible, a full, unsolicited refresh message is passed along.

The consumer application is responsible for ensuring that items are available and equivalent across all providers in a warm standby group. This includes managing state and availability differences as well as item group differences.

The **RsslRDMLLoginConsumerConnectionStatus** relies on the **RsslGenericMsg** and represents all members necessary for applications that leverage RDMS. This structure follows the behavior and layout that is defined in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.3.5.1 Login Consumer Connection Status Structure Members

STRUCTURE MEMBER	DESCRIPTION
flags	Required. Indicate the presence of optional login consumer connection status members. For details, refer to Section 8.3.5.2.
rdmMsgBase	Required. Contains general message information like streamId and domainType . Indicates the Login Message type (for login connection status, set to LoginMsgType.CONSUMER_CONNECTION_STATUS).
warmStandbyInfo	Optional. Includes RsslRDMLLoginWarmStandbyInfo to convey the state of the upstream provider. For details, refer to Section 8.3.5.3. If present, a flags value of RDM_LG_CCSF_HAS_WARM_STANDBY_INFO should be specified.

Table 125: RsslRDMLLoginConsumerConnectionStatus Structure Members

8.3.5.2 Login Consumer Connection Status Flag Enumeration Value

FLAG ENUMERATION	DESCRIPTION
RDM_LG_CCSF_HAS_WARM_STANDBY_INFO	Indicates presence of warmStandbyInfo .

Table 126: RsslRDMLLoginConsumerConnectionStatus Flags

8.3.5.3 Login Warm Standby Info Structure Members

STRUCTURE MEMBER	DESCRIPTION
action	Required. Indicates how a cache of Warm Standby content should apply this information. For information on RsslMapEntry actions, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
warmStandbyMode	Required. Indicates the presence of optional login consumer connection status members. For details, refer to Section 8.3.5.4.

Table 127: RsslRDMLLoginWarmStandbyInfo Structure Members

8.3.5.4 Login Warm Standby Mode Enumeration Values

ENUMERATION	DESCRIPTION
RDM_LOGIN_SERVER_TYPE_ACTIVE	Indicates that the server is acting as the active or primary server in a warm standby configuration.
RDM_LOGIN_SERVER_TYPE_STANDBY	Indicates that the server is acting as the standby or backup server in a warm standby configuration.

Table 128: RDMLLoginServerTypes Enumeration Values

8.3.5.5 Login Consumer Connection Status Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMLLoginConsumerConnectionStatus</code>	Clears an RsslRDMLLoginConsumerConnectionStatus structure. Useful for structure reuse.
<code>rsslClearRDMLLoginWarmStandbyInfo</code>	Clears the RsslRDMLLoginWarmStandbyInfo structure.
<code>rsslCopyRDMLLoginConsumerConnectionStatus</code>	Performs a deep copy of an RsslRDMLLoginConsumerConnectionStatus structure.

Table 129: RsslRDMLLoginConsumerConnectionStatus Utility Functions

8.3.6 Login Round Trip Time Message Use

Interactive Provider applications use Login Round Trip Time messages to measure the full roundtrip latency time between the provider and consumer. You enable Round Trip Time by setting the **RDM_LG_RQF_RTT_SUPPORT** flag on the initial Login RDM request message. When **RDM_LG_RQF_RTT_SUPPORT** is set on the consumer, the Reactor attempts to reflect the Round Trip Time message whenever the provider sends a Round Trip Time message, and the login callback will contain the incoming Round Trip Time message for informational purposes. On the Login Callback Event, this will be indicated by a **RSSL_RDM_LG_LME_RTT_RESPONSE_SENT** flag. The Consumer application does not need to take further action to handle Round Trip Time messages.

For more specific usage information about this message type, refer to the Enterprise Transport API C Edition *RDM Usage Guide*.

8.3.6.1 Login Round Trip Time Members

STRUCTURE MEMBER	DESCRIPTION
flags	Required. Indicates the presence of optional Round Trip Time members. For details, refer to Section 8.3.6.2.
rdmMsgBase	Required. Contains general message information, such as streamId and domainType .
lastLatency	Specifies the previous Round Trip Latency value (in microseconds) calculated by the provider.
tcpRetrans	Indicates the total number of TCP retransmissions.
ticks	Required. Specifies the tick count sent by the provider. After receiving ticks , the consumer must reflect this value back to the provider using this element.

Table 130: Login Round Trip Time Members

8.3.6.2 Login Round Trip Time Flag Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_LG_RTT_HAS_TCP_RETRANS	Indicates the presence of the tcpRetrans member.
RDM_LG_RTT_HAS_LATENCY	Indicates the presence of the lastLatency member.

Table 131: Login Round Trip Time Flag Enumeration Values

8.3.6.3 Login Round Trip Time Utility Functions

The Enterprise Transport API provides the following utility function for use with Login Round Trip Time Messages.

FUNCTION NAME	DESCRIPTION
rsslClearRDMRTT	Clears an RsslRDMLLoginRTT structure for reuse.
rsslCopyRDMRTT	Performs a deep copy of an RsslRDMLLoginRTT structure.

Table 132: Login Round Trip Time Utility Functions

8.3.7 Login Post Message Use

OMM consumer applications can encode and send data for any item via Post messages on the item's login stream. This is known as **off-stream posting** because items are posted without using that item's dedicated stream. Posting an item on its own dedicated stream is referred to as **on-stream posting**.

When an application is off-stream posting, **msgKey** information is required on the **RsslPostMsg**. For more details on posting, refer to the Enterprise Transport API C Edition *Developers Guide*.

8.3.8 Login Ack Message Use

OMM provider applications encode and send Ack messages to acknowledge the receipt of Post messages. An Ack message is used whenever a consumer posts and asks for acknowledgments. For more details on posting, see the Enterprise Transport API C Edition *Developers Guide*.

8.3.9 RSSL RDM Login Message Union

This union can contain any of the RDM Login message types. This is provided for use with Login specific functionality.

8.3.9.1 Login Union

UNION MEMBERS	DESCRIPTION
close	The RsslRDMLLoginClose as described in Section 8.3.4.
consumerConnectionStatus	The RsslRDMLLoginConsumerConnectionStatus as described in Section 8.3.5.
rdmMsgBase	The message base information.
refresh	The RsslRDMLLoginRefresh as described in Section 8.3.2.
request	The RsslRDMLLoginRequest as described in Section 8.3.1.
RTT	The RsslRDMLLoginRTT as described in Section 8.3.6.
status	The RsslRDMLLoginStatus as described in Section 8.3.3.

Table 133: RsslRDMLLoginMsg Union Members

8.3.9.2 Login Message Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearRDMLLoginMsg	Clears an RsslRDMLLoginMsg union. Useful for reuse.
rsslCopyRDMLLoginMsg	Performs a deep copy of an RsslRDMLLoginMsg structure.

Table 134: RsslRDMLLoginMsg Utility Functions

8.3.10 RSSL RDM Login Encoding and Decoding

8.3.10.1 Directory Login Encoding and Decoding Functions

FUNCTION NAME	DESCRIPTION
<code>rsslDecodeRDMLLoginMsg</code>	Decodes an RDM Login message. This function populates the RsslRDMLLoginMsg and leverages the Value Added Utility message buffer (refer to Section 11.2). Alternatively, rsslDecodeRDMMsg can be used to decode into an RsslRDMMsg .
<code>rsslEncodeRDMLLoginMsg</code>	Encodes an RDM Login message. This function takes the RsslRDMLLoginMsg as a parameter. Alternately, rsslEncodeRDMMsg can be used if encoding from an RsslRDMMsg .

Table 135: RDM Login Encoding and Decoding Functions

8.3.10.2 Encoding a Login Request

```

RsslEncodeIterator encodeIter;
RsslRDMLLoginRequest loginRequest;

/* Clear the Login Request structure. */
rsslClearRDMLLoginRequest(&loginRequest);

/* Set flags indicating presence of optional members. */
loginRequest.flags =
    RDM_LG_RQF_HAS_APPLICATION_NAME
    | RDM_LG_RQF_HAS_APPLICATION_ID
    | RDM_LG_RQF_HAS_POSITION;

/* Set UserName. */
loginRequest.userName.data = "username";
loginRequest.userName.length = 8;

/* Set ApplicationName */
loginRequest.applicationName.data = "upa";
loginRequest.applicationName.length = 3;

/* Set ApplicationId */
loginRequest.applicationId.data = "256";
loginRequest.applicationId.length = 3;

/* Set Position */
loginRequest.position.data = "127.0.0.1/net";
loginRequest.position.length = 13;

/* Clear the encode iterator, set its RWF Version, and set it to a buffer for encoding into. */
rsslClearEncodeIterator(&encodeIter);
ret = rsslSetEncodeIteratorRWFVersion(&encodeIter, channelMajorVersion, channelMinorVersion);
ret = rsslSetEncodeIteratorBuffer(&encodeIter, &msgBuffer);

/* Encode the message. */
ret = rsslEncodeRDMMsg(&encodeIter, (RsslRDMMsg*)&loginRequest, &msgBuffer.length, &rsslErrorInfo);

```

Code Example 19: Login Request Encoding Example

8.3.10.3 Decoding a Login Request

```

/* The decoder may require additional space to store things such as lists. */
char memoryArray[1024];
RsslBuffer memoryBuffer = { 1024, memoryArray };

RsslDecodeIterator decodeIter;
RsslMsg msg;

```

```

RsslRDMMsg rdmMsg;
RsslRDMLoginRequest *pLoginRequest;

/* Clear the decode iterator, set its RWF Version, and set it to the encoded buffer. */
rsslClearDecodeIterator(&decodeIter);
ret = rsslSetDecodeIteratorRWFVersion(&decodeIter, channelMajorVersion, channelMinorVersion);
ret = rsslSetDecodeIteratorBuffer(&decodeIter, &msgBuffer);

/* Decode the message to an RsslMsg structure and RsslRDMMsg structure. */
ret = rsslDecodeRDMMsg(&decodeIter, &msg, &rdmMsg, &memoryBuffer, &rsslErrorInfo);

if (ret == RSSL_RET_SUCCESS
    && rdmMsg.rdmMsgBase.domainType == RSSL_DMT_LOGIN && rdmMsg.rdmMsgBase.rdmMsgType ==
    RDM_LG_MT_REQUEST)
{
    /* The message we decoded is an RsslRDMLoginRequest. */
    pLoginRequest = &rdmMsg.loginMsg.request;

    /* Print username. */
    printf("Username: %.*s\n", pLoginRequest->userName.length, pLoginRequest->userName.data);

    /* Print ApplicationName if present. */
    if (pLoginRequest->flags & RDM_LG_RQF_HAS_APPLICATION_NAME)
        printf("ApplicationName: %.*s\n", pLoginRequest->applicationName.length, pLoginRequest->
            applicationName.data);

    /* Print ApplicationId if present. */
    if (pLoginRequest->flags & RDM_LG_RQF_HAS_APPLICATION_ID)
        printf("ApplicationId: %.*s\n", pLoginRequest->applicationId.length, pLoginRequest->
            applicationId.data);

    /* Print Position if present. */
    if (pLoginRequest->flags & RDM_LG_RQF_HAS_POSITION)
        printf("Position: %.*s\n", pLoginRequest->position.length, pLoginRequest->position.data);
}

```

Code Example 20: Login Request Decoding Example

8.3.10.4 Encoding a Login Refresh

```

RsslEncodeIterator encodeIter;
RsslRDMLLoginRefresh loginRefresh;

/* Clear the Login Refresh structure. */
rsslClearRDMLLoginRefresh(&loginRefresh);

/* Set flags indicating presence of optional members. */
loginRefresh.flags =
    RDM_LG_RFF_HAS_USERNAME
    | RDM_LG_RFF_HAS_APPLICATION_NAME
    | RDM_LG_RFF_HAS_APPLICATION_ID
    | RDM_LG_RFF_HAS_POSITION;

/* Set UserName(should match request). */
loginRefresh.userName.data = "username";
loginRefresh.userName.length = 8;

/* Set ApplicationName(should match request). */
loginRefresh.applicationName.data = "upa";
loginRefresh.applicationName.length = 3;

/* Set ApplicationId(should match request). */
loginRefresh.applicationId.data = "256";
loginRefresh.applicationId.length = 3;

/* Set Position(should match request). */
loginRefresh.position.data = "127.0.0.1/net";
loginRefresh.position.length = 13;

/* Clear the encode iterator, set its RWF Version, and set it to a buffer for encoding into. */
rsslClearEncodeIterator(&encodeIter);
ret = rsslSetEncodeIteratorRWFVersion(&encodeIter, channelMajorVersion, channelMinorVersion);
ret = rsslSetEncodeIteratorBuffer(&encodeIter, &msgBuffer);

/* Encode the message. */
ret = rsslEncodeRDMMsg(&encodeIter, (RsslRDMMsg*)&loginRefresh, &msgBuffer.length, &rsslErrorInfo);

```

Code Example 21: Login Refresh Encoding Example

8.3.10.5 Decoding a Login Refresh

```

/* The decoder may require additional space to store things such as lists. */
char memoryArray[1024];
RsslBuffer memoryBuffer = { 1024, memoryArray };

RsslDecodeIterator decodeIter;
RsslMsg msg;
RsslRDMMsg rdmMsg;
RsslRDMLLoginRefresh *pLoginRefresh;

/* Clear the decode iterator, set its RWF Version, and set it to the encoded buffer. */
rsslClearDecodeIterator(&decodeIter);
ret = rsslSetDecodeIteratorRWFVersion(&decodeIter, channelMajorVersion, channelMinorVersion);
ret = rsslSetDecodeIteratorBuffer(&decodeIter, &msgBuffer);

/* Decode the message to an RsslMsg structure and RsslRDMMsg structure. */
ret = rsslDecodeRDMMsg(&decodeIter, &msg, &rdmMsg, &memoryBuffer, &rsslErrorInfo);

if (ret == RSSL_RET_SUCCESS
    && rdmMsg.rdmMsgBase.domainType == RSSL_DMT_LOGIN && rdmMsg.rdmMsgBase.rdmMsgType ==
    RDM_LG_MT_REFRESH)
{
    /* The message we decoded is an RsslRDMLLoginRefresh. */
    pLoginRefresh = &rdmMsg.loginMsg.refresh;

    /* Print username if present. */
    if (pLoginRefresh->flags & RDM_LG_RFF_HAS_APPLICATION_NAME)
        printf("Username: %.*s\n", pLoginRefresh->userName.length, pLoginRefresh->userName.data);

    /* Print ApplicationName if present. */
    if (pLoginRefresh->flags & RDM_LG_RFF_HAS_APPLICATION_NAME)
        printf("ApplicationName: %.*s\n", pLoginRefresh->applicationName.length, pLoginRefresh->
            applicationName.data);

    /* Print ApplicationId if present. */
    if (pLoginRefresh->flags & RDM_LG_RFF_HAS_APPLICATION_ID)
        printf("ApplicationId: %.*s\n", pLoginRefresh->applicationId.length, pLoginRefresh->
            applicationId.data);

    /* Print Position if present. */
    if (pLoginRefresh->flags & RDM_LG_RFF_HAS_POSITION)
        printf("Position: %.*s\n", pLoginRefresh->position.length, pLoginRefresh->position.data);
}

```

Code Example 22: Login Refresh Decoding Example

8.4 RSSL RDM Source Directory Domain

The Source Directory domain model conveys information about:

- All available services and their capabilities, their supported domain types, services' states, quality of service, and item group information (associated with any particular service). Each service is associated with a unique **serviceId**.
- Item group status, allowing a single message to change the state of all associated items. Thus, using the Source Directory domain an application can send a mass update for multiple items instead of sending a status message for each individual item. The consumer is responsible for applying any changes to its open items. For details, refer to Section 8.4.10.
- Source Mirroring between an LSEG Real-Time Advanced Distribution Hub and OMM interactive provider applications. The Source Directory exchanges this information via a specifically-formatted generic message as described in Section 8.4.6.

8.4.1 RSSL RDM Directory Request

An OMM consumer application encodes and sends **Directory Request** messages to request information from an OMM provider about available services. A consumer may request information about all services by omitting the **serviceId** member, or request information about a specific service by setting it to the ID of the desired service.

The **RsslRDMDirectoryRequest** represents all members of a directory request message and is easily used in OMM applications that leverage RDMs. This structure follows the behavior and layout that is defined in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.4.1.1 Directory Request Structure Members

STRUCTURE MEMBER	DESCRIPTION
filter	Required. Indicates the service information in which the consumer is interested. The available flags are: <ul style="list-style-type: none"> • RDM_DIRECTORY_SERVICE_INFO_FILTER == 0x01 • RDM_DIRECTORY_SERVICE_STATE_FILTER == 0x02 • RDM_DIRECTORY_SERVICE_GROUP_FILTER == 0x04 • RDM_DIRECTORY_SERVICE_LOAD_FILTER == 0x08 • RDM_DIRECTORY_SERVICE_DATA_FILTER == 0x10 • RDM_DIRECTORY_SERVICE_LINK_FILTER == 0x20 In most cases, you should set the RDM_DIRECTORY_SERVICE_INFO_FILTER , RDM_DIRECTORY_SERVICE_STATE_FILTER , and RDM_DIRECTORY_SERVICE_GROUP_FILTER .
flags	Required. Indicates the presence of optional directory request members. For details, refer to Section 8.4.1.2.
rdmMsgBase	Required. Contains general message information like streamId and domainType .
serviceId	Optional. <ul style="list-style-type: none"> • If not present, this indicates the consumer wants information about all available services. • If present, this indicates the ID of the service about which the consumer wants information. Additionally, a flags value of RDM_DR_RQF_HAS_SERVICE_ID should be specified.

Table 136: RsslRDMDirectoryRequest Structure Members

8.4.1.2 Directory Request Flag Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_DR_RQF_HAS_SERVICE_ID	Indicates the presence of serviceId .
RDM_DR_RQF_STREAMING	Indicates that the consumer wants to receive updates about directory information after the initial refresh.

Table 137: RsslRDMDirectoryRequest Flags

8.4.1.3 Directory Request Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearRDMDirectoryRequest	Clears an RsslRDMDirectoryRequest structure. Useful for structure reuse.
rsslInitDefaultRDMDirectoryRequest	Clears an RsslRDMDirectoryRequest , sets the structure to request all services and receive updates for them, and populates filter with default values.
rsslCopyRDMDirectoryRequest	Performs a deep copy of an RsslRDMDirectoryRequest structure.

Table 138: RsslRDMDirectoryRequest Utility Functions

8.4.2 RSSL RDM Directory Refresh

A **Directory Refresh** message is encoded and sent by OMM provider and non-interactive provider applications. This message can provide information about the services supported by the provider application.

The **RsslRDMDirectoryRefresh** represents all members of a directory refresh message and is easily used in OMM applications that leverage RDMs. This structure follows the behavior and layout that is defined in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.4.2.1 Directory Refresh Structure Members

STRUCTURE MEMBER	DESCRIPTION
filter	Required. Indicates the information being provided about supported services. This should match the filter of the consumer's RsslRDMDirectoryRequest . The available flags are: <ul style="list-style-type: none"> RDM_DIRECTORY_SERVICE_INFO_FILTER == 0x01 RDM_DIRECTORY_SERVICE_STATE_FILTER == 0x02 RDM_DIRECTORY_SERVICE_GROUP_FILTER == 0x04 RDM_DIRECTORY_SERVICE_LOAD_FILTER == 0x08 RDM_DIRECTORY_SERVICE_DATA_FILTER == 0x10 RDM_DIRECTORY_SERVICE_LINK_FILTER == 0x20
flags	Required. Indicates the presence of optional directory refresh members. Refer to Section 8.4.2.2.
rdmMsgBase	Required. Contains general message information, such as streamId and domainType .

Table 139: RsslRDMDirectoryRefresh Structure Members

STRUCTURE MEMBER	DESCRIPTION
sequenceNumber	Optional. If present, a flags value of RDM_DR_RFF_HAS_SEQ_NUM should be specified. sequenceNumber is a user-specified, item-level sequence number that the application can use to sequence messages in the stream.
serviceCount	Required. Indicates the number of services present in the serviceList .
serviceId	Optional. If present, a flags value of RDM_DR_RFF_HAS_SERVICE_ID should be specified, which should match the serviceId of the consumer's RsslRDMDirectoryRequest .
serviceList	Optional. Presence indicated by serviceCount . Contains an array of information about available services.
state	Required. Indicates stream and data state information. For further details on RsslState , refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .

Table 139: RsslRDMDirectoryRefresh Structure Members (Continued)

8.4.2.2 Directory Refresh Flag Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_DR_RFF_CLEAR_CACHE	Indicates that any stored payload information associated with the directory stream should be cleared. This might happen if some portion of data is known to be invalid.
RDM_DR_RFF_HAS_SEQ_NUM	Indicates the presence of sequenceNumber .
RDM_DR_RFF_HAS_SERVICE_ID	Indicates the presence of serviceId .
RDM_DR_RFF_SOLICITED	If present, this flag indicates that the directory refresh is solicited (i.e., it is in response to a request). The absence of this flag indicates that the refresh is unsolicited.

Table 140: RsslRDMDirectoryRefresh Flags

8.4.2.3 Directory Refresh Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearRDMDirectoryRefresh	Clears an RsslRDMDirectoryRefresh structure. Useful for structure reuse.
rsslCopyRDMDirectoryRefresh	Performs a deep copy of an RsslRDMDirectoryRefresh structure.

Table 141: RsslRDMDirectoryRefresh Utility Functions

8.4.3 RSSL RDM Directory Update

A **Directory Update** message is encoded and sent by OMM provider and non-interactive provider applications. This message can provide information about new or removed services, or changes to existing services.

The **RsslRDMDirectoryUpdate** represents all members of a directory update message and allows for simplified use in OMM applications that leverage RDMs. This structure follows the behavior and layout that is defined in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.4.3.1 Directory Update Structure Members

STRUCTURE MEMBER	DESCRIPTION
filter	Optional. Indicates what information is provided about supported services. This should match the filter of the consumer's RsslRDMDirectoryRequest . If present, a flags value of RDM_DR_UPF_HAS_FILTER should be specified. Available flags are: <ul style="list-style-type: none"> • RDM_DIRECTORY_SERVICE_INFO_FILTER == 0x01 • RDM_DIRECTORY_SERVICE_STATE_FILTER == 0x02 • RDM_DIRECTORY_SERVICE_GROUP_FILTER == 0x04 • RDM_DIRECTORY_SERVICE_LOAD_FILTER == 0x08 • RDM_DIRECTORY_SERVICE_DATA_FILTER == 0x10 • RDM_DIRECTORY_SERVICE_LINK_FILTER == 0x20
flags	Required. Indicates the presence of optional directory update members. For details refer to Section 8.4.3.2.
sequenceNumber	Optional. A user-specified, item-level sequence number which the application can use to sequence messages in this stream. If present, a flags value of RDM_DR_UPF_HAS_SEQ_NUM should be specified.
serviceCount	Required. Indicates the number of services present in the serviceList .
serviceId	Optional. This member's value must match the serviceId of the consumer's RsslRDMDirectoryRequest . If present, a flags value of RDM_DR_UPF_HAS_SERVICE_ID should be specified.
serviceList	Optional. Presence indicated by serviceCount . Contains an array of information about available services.
rdmMsgBase	Required. Contains general message information like streamId and domainType .

Table 142: RsslRDMDirectoryUpdate Structure Members

8.4.3.2 Directory Update Flag Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_DR_UPF_HAS_FILTER	Indicates the presence of filter .
RDM_DR_UPF_HAS_SEQ_NUM	Indicates the presence of sequenceNumber .
RDM_DR_UPF_HAS_SERVICE_ID	Indicates the presence of serviceId .

Table 143: RsslRDMDirectoryUpdate Flags

8.4.3.3 Directory Update Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMDirectoryUpdate</code>	Clears an <code>RsslRDMDirectoryUpdate</code> structure. Useful for structure reuse.
<code>rsslCopyRDMDirectoryUpdate</code>	Performs a deep copy of an <code>RsslRDMDirectoryUpdate</code> structure.

Table 144: RsslRDMDirectoryUpdate Utility Functions

8.4.4 RSSL RDM Directory Status

OMM providers and non-interactive providers use the *Directory Status* message to convey state information associated with the directory stream. Such state information can indicate that a directory stream cannot be established or to inform a consumer of a state change associated with an open directory stream. An application can also use the Directory Status message to close an existing directory stream.

The `RsslRDMDirectoryStatus` represents all members of a directory status message and allows for simplified use in OMM applications that leverage RDMs. This structure follows the behavior and layout that is defined in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.4.4.1 Directory Status Structure Members

STRUCTURE MEMBER	DESCRIPTION
<code>filter</code>	Optional. If present, a flags value of <code>RDM_DR_STF_HAS_FILTER</code> should be specified. Indicates what information is being provided about supported services. This should match the filter of the consumer's <code>RsslRDMDirectoryRequest</code> . The available flags are: <ul style="list-style-type: none"> <code>RDM_DIRECTORY_SERVICE_INFO_FILTER == 0x01</code> <code>RDM_DIRECTORY_SERVICE_STATE_FILTER == 0x02</code> <code>RDM_DIRECTORY_SERVICE_GROUP_FILTER == 0x04</code> <code>RDM_DIRECTORY_SERVICE_LOAD_FILTER == 0x08</code> <code>RDM_DIRECTORY_SERVICE_DATA_FILTER == 0x10</code> <code>RDM_DIRECTORY_SERVICE_LINK_FILTER == 0x20</code>
<code>flags</code>	Required. Indicates the presence of optional directory status members. For details, refer to Section 8.4.4.2.
<code>serviceId</code>	Optional. If present, a flags value of <code>RDM_DR_STF_HAS_SERVICE_ID</code> should be specified. This member should match the serviceId of the consumer's <code>RsslRDMDirectoryRequest</code> .
<code>state</code>	Optional. Indicates the state of the directory stream. If present, a flags value of <code>RDM_DR_STF_HAS_STATE</code> should be specified. For more information on <code>RsslState</code> , refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
<code>rdmMsgBase</code>	Required. Contains general message information like streamId and domainType .

Table 145: RsslRDMDirectoryStatus Structure Members

8.4.4.2 Directory Status Flag Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_DR_STF_CLEAR_CACHE	Indicates that any stored payload data associated with the directory stream should be cleared. This might happen if some portion of data is known to be invalid.
RDM_DR_STF_HAS_FILTER	Indicates the presence of filter .
RDM_DR_STF_HAS_SERVICE_ID	Indicates the presence of serviceId .
RDM_DR_STF_HAS_STATE	Indicates the presence of state . If not present, any previously conveyed state should continue to apply.

Table 146: RsslRDMDirectoryStatus Flags

8.4.4.3 Directory Status Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearRDMDirectoryStatus	Clears an RsslRDMDirectoryStatus structure. Useful for structure reuse.
rsslCopyRDMDirectoryStatus	Performs a deep copy of an RsslRDMDirectoryStatus structure.

Table 147: RsslRDMDirectoryStatus Utility Functions

8.4.5 RSSL RDM Directory Close

8.4.5.1 Directory Close Structure Member

STRUCTURE MEMBER	DESCRIPTION
rdmMsgBase	Required . Contains general message information like streamId and domainType.

Table 148: RsslRDMDirectoryClose Structure Member

8.4.5.2 Directory Close Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearRDMDirectoryClose	Clears an RsslRDMDirectoryClose structure. Useful for structure reuse.
rsslCopyRDMDirectoryClose	Performs a deep copy of an RsslRDMDirectoryClose structure.

Table 149: RsslRDMDirectoryClose Utility Functions

8.4.6 RSSL RDM Consumer Status

The **Directory Consumer Status** is sent by OMM consumer applications to inform a service of how the consumer is used for **Source Mirroring**. This message is primarily informational.

The **RsslRDMDirectoryConsumerStatus** relies on the **RsslGenericMsg** and represents all members necessary for applications that leverage RDMs. This structure follows the behavior and layout that is defined in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.4.6.1 Directory Consumer Status Structure Members

STRUCTURE MEMBER	DESCRIPTION
consumerServiceStatusCount	Required . Indicates the number of services present in the serviceList .
consumerServiceStatusList	Optional . Presence indicated by consumerServiceStatusCount . Contains an array of RsslRDMConsumerStatusService structures.
rdmMsgBase	Required . Contains general message information like streamId and domainType .

Table 150: RsslRDMDirectoryConsumerStatus Structure Members

8.4.6.2 Directory Consumer Status Service Structure Members

STRUCTURE MEMBER	DESCRIPTION
action	Required . Indicates how a cache of Source Mirroring content should apply this information. For information on RsslMapEntry actions, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
serviceId	Required . Indicates the service associated with this status.
sourceMirroringMode	Required . Indicates how the consumer is using the service. Available enumerations are: <ul style="list-style-type: none"> • RDM_DIRECTORY_SOURCE_MIRROR_MODE_ACTIVE_NO_STANDBY == 0, • RDM_DIRECTORY_SOURCE_MIRROR_MODE_ACTIVE_WITH_STANDBY == 1, • RDM_DIRECTORY_SOURCE_MIRROR_MODE_STANDBY == 2

Table 151: RsslRDMConsumerStatusService Structure Members

8.4.6.3 Directory Consumer Status Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearRDMDirectoryConsumerStatus	Clears an RsslRDMDirectoryConsumerStatus structure. Useful for structure reuse.
rsslClearRDMConsumerStatusService	Clears the RsslRDMConsumerStatusService structure.
rsslCopyRDMDirectoryConsumerStatus	Performs a deep copy of an RsslRDMDirectoryConsumerStatus structure.

Table 152: RsslRDMDirectoryConsumerStatus Utility Functions

8.4.7 Source Directory RDM Service

An **RsslRDMSERVICE** structure conveys information about a service. An array of **RsslRDMServices** forms the **serviceList** member of the **RsslRDMDirectoryRefresh** and **RsslRDMDirectoryUpdate** messages.

The members of an **RsslRDMSERVICE** represent the different filters used to categorize service information.

8.4.7.1 RSSL RDM Service Structure Members

STRUCTURE MEMBER	DESCRIPTION
action	Required. Indicates how a cache of the service should apply this information. For information on RsslMapEntry actions, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
data	Optional. Contains data that applies to the items requested from the service and represents the Source Directory Data Filter. If present, a flags value of RDM_SVCF_HAS_DATA should be specified.
flags	Required. Indicates the presence of optional service members. For details, refer to Section 8.4.7.2.
groupStateCount	Required. Indicates the number of elements present in groupStateList .
groupStateList	Optional. Presence indicated by groupStateCount . Contains an array of elements indicating changes to item groups and represents the Source Directory Group filter.
info	Optional. Contains information related to the Source Directory Info Filter. If present, a flags value of RDM_SVCF_HAS_INFO should be specified.
linkInfo	Optional. Contains information about upstream sources that provide data to this service and represents the Source Directory Link Filter. If present, a flags value of RDM_SVCF_HAS_LINK should be specified.
load	Optional. Contains information about the service's operating workload and represents the Source Directory Load Filter. If present, a flags value of RDM_SVCF_HAS_LOAD should be specified.
serviceId	Required. Indicates the service associated with this RsslRDMSERVICE .
state	Optional. Contains information related to the Source Directory State Filter. If present, a flags value of RDM_SVCF_HAS_STATE should be specified.

Table 153: RsslRDMSERVICE Structure Members

8.4.7.2 RSSL Service Flag Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_SVCF_HAS_DATA	Indicates the presence of data .
RDM_SVCF_HAS_INFO	Indicates the presence of info .
RDM_SVCF_HAS_LINK	Indicates the presence of linkInfo .
RDM_SVCF_HAS_LOAD	Indicates the presence of load .
RDM_SVCF_HAS_STATE	Indicates the presence of state .

Table 154: RsslRDMSERVICE Flags

8.4.7.3 RSSL Service Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMSERVICE</code>	Clears an RsslRDMSERVICE structure. Useful for structure reuse.

Table 155: RsslRDMSERVICE Utility Function

8.4.8 Source Directory RDM Service Info

An **RsslRDMSERVICEInfo** structure conveys information that identifies the service and the content it provides. The **RsslRDMSERVICEInfo** structure represents the Source Directory Info filter. More information about the Info filter is available in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.4.8.1 RSSL Service Info Members

STRUCTURE MEMBER	DESCRIPTION
<code>acceptingConsumerStatus</code>	Optional. Indicates whether this service supports accepting RsslRDMDirectoryConsumerStatus messages for Source Mirroring. Available values are: <ul style="list-style-type: none"> 1: The service will accept Consumer Status messages. If not present, a value of 1 is assumed. 0: The service will not accept Consumer Status messages. If present, a flags value of RDM_SVC_IFF_HAS_ACCEPTING_CONS_STATUS should be specified.
<code>action</code>	Required. Indicates how a service info cache should apply this information. For information on RsslFilterEntry actions, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
<code>capabilitiesCount</code>	Required. Indicates the number of capabilities present in the capabilitiesList .
<code>capabilitiesList</code>	Required. Contains a list of capabilities that the service supports. Populated by domain types.
<code>dictionariesProvidedCount</code>	Optional. Indicates the number of elements present in dictionariesProvided . If present, a flags value of RDM_SVC_IFF_HAS_DICTS_PROVIDED should be specified.
<code>dictionariesProvidedList</code>	Optional. Contains an array of elements that identify dictionaries that can be requested from this service. If present, a flags value of RDM_SVC_IFF_HAS_DICTS_PROVIDED and dictionariesProvidedCount should be specified.
<code>dictionariesUsedCount</code>	Optional. Indicates the number of elements present in dictionariesUsed . If present, a flags value of RDM_SVC_IFF_HAS_DICTS_USED should be specified.
<code>dictionariesUsedList</code>	Optional. Contains an array of elements that identify dictionaries used to decode data from this service. If present, a flags value of RDM_SVC_IFF_HAS_DICTS_USED and dictionariesUsedCount should be specified.
<code>flags</code>	Required. Indicates the presence of optional service info members. For details, refer to Section 8.4.8.2.

Table 156: RsslRDMSERVICEInfo Structure Members

STRUCTURE MEMBER	DESCRIPTION
isSource	Optional. Indicates whether the service is provided directly by a source or represents a group of sources. <ul style="list-style-type: none"> • 1: The service is provided directly by a source • 0: The service represents a group of sources. If absent, a value of 0 is assumed. If present, a flags value of RDM_SVC_IFF_HAS_IS_SOURCE should be specified.
itemList	Optional. Specifies a name that can be requested on the RSSL_DMT_SYMBOL_LIST domain to get a list of all items available from this service. If present, a flags value of RDM_SVC_IFF_HAS_ITEM_LIST should be specified.
qosCount	Optional. Indicates the number of elements present in qosList . If present, a flags value of RDM_SVC_IFF_HAS_QOS should be specified.
qosList	Optional. Contains an array of elements that identify the available Qualities of Service. If present, a flags value of RDM_SVC_IFF_HAS_QOS and the qosCount should be specified.
serviceName	Required. Indicates the name of the service.
supportsOutOfBandSnapshots	Optional. Indicates whether this service supports making snapshot requests even when the OpenLimit is reached. Available values are: <ul style="list-style-type: none"> • 1: Snapshot requests are allowed. If not present, a value of 1 is assumed. • 0: Snapshot requests are not allowed. If present, a flags value of RDM_SVC_IFF_HAS_SUPPORT_OOB_SNAPSHOTS should be specified.
supportsQosRange	Optional. Indicates whether this service supports specifying a range of Qualities of Service when requesting an item. For further information, refer to the qos and worstQos members of the RsslRequestMsg in the Enterprise Transport API C Edition <i>Developers Guide</i> . Available values are: <ul style="list-style-type: none"> • 1: Quality of Service Range requests are supported. • 0: Quality of Service Range requests are not supported. If not present, a value of 0 is assumed. If present, a flags value of RDM_SVC_IFF_HAS_SUPPORT_QOS_RANGE should be specified.
vendor	Optional. Identifies the vendor of the data. If present, a flags value of RDM_SVC_IFF_HAS_VENDOR should be specified.

Table 156: RsslRDMServiceInfo Structure Members (Continued)

8.4.8.2 RSSL Service Info Flag Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_SVC_IFF_HAS_ACCEPTING_CONS_STATUS	Indicates the presence of acceptingConsumerStatus .
RDM_SVC_IFF_HAS_DICTS_PROVIDED	Indicates the presence of dictionariesProvidedList and dictionariesProvidedCount .
RDM_SVC_IFF_HAS_DICTS_USED	Indicates the presence of dictionariesUsedList and dictionariesUsedCount .
RDM_SVC_IFF_HAS_IS_SOURCE	Indicates the presence of isSource .

Table 157: RsslRDMServiceInfo Flags

FLAG ENUMERATION	DESCRIPTION
RDM_SVC_IFF_HAS_ITEM_LIST	Indicates the presence of itemList .
RDM_SVC_IFF_HAS_QOS	Indicates the presence of qosList and qosCount .
RDM_SVC_IFF_HAS_SUPPORT_OOB_SNAPSHOTS	Indicates the presence of supportsOutOfBandSnapshots .
RDM_SVC_IFF_HAS_SUPPORT_QOS_RANGE	Indicates the presence of supportsQosRange .
RDM_SVC_IFF_HAS_VENDOR	Indicates the presence of vendor .

Table 157: RsslRDMServiceInfo Flags (Continued)

8.4.8.3 RSSL Service Info Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearRDMServiceInfo	Clears an RsslRDMServiceInfo structure. Useful for structure reuse.

Table 158: RsslRDMServiceInfo Utility Functions

8.4.9 Source Directory RDM Service State

An **RsslRDMServiceState** structure conveys information about service's current state. It represents the Source Directory State filter. For more information about the State filter, refer to the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.4.9.1 RSSL Service State Members

STRUCTURE MEMBER	DESCRIPTION
acceptingRequests	Indicates whether the immediate provider (to which the consumer is directly connected) can handle the request. Available values are: <ul style="list-style-type: none"> 1: The service will accept new requests. 0: The service does not currently accept new requests. If present, flags value of RDM_SVC_STF_HAS_ACCEPTING_REQS should be specified.
action	Required . Indicates how a cache of the service state should apply this information. For details on RsslFilterEntry actions, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
flags	Required . Indicates the presence of optional service state members. For details refer to Section 8.4.9.2.
serviceState	Required . Indicates whether the original provider of the data can respond to new requests. Requests can still be made if so indicated by acceptingRequests . Available values are: <ul style="list-style-type: none"> 1: The original provider of the data is available. 0: The original provider of the data is not currently available.
status	This status should be applied to all open items associated with this service. If present, flags value of RDM_SVC_STF_HAS_STATUS should be specified.

Table 159: RsslRDMServiceState Structure Members

8.4.9.2 RSSL Service State Flag Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_SVC_STF_HAS_ACCEPTING_REQS	Indicates the presence of acceptingRequests .
RDM_SVC_STF_HAS_STATUS	Indicates the presence of status .

Table 160: RsslRDMServiceState Flags

8.4.9.3 RSSL Service State Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearRDMServiceState	Clears an RsslRDMServiceState structure. Useful for structure reuse.

Table 161: RsslRDMServiceState Utility Functions

8.4.10 Source Directory RDM Service Group State

An **RsslRDMServiceGroupState** structure is used to convey status and name changes for an item group. It represents the Source Directory Group filter. For further details about the Group State filter, refer to the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.4.10.1 RSSL Service Group State Members

STRUCTURE MEMBER	DESCRIPTION
action	Required . Indicates how a cache of the service group state should apply this information. For further details on RsslFilterEntry actions, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
flags	Required . Indicates the presence of optional service group members. For details, refer to Section 8.4.10.2.
group	Required . Identifies the name of the item group being changed.
mergedToGroup	Optional . Specifies the new group name. All items of the specified group are put into this new group. If present, a flags value of RDM_SVC_GRP_HAS_MERGED_TO_GROUP should be specified.
status	Optional . Specifies the status to apply to all open items associated with the group specified by group . If present, a flags value of RDM_SVC_GRP_HAS_STATUS should be specified.

Table 162: RsslRDMServiceGroupState Structure Members

8.4.10.2 RSSL Service Group State Flag Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_SVC_GRF_HAS_MERGED_TO_GROUP	Indicates the presence of mergedToGroup .
RDM_SVC_GRF_HAS_STATUS	Indicates the presence of status .

Table 163: `RsslRDMServiceGroupState` Flags

8.4.10.3 RSSL Service Group State Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMServiceGroupState</code>	Clears an RsslRDMServiceGroupState structure. Useful for structure reuse.

Table 164: `RsslRDMServiceGroupState` Utility Functions

8.4.11 Source Directory RDM Service Load

An **RsslRDMServiceLoad** structure conveys the workload of a service. It represents the Source Directory Load filter. For further details on the Service Load filter, refer to the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.4.11.1 RSSL Service Load Members

STRUCTURE MEMBER	DESCRIPTION
<code>action</code>	Required. Indicates how a cache of the service load should apply this information. For information on RsslFilterEntry actions, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
<code>flags</code>	Required. Indicates presence of optional service load members. For details, refer to Section 8.4.11.2.
<code>loadFactor</code>	If present, flags value of RDM_SVC_LDF_HAS_LOAD_FACTOR should be specified. Indicates the current workload on the source that provides data. A higher load factor indicates a higher workload. For more information, refer to the Enterprise Transport API C Edition <i>LSEG Domain Model Usage Guide</i> .
<code>openLimit</code>	Specifies the maximum number of streaming requests that the service allows. If present, flags value of RDM_SVC_LDF_HAS_OPEN_LIMIT should be specified.
<code>openWindow</code>	Specifies the maximum number of outstanding requests (i.e., requests awaiting a refresh) that the service allows. If present, flags value of RDM_SVC_LDF_HAS_OPEN_WINDOW should be specified.

Table 165: `RsslRDMServiceLoad` Structure Members

8.4.11.2 RSSL Service Load Flag Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_SVC_LDF_HAS_LOAD_FACTOR	Indicates the presence of loadFactor .
RDM_SVC_LDF_HAS_OPEN_LIMIT	Indicates the presence of openLimit .
RDM_SVC_LDF_HAS_OPEN_WINDOW	Indicates the presence of openWindow .

Table 166: RsslRDMSERVICELOAD Flags

8.4.11.3 RSSL Service Load Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearRDMSERVICELOAD	Clears a RsslRDMSERVICELOAD structure. Useful for structure reuse.

Table 167: RsslRDMSERVICELOAD Utility Functions

8.4.12 Source Directory RDM Service Data

An **RsslRDMSERVICEData** structure conveys the data to apply to all items of a service. It represents the Source Directory Data filter. For further details on the Data filter, refer to the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.4.12.1 RSSL Service Data Members

STRUCTURE MEMBER	DESCRIPTION
action	Required . Indicates how a cache of the service data should apply this information. For further details on RsslFilterEntry actions, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
data	Optional . Contains the encoded RsslBuffer representing the data. The type of the data is given by dataType . If present, a flags value of RDM_SVC_DTF_HAS_DATA should be specified.
dataType	Optional . Specifies the RsslDataType of the data. For information on RsslDataTypes , refer to the Enterprise Transport API C Edition <i>Developers Guide</i> . If present, a flags value of RDM_SVC_DTF_HAS_DATA should be specified.
flags	Required . Indicates the presence of optional service data members. For details, refer to Section 8.4.12.2.
type	Optional . Indicates the type of content present in data . Available enumerations are: <ul style="list-style-type: none"> RDM_DIRECTORY_DATA_TYPE_TIME == 1 RDM_DIRECTORY_DATA_TYPE_ALERT == 2 RDM_DIRECTORY_DATA_TYPE_HEADLINE == 3 RDM_DIRECTORY_DATA_TYPE_STATUS == 4 If present, flags value of RDM_SVC_DTF_HAS_DATA should be specified.

Table 168: RsslRDMSERVICEData Structure Members

8.4.12.2 RSSL Service Load Data Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_SVC_DTF_HAS_DATA	Indicates the presence of type , dataType , and data .

Table 169: RsslRDMServiceData Flags

8.4.12.3 RSSL Service Data Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearRDMServiceData	Clears an RsslRDMServiceData structure. Useful for structure reuse.

Table 170: RsslRDMServiceData Utility Functions

8.4.13 Source Directory RDM Service Link Information

An **RsslRDMServiceLinkInfo** structure conveys information about upstream sources that form a service. It represents the Source Directory Link filter. More information about the Service Link filter content is available in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

The **RsslRDMServiceLinkInfo** structure contains an array of **RsslRDMServiceLink** structures that each represents an upstream source.

8.4.13.1 RSSL Service Link Info Members

STRUCTURE MEMBER	DESCRIPTION
action	Required . Indicates how a cache of the service link information should apply this information. For further information on RsslFilterEntry actions, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
linkCount	Required . Indicates the number of link elements present in linkList .
linkList	Optional . Presence indicated by linkCount . Contains an array of RsslRDMServiceLink structures, each representing a source.

Table 171: RsslRDMServiceLinkInfo Structure Members

8.4.13.2 RSSL Service Link Info Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearRDMServiceLinkInfo	Clears an RsslRDMServiceLinkInfo structure. Useful for structure reuse.

Table 172: RsslRDMServiceLinkInfo Utility Functions

8.4.14 Source Directory RDM Service Link

An **RsslRDMSERVICELink** structure conveys information about an upstream source. It represents an entry in the Source Directory Link filter and is used by the **linkList** member of the **RsslRDMSERVICELinkInfo** structure. For further details on Service Link filter content, refer to the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.4.14.1 RSSL Service Link Members

STRUCTURE MEMBER	DESCRIPTION
action	Required. Indicates how a cache of the service link should apply this information. For information on RsslMapEntry actions, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
flags	Required. Indicates the presence of optional service link members. For details, refer to Section 8.4.14.2.
linkCode	Optional. Indicates additional information about the status of a source. Available enumerations are: <ul style="list-style-type: none"> RDM_DIRECTORY_LINK_CODE_NONE == 0 RDM_DIRECTORY_LINK_CODE_OK == 1 RDM_DIRECTORY_LINK_CODE_RECOVERY_STARTED == 2 RDM_DIRECTORY_LINK_CODE_RECOVERY_COMPLETED == 3 If present, a flags value of RDM_SVC_LKF_HAS_CODE should be specified.
linkState	<ul style="list-style-type: none"> Required. Indicates whether the source is up or down.
name	Required. Specifies the name of the source. Sources with identical names are typically load-balanced sources.
text	Optional. Gives additional status details regarding the source. If present, a flags value of RDM_SVC_LKF_HAS_TEXT should be specified.
type	Optional. Specifies whether the source is interactive or broadcast. Available enumerations are: <ul style="list-style-type: none"> RDM_DIRECTORY_LINK_TYPE_INTERACTIVE == 1 RDM_DIRECTORY_LINK_TYPE_BROADCAST == 2 If present, a flags value of RDM_SVC_LKF_HAS_TYPE should be specified.

Table 173: RsslRDMSERVICELink Structure Members

8.4.14.2 RSSL Service Link Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_SVC_LKF_HAS_CODE	Indicates the presence of code .
RDM_SVC_LKF_HAS_TEXT	Indicates the presence of text .
RDM_SVC_LKF_HAS_TYPE	Indicates the presence of type .

Table 174: RsslRDMSERVICELink Flags

8.4.14.3 RSSL Service Link Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMSERVICELink</code>	Clears an RsslRDMSERVICELink structure. Useful for structure reuse.

Table 175: RsslRDMSERVICELink Utility Functions

8.4.15 Source Directory RDM Sequenced Multicast Information

An **RsslRDMSERVICESeqMcastInfo** structure is included in the services advertised by the Reference Data Server component of an Real Time Direct system. It identifies components in the system to which an Open Message Model Consumer application connects for content.

- For further information on the service sequenced multicast information filter, refer to the Transport API C Edition *RDM Usage Guide*.
- For further information on Real Time Direct, refer to the *LSEG Real Time Direct Developers Guide*.

8.4.15.1 RSSL Service Sequenced Multicast Information Structure

STRUCTURE MEMBER	DESCRIPTION
<code>flags</code>	Required. Sets any optional members. For details, refer to Section 8.4.15.2.
<code>action</code>	Required. Sets how a cache should apply this information. For details, refer to the <i>Transport API C Edition Developers Guide</i> .
<code>snapshotServer</code>	Sets the network address/port information for the Snapshot Server, if one is present.
<code>gapRecoveryServer</code>	Sets the network address/port information for the Gap Recovery Server, if one is present.
<code>refDataServer</code>	Sets the network address/port information for the Reference Data Server, if one is present.
<code>StreamingMcastChanServerCount</code>	The number of real time stream components in StreamingMcastChanServerList .
<code>StreamingMcastChanServerList</code>	Sets the network address/port information for real time stream components.
<code>GapMcastChanServerCount</code>	Number of Gap Fill Server components in GapMcastServerList .
<code>GapMcastChanServerList</code>	Sets the network address/port information for Gap Fill Server components.

Table 176: RsslRDMSERVICESeqMcastInfo Structure Members

8.4.15.2 RSSL Service Sequenced Multicast Info Enumeration Values

FLAG ENUMERATION	DESCRIPTION
<code>RDM_SVC_SMF_HAS_SNAPSHOT_SERV</code>	Indicates the presence of snapshotServer .
<code>RDM_SVC_SMF_HAS_GAP_REC_SERV</code>	Indicates the presence of gapRecoveryServer .
<code>RDM_SVC_SMF_HAS_REF_DATA_SERV</code>	Indicates the presence of refDataServer .
<code>RDM_SVC_SMF_HAS_SMC_SERV</code>	Indicates the presence of StreamingMcastChanServerList .
<code>RDM_SVC_SMF_HAS_GMC_SERV</code>	Indicates the presence of GapMcastChanServerList .

Table 177: RSSL RDM Service Sequenced Multicast Info Enumeration Values

8.4.15.3 RSSL Address/Port Information

STRUCTURE MEMBER	DESCRIPTION
address	The network address of the component.
port	The network port of the component.
domain	The item domain associated with this component (e.g.: 6 (MarketPrice)).

Table 178: RSSL RDM Address/Port Information Structure Members

8.4.15.4 RSSL Sequenced Multicast Info Utility Functions

UTILITY	DESCRIPTION
rsslClearRDMMCAAddressPortInfo	Clears an RsslRDMMCAAddressPortInfo structure.
rsslClearRDMSERVICESeqMcastInfo	Clears an RsslRDMSERVICESeqMcastInfo structure.

Table 179: RsslRDMSERVICESeqMcastInfo Utility Functions

8.4.16 RSSL RDM Directory Message Union

This union can contain any of the RDM Directory message types. This is provided for use with directory-specific functionality.

8.4.16.1 Directory Union

UNION MEMBERS	DESCRIPTION
rdmMsgBase	The message base information.
request	The RsslRDMDirectoryRequest as described in Section 8.4.1.
close	The RsslRDMDirectoryClose as described in Section 8.4.5.
refresh	The RsslRDMDirectoryRefresh as described in Section 8.4.2.
status	The RsslRDMDirectoryStatus as described in Section 8.4.4.
update	The RsslRDMDirectoryUpdate as described in Section 8.4.3.
consumerStatus	The RsslRDMDirectoryConsumerStatus as described in Section 8.4.6.

Table 180: RsslRDMDirectoryMsg Union Members

8.4.16.2 Directory Message Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearRDMDirectoryMsg	Clears an RsslRDMDirectoryMsg union. Useful for reuse.
rsslCopyRDMDirectoryMsg	Performs a deep copy of an RsslRDMDirectoryMsg structure.

Table 181: RsslRDMDirectoryMsg Utility Functions

8.4.17 Source Directory Encoding and Decoding

8.4.17.1 RSSL RDM Directory Encoding and Decoding Functions

FUNCTION NAME	DESCRIPTION
<code>rsslEncodeRDMDirectoryMsg</code>	Used to encode an RDM Directory message. This function takes the RsslRDMDirectoryMsg as a parameter. Alternately, rsslEncodeRDMMsg can be used if encoding from an RsslRDMMsg .
<code>rsslDecodeRDMDirectoryMsg</code>	Used to decode an RDM Directory message. This function populates the RsslRDMDirectoryMsg and leverages the Value Added Utility message buffer (refer to Section 11.2). Alternately, rsslDecodeRDMMsg can be used to decode into an RsslRDMMsg .

Table 182: RDM Directory Encoding and Decoding Functions

8.4.17.2 Encoding a Source Directory Request

```

RsslEncodeIterator encodeIter;
RsslRDMDirectoryRequest directoryRequest;

/* Clear the Directory Request structure. */
rsslClearRDMDirectoryRequest(&directoryRequest);

/* Set flags indicating presence of optional members. */
directoryRequest.flags =
    RDM_DR_RQF_HAS_SERVICE_ID
    | RDM_DR_RQF_STREAMING;

/* Set Service ID. */
directoryRequest.serviceId = 273;

/* Set ApplicationName. */
directoryRequest.filter =
    RDM_DIRECTORY_SERVICE_INFO_FILTER
    | RDM_DIRECTORY_SERVICE_STATE_FILTER
    | RDM_DIRECTORY_SERVICE_GROUP_FILTER;

/* Clear the encode iterator, set its RWF Version, and set it to a buffer for encoding into. */
rsslClearEncodeIterator(&encodeIter);
ret = rsslSetEncodeIteratorRWFVersion(&encodeIter, channelMajorVersion, channelMinorVersion);
ret = rsslSetEncodeIteratorBuffer(&encodeIter, &msgBuffer);

/* Encode the message. */
ret = rsslEncodeRDMMsg(&encodeIter, (RsslRDMMsg*)&directoryRequest, &msgBuffer.length,
    &rsslErrorInfo);

```

Code Example 23: Directory Request Encoding Example

8.4.17.3 Decoding a Source Directory Request

```

/* The decoder may require additional space to store things such as lists. */
char memoryArray[1024];
RsslBuffer memoryBuffer = { 1024, memoryArray };

RsslDecodeIterator decodeIter;
RsslMsg msg;
RsslRDMMsg rdmMsg;
RsslRDMDirectoryRequest *pDirectoryRequest;

/* Clear the decode iterator, set its RWF Version, and set it to the encoded buffer. */
rsslClearDecodeIterator(&decodeIter);
ret = rsslSetDecodeIteratorRWFVersion(&decodeIter, channelMajorVersion, channelMinorVersion);
ret = rsslSetDecodeIteratorBuffer(&decodeIter, &msgBuffer);

/* Decode the message to an RsslMsg structure and RsslRDMMsg structure. */
ret = rsslDecodeRDMMsg(&decodeIter, &msg, &rdmMsg, &memoryBuffer, &rsslErrorInfo);

if (ret == RSSL_RET_SUCCESS
    && rdmMsg.rdmMsgBase.domainType == RSSL_DMT_SOURCE && rdmMsg.rdmMsgBase.rdmMsgType ==
    RDM_DR_MT_REQUEST)
{
    /* The message we decoded is an RsslRDMDirectoryRequest. */
    pDirectoryRequest = &rdmMsg.directoryMsg.request;

    /* Print if Info filter was requested. */
    if (pDirectoryRequest->filter & RDM_DIRECTORY_SERVICE_INFO_FILTER)
        printf("Info filter requested.\n");

    /* Print if State filter was requested. */
    if (pDirectoryRequest->filter & RDM_DIRECTORY_SERVICE_STATE_FILTER)
        printf("State filter requested.\n");

    /* Print if Group filter was requested. */
    if (pDirectoryRequest->filter & RDM_DIRECTORY_SERVICE_GROUP_FILTER)
        printf("Group filter requested.\n");

    /* Print service ID if present. */
    if (pDirectoryRequest->flags & RDM_DR_RQF_HAS_SERVICE_ID)
        printf("Service ID: %u\n", pDirectoryRequest->serviceId);
}

```

Code Example 24: Directory Request Decoding Example

8.4.17.4 Encoding a Source Directory Refresh

```

RsslEncodeIterator encodeIter;
RsslRDMDirectoryRefresh directoryRefresh;

/* List of services to be used.
 * This example will show encoding of one service. Additional services
 * can be set up using the same method shown below. */

RsslRDMService serviceList[1];
/* Lists to be used with MY_SERVICE. */
RsslUInt capabilitiesList[3];
RsslBuffer dictionariesList[2];
RsslQos qosList[2];

/* Clear the Directory Refresh structure. */
rsslClearRDMDirectoryRefresh(&directoryRefresh);

/* Set flags */
directoryRefresh.flags = RDM_DR_RFF_SOLICITED;

/* Set state. */
directoryRefresh.state.streamState = RSSL_STREAM_OPEN;
directoryRefresh.state.dataState = RSSL_DATA_OK;
directoryRefresh.state.code = RSSL_SC_NONE;

/* Set filter to say the Info, State, and Group filters are supported. */
directoryRefresh.filter =
    RDM_DIRECTORY_SERVICE_INFO_FILTER
    | RDM_DIRECTORY_SERVICE_STATE_FILTER
    | RDM_DIRECTORY_SERVICE_GROUP_FILTER;

/** Build Service MY_SERVICE. */

rsslClearRDMService(&serviceList[0]);

/* Set flags to indicate Info and State filter are present. */
serviceList[0].flags =
    RDM_SVCF_HAS_INFO
    | RDM_SVCF_HAS_STATE;

/* Set action to indicate adding a new service. */
serviceList[0].info.action = RSSL_MPEA_ADD_ENTRY;

/** Build Info for MY_SERVICE. */

/* Set flags to indicate optional members. */
serviceList[0].info.flags =
    RDM_SVC_IFF_HAS_VENDOR
    | RDM_SVC_IFF_HAS_DICTS_PROVIDED

```

```

    | RDM_SVC_IFF_HAS_DICTS_USED
    | RDM_SVC_IFF_HAS_QOS;

/* Set service name. */
serviceList[0].info.serviceName.data = "MY_SERVICE";
serviceList[0].info.serviceName.length = 10;

/* Set vendor name. */
serviceList[0].info.vendor.data = "LSEG";
serviceList[0].info.vendor.length = 4;

/* Build capabilities list. */
capabilitiesList[0] = RSSL_DMT_DICTIONARY;
capabilitiesList[1] = RSSL_DMT_MARKET_PRICE;
capabilitiesList[2] = RSSL_DMT_MARKET_BY_ORDER;

/* Set capabilities list. */
serviceList[0].info.capabilitiesList = capabilitiesList;
serviceList[0].info.capabilitiesCount = 3;

/* Build dictionary list to use with dictionariesProvidedList and dictionariesUsedList. */
dictionariesList[0].data = "RWFFld";
dictionariesList[0].length = 6;
dictionariesList[1].data = "RWFEEnum";
dictionariesList[1].length = 7;

/* Set dictionaries provided. */
serviceList[0].info.dictionariesProvidedList = dictionariesList;
serviceList[0].info.dictionariesProvidedCount = 2;

/* Set dictionaries used. */
serviceList[0].info.dictionariesUsedList = dictionariesList;
serviceList[0].info.dictionariesUsedCount = 2;

/* Build QoS list. */
qosList[0].timeliness = RSSL_QOS_TIME_REALTIME;
qosList[0].rate = RSSL_QOS_RATE_TICK_BY_TICK;
qosList[1].timeliness = RSSL_QOS_TIME_REALTIME;
qosList[1].rate = RSSL_QOS_RATE_JIT_CONFLATED;

/* Set QoS list. */
serviceList[0].info.qosList = qosList;
serviceList[0].info.qosCount = 2;

/** Build Service State for MY_SERVICE */
serviceList[0].state.flags = RDM_SVC_STF_HAS_ACCEPTING_REQS;
serviceList[0].state.serviceState = 1;
serviceList[0].state.acceptingRequests = 1;

/** Finish and encode. */

```



```

/* Set the array of services on the message. The refresh will information about 2 services*/
directoryRefresh.serviceList = serviceList;
directoryRefresh.serviceCount = 1;

/* Clear the encode iterator, set its RWF Version, and set it to a buffer for encoding into. */
rsslClearEncodeIterator(&encodeIter);
ret = rsslSetEncodeIteratorRWFVersion(&encodeIter, channelMajorVersion, channelMinorVersion);
ret = rsslSetEncodeIteratorBuffer(&encodeIter, &msgBuffer);

/* Encode the message. */
ret = rsslEncodeRDMMsg(&encodeIter, (RsslRDMMsg*)&directoryRefresh, &msgBuffer.length,
    &rsslErrorInfo);

```

Code Example 25: Directory Refresh Encoding Example

8.4.17.5 Decoding a Source Directory Refresh

```

/* The decoder may require additional space to store things such as lists. */
char memoryArray[4096];
RsslBuffer memoryBuffer = { 4096, memoryArray };

RsslDecodeIterator decodeIter;
RsslMsg msg;
RsslRDMMsg rdmMsg;
RsslRDMDirectoryRefresh *pDirectoryRefresh;

/* Clear the decode iterator, set its RWF Version, and set it to the encoded buffer. */
rsslClearDecodeIterator(&decodeIter);
ret = rsslSetDecodeIteratorRWFVersion(&decodeIter, channelMajorVersion, channelMinorVersion);
ret = rsslSetDecodeIteratorBuffer(&decodeIter, &msgBuffer);

/* Decode the message to an RsslMsg structure and RsslRDMMsg structure. */
ret = rsslDecodeRDMMsg(&decodeIter, &msg, &rdmMsg, &memoryBuffer, &rsslErrorInfo);

if (ret == RSSL_RET_SUCCESS && rdmMsg.rdmMsgBase.domainType == RSSL_DMT_SOURCE &&
    rdmMsg.rdmMsgBase.rdmMsgType == RDM_DR_MT_REFRESH)
{
    RsslUInt32 i;

    /* The message we decoded is an RsslRDMDirectoryRefresh. */
    pDirectoryRefresh = &rdmMsg.directoryMsg.refresh;

    /* Print serviceId if present. */
    if (pDirectoryRefresh->flags & RDM_DR_RFF_HAS_SERVICE_ID)
        printf("Service ID: %u\n", pDirectoryRefresh->serviceId);

    /* Print information about each service present in the refresh. */
    for(i = 0; i < pDirectoryRefresh->serviceCount; ++i)

```

```

{
    /* Print Service Info if present */
    if (pDirectoryRefresh->serviceList[i].flags & RDM_SVCF_HAS_INFO)
    {
        RsslUInt32 j;
        RsslRDMSERVICEINFO *pInfo = &pDirectoryRefresh->serviceList[i].info;

        /* Print service name. */
        printf("Service Name: %.*s\n", pInfo->serviceName.length, pInfo->serviceName.data);

        /* Print vendor name if present. */
        if (pInfo->flags & RDM_SVC_IFF_HAS_VENDOR)
            printf("Vendor: %.*s\n", pInfo->vendor.length, pInfo->vendor.data);

        /* Print supported domains if present. */
        for (j = 0; j < pInfo->capabilitiesCount; ++j)
            printf("Capability: %s\n", rsslDomainTypeToString(pInfo->capabilitiesList[j]));

        /* Print dictionaries provided if present. */
        if (pInfo->flags & RDM_SVC_IFF_HAS_DICTS_PROVIDED)
        {
            for (j = 0; j < pInfo->dictionariesProvidedCount; ++j)
                printf("Dictionary Provided: %.*s\n", pInfo->dictionariesProvidedList[j].length,
                    pInfo->dictionariesProvidedList[j].data);
        }

        /* Print dictionaries used if present. */
        if (pInfo->flags & RDM_SVC_IFF_HAS_DICTS_USED)
        {
            for (j = 0; j < pInfo->dictionariesUsedCount; ++j)
                printf("Dictionary Used: %.*s\n", pInfo->dictionariesUsedList[j].length,
                    pInfo->dictionariesUsedList[j].data);
        }

        /* Print qualities of service supported if present. */
        if (pInfo->flags & RDM_SVC_IFF_HAS_QOS)
        {
            for (j = 0; j < pInfo->qosCount; ++j)
                printf("QoS: %s,%s\n", rsslQosTimelinessToString(pInfo->
                    qosList[j].timeliness), rsslQosRateToString(pInfo->qosList[j].rate));
        }
    }

    /* Print Service State if present */
    if (pDirectoryRefresh->serviceList[i].flags & RDM_SVCF_HAS_STATE)
    {
        RsslRDMSERVICESTATE *pState = &pDirectoryRefresh->serviceList[i].state;

        printf("Service State: %llu\n", pState->serviceState);
    }
}

```

```
        if (pState->flags & RDM_SVC_STF_HAS_ACCEPTING_REQS)
            printf("Accepting Requests: %llu\n", pState->acceptingRequests);
    }
}
```

Code Example 26: Directory Refresh Decoding Example

8.5 Dictionary Domain

The Dictionary domain model conveys information needed for parsing published data. Dictionaries provide additional meta-data, such as that necessary to decode the content of an **RsslFieldEntry** or additional content related to its **fieldId**. For more information about the different types of dictionaries and their usage, refer to the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

The structures provided for this domain make it easier to use the existing utilities for encoding, decoding, and caching dictionary information. For more information on these utilities, see the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.5.1 RSSL RDM Dictionary Request

A **Dictionary Request** message is encoded and sent by OMM consumer applications. This message requests a dictionary from a service.

The **RsslRDMDictionaryRequest** represents all members of a dictionary request message and is easily used in OMM applications that leverage RDMs. This structure follows the behavior and layout that is defined in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.5.1.1 Dictionary Request Structure Members

STRUCTURE MEMBER	DESCRIPTION
dictionaryName	Required . Indicates the name of the dictionary being requested.
flags	Required . Indicates the presence of optional dictionary request members. For details, refer to Section 8.5.1.2.
rdmMsgBase	Required . Contains general message information like streamId and domainType.
serviceId	Required . Specifies the service from which to request the dictionary.
verbosity	Required . Indicates the amount of information desired from the dictionary. Available enumerations are: <ul style="list-style-type: none"> RDM_DICTIONARY_INFO == 0x00: Version information only RDM_DICTIONARY_MINIMAL == 0x03: Provides information needed for caching RDM_DICTIONARY_NORMAL == 0x07: Provides all information needed for decoding RDM_DICTIONARY_VERBOSE == 0x0F: Provides all information (including comments) Providers are not required to support the MINIMAL and VERBOSE filters.

Table 183: RsslRDMDictionaryRequest Structure Members

8.5.1.2 Dictionary Request Flag Enumeration Value

FLAG ENUMERATION	DESCRIPTION
RDM_DC_RQF_STREAMING	Indicates that the dictionary stream should remain open after the initial refresh. An open stream can listen for status messages that indicate changes to the dictionary version. For more information, see the Enterprise Transport API C Edition <i>LSEG Domain Model Usage Guide</i> .

Table 184: RsslRDMDictionaryRequest Flag

8.5.1.3 Dictionary Request Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMDictionaryRequest</code>	Clears an RsslRDMDictionaryRequest structure. Useful for structure reuse.
<code>rsslCopyRDMDictionaryRequest</code>	Performs a deep copy of an RsslRDMDictionaryRequest structure.

Table 185: RsslRDMDictionaryRequest Utility Functions

8.5.2 RSSL RDM Dictionary Refresh

A **Dictionary Refresh** message is encoded and sent by OMM provider applications. This message transmits dictionary content in response to a request.

The **RsslRDMDictionaryRefresh** represents all members of a dictionary refresh message and is easy to use in OMM applications that leverage RDMs. This structure follows the behavior and layout that is defined in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.5.2.1 Dictionary Refresh Structure Members


STRUCTURE MEMBER	DESCRIPTION
<code>dataBody</code>	When decoding, this points to the encoded data buffer with dictionary content. This buffer should be set on an RsslDecodeIterator and passed to the appropriate decode function according to the type . Not used when encoding. The dictionary is retrieved from the RsslDataDictionary structure.
<code>pDictionary</code>	Conditional (required when encoding). Points to an RsslDataDictionary object that contains content to encode. For more information on the RsslDataDictionary structure, refer to the Enterprise Transport API C Edition <i>LSEG Domain Model Usage Guide</i> . Not used when decoding.
<code>dictionaryId</code>	When decoding, this will be populated with the dictionary's ID. The Dictionary is retrieved from the RsslDataDictionary structure. This structure's presence is indicated by the RDM_DC_RFF_HAS_INFO flag. Not used when encoding.
<code>dictionaryName</code>	Required . Indicates the name of the dictionary being provided.
<code>flags</code>	Required . Indicates the presence of optional dictionary refresh members. For details, refer to Section 8.5.2.2.
<code>rdmMsgBase</code>	Required . Contains general message information like streamId and domainType .
<code>sequenceNumber</code>	Optional . A user-specified, item-level sequence number that the application can use to sequence messages in this stream. If present, a flags value of RDM_DC_RFF_HAS_SEQ_NUM should be specified.
<code>serviceId</code>	Required . Indicates the service ID of the service from which the dictionary is provided.
<code>startFid</code>	Maintains the state when encoding a dictionary across multiple messages.  WARNING! To ensure that all dictionary content is correctly encoded, the application should not modify this.

Table 186: RsslRDMDictionaryRefresh Structure Members

STRUCTURE MEMBER	DESCRIPTION
state	Required. Indicates the state of the dictionary stream. Defaults to a streamState of RSSL_STREAM_OPEN and a dataState of RSSL_DATA_OK . For more information on RsslState , refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
type	Required. Indicates the type of dictionary being provided. The dictionary encoder and decoder support the following types: <ul style="list-style-type: none"> RDM_DICTIONARY_FIELD_DEFINITIONS == 1 RDM_DICTIONARY_ENUM_TABLES == 2
verbosity	Required. Indicates the amount of information desired from the dictionary. Available enumerations are: <ul style="list-style-type: none"> RDM_DICTIONARY_INFO == 0x00: Provides version information only RDM_DICTIONARY_MINIMAL == 0x03: Provides information needed for caching RDM_DICTIONARY_NORMAL == 0x07: Provides all information needed for decoding RDM_DICTIONARY_VERBOSE == 0x0F: Provides all information (including comments) Providers do not need to support the MINIMAL and VERBOSE filters.
version	When decoding, this will be populated with the dictionary's version string. Presence is indicated by the RDM_DC_RFF_HAS_INFO flag. Not used when encoding. The Dictionary is retrieved from the RsslDataDictionary structure.

Table 186: RsslRDMDictionaryRefresh Structure Members (Continued)

8.5.2.2 Dictionary Refresh Flag Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_DC_RFF_CLEAR_CACHE	Indicates that stored payload information associated with the dictionary stream should be cleared. This might happen if some portion of data is known to be invalid.
RDM_DC_RFF_HAS_INFO	Indicates the presence of dictionaryId , version , and type . Not used when encoding. The encode function adds information to the encoded message when appropriate.
RDM_DC_RFF_HAS_SEQ_NUM	Indicates the presence of sequenceNumber .
RDM_DC_RFF_IS_COMPLETE	Indicates that this is the final fragment and that the consumer has received all content for this dictionary. Not used when encoding. The encode function adds information to the encoded message when appropriate.
RDM_DC_RFF_SOLICITED	Indicates that the directory refresh is solicited (e.g., it is a response to a request). If the flag is not present, this refresh is unsolicited.

Table 187: RsslRDMDictionaryRefreshFlags

8.5.2.3 Dictionary Refresh Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMDictionaryRefresh</code>	Clears an RsslRDMDictionaryRefresh structure. Useful for structure reuse.
<code>rsslCopyRDMDictionaryRefresh</code>	Performs a deep copy of an RsslRDMDictionaryRefresh structure.

Table 188: RsslRDMDictionaryRefresh Utility Functions

8.5.3 RSSL RDM Dictionary Status

OMM provider and non-interactive provider applications use the **Dictionary Status** message to convey state information associated with the dictionary stream. Such state information can indicate that a dictionary stream cannot be established or to inform a consumer of a state change associated with an open login stream. The Dictionary status message can also indicate that a new dictionary should be retrieved. For more information on handling Dictionary versions, see the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

The **RsslRDMDictionaryStatus** represents all members of a dictionary status message and allows for simplified use in OMM applications that leverage RDMs. This structure follows the behavior and layout that is defined in the Enterprise Transport API C Edition *LSEG Domain Model Usage Guide*.

8.5.3.1 Dictionary Status Structure Members

STRUCTURE MEMBER	DESCRIPTION
<code>flags</code>	Required. Indicate the presence of optional dictionary status members. For details, refer to Section 8.5.3.2.
<code>rdmMsgBase</code>	Required. Contains general message information like streamId and domainType .
<code>state</code>	Optional. Indicates the state of the dictionary stream. For more information on RsslState , refer to the Enterprise Transport API C Edition <i>Developers Guide</i> . If present, a flags value of RDM_DC_STF_HAS_STATE should be specified.

Table 189: RsslRDMDictionaryStatus Structure Members

8.5.3.2 Dictionary Status Flag Enumeration Value

FLAG ENUMERATION	DESCRIPTION
<code>RDM_DC_STF_CLEAR_CACHE</code>	Indicates that any stored payload information associated with the dictionary stream should be cleared. This might happen if some portion of data is known to be invalid.
<code>RDM_DC_STF_HAS_STATE</code>	Indicates the presence of state . If absent, any previously conveyed state continues to apply.

Table 190: RsslRDMDictionaryStatus Flags

8.5.3.3 Dictionary Status Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMLLoginStatus</code>	Clears an RsslRDMLLoginStatus structure. Useful for structure reuse.
<code>rsslCopyRDMLLoginStatus</code>	Performs a deep copy of an RsslRDMLLoginStatus structure.

Table 191: RsslRDMDictionaryStatus Utility Functions

8.5.4 RSSL RDM Dictionary Close

A **Dictionary Close** message is encoded and sent by Open Message Model consumer applications. This message allows a consumer to close an open dictionary stream. A provider can close the directory stream via a Dictionary Status message, refer to Section 8.5.3.

8.5.4.1 Dictionary Close Structure Members

STRUCTURE MEMBER	DESCRIPTION
<code>rdmMsgBase</code>	Required. Contains general message information like streamId and domainType .

Table 192: RsslRDMDictionaryClose Structure Members

8.5.4.2 Dictionary Close Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMDictionaryClose</code>	Clears an RsslRDMDictionaryClose structure. Useful for structure reuse.
<code>rsslCopyRDMDictionaryClose</code>	Performs a deep copy of an RsslRDMDictionaryClose structure.

Table 193: RsslRDMDictionaryClose Utility Functions

8.5.5 RSSL RDM Dictionary Message Union

This union can contain any of the RDM Dictionary message types. This is provided for use with Dictionary specific functionality.

8.5.5.1 Dictionary Union

UNION MEMBERS	DESCRIPTION
<code>rdmMsgBase</code>	The message base information.
<code>request</code>	The RsslRDMDictionaryRequest as described in Section 8.5.1.
<code>close</code>	The RsslRDMDictionaryClose as described in Section 8.5.4.
<code>refresh</code>	The RsslRDMDictionaryRefresh as described in Section 8.5.2.
<code>status</code>	The RsslRDMDictionaryStatus as described in Section 8.5.3.

Table 194: RsslRDMDictionaryMsg Union Members

8.5.5.2 Dictionary Message Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rsslClearRDMDictionaryMsg</code>	Clears an RsslRDMDictionaryMsg union. Useful for reuse.
<code>rsslCopyRDMDictionaryMsg</code>	Performs a deep copy of an RsslRDMDictionaryMsg structure.

Table 195: RsslRDMDictionaryMsg Utility Functions

8.5.6 Dictionary Encoding and Decoding

8.5.6.1 RSSL RDM Dictionary Encoding and Decoding Functions

FUNCTION NAME	DESCRIPTION
<code>rsslEncodeRDMDictionaryMsg</code>	Used to encode an RDM Dictionary message. This function takes the RsslRDMDictionaryMsg as a parameter. Alternately, <code>rsslEncodeRDMMsg</code> can be used if encoding from an RsslRDMMsg .
<code>rsslDecodeRDMDictionaryMsg</code>	Used to decode an RDM Directory message. This function populates the RsslRDMDictionaryMsg and leverages the Value Added Utility message buffer (refer to Section 11.2). Alternately, <code>rsslDecodeRDMMsg</code> can be used to decode into an RsslRDMMsg .

Table 196: RDM Dictionary Encoding and Decoding Functions

8.5.6.2 Encoding a Dictionary Request

```

RsslEncodeIterator encodeIter;
RsslRDMDictionaryRequest dictionaryRequest;

/* Clear the Dictionary Request structure. */
rsslClearRDMDictionaryRequest(&dictionaryRequest);

/* Set flags. */
dictionaryRequest.flags = RDM_DC_RQF_STREAMING;

/* Set serviceId. */
dictionaryRequest.serviceId = 273;

/* Set verbosity. */
dictionaryRequest.verbosity = RDM_DICTIONARY_NORMAL;

/* Set dictionary name. */
dictionaryRequest.dictionaryName.data = "RWFFld";
dictionaryRequest.dictionaryName.length = 6;

/* Clear the encode iterator, set its RWF Version, and set it to a buffer for encoding into. */
rsslClearEncodeIterator(&encodeIter);
ret = rsslSetEncodeIteratorRWFVersion(&encodeIter, channelMajorVersion, channelMinorVersion);
ret = rsslSetEncodeIteratorBuffer(&encodeIter, &msgBuffer);

/* Encode the message. */

```

```
ret = rsslEncodeRDMMsg(&encodeIter, (RsslRDMMsg*)&dictionaryRequest, &msgBuffer.length,
    &rsslErrorInfo);
```

Code Example 27: Dictionary Request Encoding Example

8.5.6.3 Decoding a Dictionary Request

```
/* The decoder may require additional space to store things such as lists. */
char memoryArray[1024];
RsslBuffer memoryBuffer = { 1024, memoryArray };

RsslDecodeIterator decodeIter;
RsslMsg msg;
RsslRDMMsg rdmMsg;
RsslRDMDictionaryRequest *pDictionaryRequest;

/* Clear the decode iterator, set its RWF Version, and set it to the encoded buffer. */
rsslClearDecodeIterator(&decodeIter);
ret = rsslSetDecodeIteratorRWFVersion(&decodeIter, channelMajorVersion, channelMinorVersion);
ret = rsslSetDecodeIteratorBuffer(&decodeIter, &msgBuffer);

/* Decode the message to an RsslMsg structure and RsslRDMMsg structure. */
ret = rsslDecodeRDMMsg(&decodeIter, &msg, &rdmMsg, &memoryBuffer, &rsslErrorInfo);

if (ret == RSSL_RET_SUCCESS && rdmMsg.rdmMsgBase.domainType == RSSL_DMT_DICTIONARY &&
    rdmMsg.rdmMsgBase.rdmMsgType == RDM_DC_MT_REQUEST)
{
    /* The message we decoded is an RsslRDMDictionaryRequest. */
    pDictionaryRequest = &rdmMsg.dictionaryMsg.request;

    /* Print if streaming. */
    if (pDictionaryRequest->flags & RDM_DC_RQF_STREAMING)
        printf("Request is streaming.\n");

    /* Print serviceId. */
    printf("Service ID: %u\n", pDictionaryRequest->serviceId);

    /* Print verbosity. */
    printf("Verbosity: %u\n", pDictionaryRequest->verbosity);

    /* Print dictionary name. */
    printf("Dictionary Name: %.*s\n", pDictionaryRequest->dictionaryName.length, pDictionaryRequest->
        dictionaryName.data);
}
```

Code Example 28: Dictionary Request Decoding Example

8.5.6.4 Encoding a Dictionary Refresh

```

RsslEncodeIterator encodeIter;
RsslRDMDictionaryRefresh dictionaryRefresh;
RsslDataDictionary dataDictionary;

rsslClearDataDictionary(&dataDictionary);
ret = rsslLoadFieldDictionary("RDMFieldDictionary", &dataDictionary, &errorText);

/* Clear the Dictionary Refresh structure. */
rsslClearRDMDictionaryRefresh(&dictionaryRefresh);

/* Set flags. */
dictionaryRefresh.flags = RDM_DC_RFF_SOLICITED;

/* Set dictionary name. */
dictionaryRefresh.dictionaryName.data = "RWFFld";
dictionaryRefresh.dictionaryName.length = 6;

/* Set type. */
dictionaryRefresh.type = RDM_DICTIONARY_FIELD_DEFINITIONS;

/* Set the dictionary. */
dictionaryRefresh.pDictionary = &dataDictionary;

/* Set serviceId. */
dictionaryRefresh.serviceId = 273;

/* Set verbosity. */
dictionaryRefresh.verbosity = RDM_DICTIONARY_NORMAL;

do
{
    /* (Represents the application getting a new buffer to encode the message into.) */
    getNextEncodeBuffer(&msgBuffer);

    /* Clear the encode iterator, set its RWF Version, and set it to a buffer for encoding into. */
    rsslClearEncodeIterator(&encodeIter);
    ret = rsslSetEncodeIteratorRWFVersion(&encodeIter, channelMajorVersion, channelMinorVersion);
    ret = rsslSetEncodeIteratorBuffer(&encodeIter, &msgBuffer);

    /* Encode the message. This will return RSSL_RET_DICT_PART_ENCODED if it only a part
    * was encoded. We must keep encoding the message until RSSL_RET_SUCCESS is returned. */
    ret = rsslEncodeRDMMsg(&encodeIter, (RsslRDMMsg*)&dictionaryRefresh, &msgBuffer.length,
        &rsslErrorInfo);
} while (ret == RSSL_RET_DICT_PART_ENCODED);

```

Code Example 29: Dictionary Refresh Encoding Example

8.5.6.5 Decoding a Dictionary Refresh

```

/* The decoder may require additional space to store things such as lists. */
char memoryArray[4096];
RsslBuffer memoryBuffer = { 4096, memoryArray };

RsslDecodeIterator decodeIter;
RsslMsg msg;
RsslRDMMsg rdmMsg;
RsslRDMDictionaryRefresh *pDictionaryRefresh;
RsslInt32 dictionaryTypeForThisStreamId = 0;

RsslDataDictionary dataDictionary;

rsslClearDataDictionary(&dataDictionary);

do
{
    /* (Represents the application getting the next buffer to decode.) */
    getNextDecodeBuffer(&msgBuffer);

    /* Reset our memory buffer. */
    memoryBuffer.length = 4096;
    memoryBuffer.data = memoryArray;

    /* Clear the decode iterator, set its RWF Version, and set it to the encoded buffer. */
    rsslClearDecodeIterator(&decodeIter);
    ret = rsslSetDecodeIteratorRWFVersion(&decodeIter, channelMajorVersion, channelMinorVersion);
    ret = rsslSetDecodeIteratorBuffer(&decodeIter, &msgBuffer);

    /* Decode the message to an RsslMsg structure and RsslRDMMsg structure. */
    ret = rsslDecodeRDMMsg(&decodeIter, &msg, &rdmMsg, &memoryBuffer, &rsslErrorInfo);

    if (ret == RSSL_RET_SUCCESS && rdmMsg.rdmMsgBase.domainType == RSSL_DMT_DICTIONARY &&
        rdmMsg.rdmMsgBase.rdmMsgType == RDM_DC_MT_REFRESH)
    {
        /* The message we decoded is an RsslRDMDictionaryRefresh. */
        pDictionaryRefresh = &rdmMsg.dictionaryMsg.refresh;

        /* Print if request is streaming. */
        if (pDictionaryRefresh->flags & RDM_DC_RFF_SOLICITED)
            printf("Refresh is solicited.\n");

        /* Print info if present. If the dictionary is split into parts, this is normally only present
        * on the first part. */
        if (pDictionaryRefresh->flags & RDM_DC_RFF_HAS_INFO)
        {
            /* Remember the dictionary type for this stream since subsequent parts will not indicate it.
            */
            dictionaryTypeForThisStreamId = pDictionaryRefresh->type;
        }
    }
}

```

```

    /* Print version. */
    printf("Version: %.*s\n", pDictionaryRefresh->version.length, pDictionaryRefresh->
        version.data);

    /* Print dictionary ID. */
    printf("Dictionary ID: %lld\n", pDictionaryRefresh->dictionaryId);
}

/* Print serviceId. */
printf("Service ID: %u\n", pDictionaryRefresh->serviceId);

/* Print verbosity. */
printf("Verbosity: %u\n", pDictionaryRefresh->verbosity);

/* Print dictionary name. */
printf("Dictionary Name: %.*s\n", pDictionaryRefresh->dictionaryName.length,
    pDictionaryRefresh->dictionaryName.data);

if (dictionaryTypeForThisStreamId == RDM_DICTIONARY_FIELD_DEFINITIONS)
{
    /* Decode the dictionary content into the RsslDataDictionary structure. */
    rsslClearDecodeIterator(&decodeIter);
    ret = rsslSetDecodeIteratorRWFVersion(&decodeIter, channelMajorVersion,
        channelMinorVersion);
    ret = rsslSetDecodeIteratorBuffer(&decodeIter, &pDictionaryRefresh->dataBody);
    ret = rsslDecodeFieldDictionary(&decodeIter, &dataDictionary, RDM_DICTIONARY_NORMAL,
        &errorText);
}
}
} while(!(pDictionaryRefresh->flags & RDM_DC_RFF_IS_COMPLETE));

```

Code Example 30: Dictionary Refresh Decoding Example

8.6 RDM Queue Messages

The Queue Messaging domain model is a series of message constructs that you use to interact with a Queue Provider. A Queue Provider can persist content for which users want to have guaranteed delivery and can also help send content to destinations with which users cannot directly communicate.

8.6.1 Queue Data Message Persistence

When opening a queue messaging stream with a queue provider, using a persistence file can guarantee delivery of messages sent by the OMM consumer on that queue stream. The queue file will be named after the name of the queue stream (as specified in the **RsslRDMQueueRequest** message that opened the stream). When the consumer submits **RsslRDMQueueData** messages, the consumer stores these messages in the persistence file in case the tunnel stream to the queue provider is lost and reconnected. As **RsslRDMQueueAck** messages are received from the queue provider, space in the persistence file is freed for additional messages. If at any time the application submits an **RsslRDMQueueData** message but the persistence file has no room for it, the application receives the **RSSL_RET_PERSISTENCE_FULL** return code.

The **RsslClassOfService.guarantee.persistLocally** option (set when opening the tunnel stream) specifies whether to create and maintain persistence files. The location for storage of persistent files is specified by the **RsslClassOfService.guarantee.persistenceFilePath** option. For more information on these options, refer to Section 6.9.3

NOTE: LSEG recommends that the **RsslClassOfService.guarantee.persistenceFilePath** be set to a local storage device.

If a particular queue stream is no longer needed, the user may delete the persistence file that carries the associated queue stream's name.



WARNING! If you delete a persistence file that stores messages that were not successfully transmitted, the messages will be lost.

8.6.2 Queue Request

The OMM application encodes and sends a **Queue Request** message to a Queue Provider to open a user queue. By opening a queue with an **RsslRDMQueueRequest**, the user receives any content previously sent to and persisted on a Queue Provider. To send content to another user's queue, a user must first open their own queue.

MEMBER	DESCRIPTION
rdmMsgBase	Required. Specifies the message type (i.e., RDM_QMSG_MT_REQUEST) and contains general message information, including the stream's ID (streamId) and domain type (domainType).
sourceName	Required. Specifies the name of the queue you want to open.

Table 197: RsslRDMQueueRequest Members

8.6.3 Queue Refresh

A Queue Provider encodes and sends a **Queue Refresh** message to OMM applications to inform users about queue open requests and give state information pertaining to specific queue refresh request attempts.

MEMBER	DESCRIPTION
rdmMsgBase	Required. Sets the message type (i.e., RDM_QMSG_MT_REFRESH) and contains general message information, including the stream's ID (streamId) and domain type (domainType).
sourceName	Required. Specifies the name of a queue you want to open, which should match the sourceName specified in the initial queue request.
state	Required. Indicates the state of the queue. <ul style="list-style-type: none"> States of Open and Ok indicate the queue was successfully opened. Other state combinations indicate an issue, for which additional code and text provide supplemental information. For more information on RsslState , refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
queueDepth	Required. Indicates how many Queue Data or Queue Data Expired messages are inbound on this Queue Stream.

Table 198: RsslRDMQueueRefresh Members

8.6.4 Queue Status

A Queue Provider encodes and sends **Queue Status** messages to OMM applications, conveying state information about a user's queue.

MEMBER	DESCRIPTION
flags	Required. Indicates the presence of optional queue status members. flags has only one enumeration: HAS_STATE , which indicates the presence of the state member. If flags is absent (or has no value), any previously conveyed state continues to apply.
rdmMsgBase	Required. Sets the message type (i.e., RDM_QMSG_MT_STATUS) and contains general message information, including the stream's ID (streamId) and domain type (domainType).
state	Indicates the state of the queue: <ul style="list-style-type: none"> States of Open and Ok indicate the queue is in a good state. Other state combinations indicate an issue, for which additional code and text provide supplemental information. For more information on RsslState , refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .

Table 199: RsslRDMQueueStatus Members

8.6.5 Queue Close

An OMM application encodes and sends a **Queue Close** message to a Queue Provider, closing the user's queue.

MEMBER	DESCRIPTION
rdmMsgBase	Required. Sets the message type (i.e., RDM_QMSG_MT_CLOSE) and contains general message information, including the stream's ID (streamId) and domain type (domainType).

Table 200: RsslRDMQueueClose Members

8.6.6 Queue Data

Both OMM applications and queue providers can send and receive **Queue Data** messages, which exchange data content between queue users and also communicate whether content was undeliverable.

8.6.6.1 Queue Data Members

MEMBER	DESCRIPTION
containerType	Required. Indicates the type of contents in this queue data message.
destName	Required. Specifies the name of the queue to which content is sent.
encDataBody	Optional. <ul style="list-style-type: none"> If sending a message, populate encDataBody with pre-encoded content. If sending a message without pre-encoded contents, you can use the encoding methods described in Section 8.6.6.4. If receiving a message, encDataBody can be used to access payload contents for decoding.
flags	Required. Specifies any flags that indicate more information about this message. For further details on available flags, refer to Section 8.6.6.2. flags is only for decoding, and OMM consumer applications do not need to set it.
identifier	Required. A user-specified unique identifier for the message being sent. identifier is used when acknowledging this content via a Queue Ack message.
queueDepth	Required. Indicates the number of Queue Data or Queue Data Expired messages still inbound on this queue stream, following this message. queueDepth is only for reading, and OMM consumer applications do not need to set it.
rdmMsgBase	Required. Sets the message type (i.e., RDM_QMSG_MT_DATA) and contains general message information, including the stream ID (streamId) and domain type (domainType).
sourceName	Required. Specifies the name of the queue from which content is sourced, which should match the sourceName specified in the Queue Request for this substream.
timeout	Optional. Specifies the desired timeout for this content (which can be any of the RssIRDMQueueTimeCodes in Section 8.6.6.3 or a specific time interval in milliseconds). If a timeout value expires during the course of delivery, the content is returned as an RssIRDMQueueDataExpired message. If not specified, this defaults to QueueMsgTimeoutCodes.INFINITE (i.e., the content never times out).

Table 201: RssIRDMQueueData Members

8.6.6.2 Queue Data Flag

RssIRDMQueueData messages and **RssIRDMQueueDataExpired** messages use the following flag:

FLAG	DESCRIPTION
RDM_QMSG_DF_POSSIBLE_DUPLICATE== 0x1	Indicates that the message was retransmitted and that the application might have already received it.

Table 202: Queue Data Flag

8.6.6.3 Queue Message Timeout Codes

Queue message timeout codes are special codes that can be set on the `RsslRDMQueueData.timeout` member to specify timeout behavior.

ENUMERATION	DESCRIPTION
RDM_QMSG_TC_INFINITE	This message persists in the system for an infinite amount of time.
RDM_QMSG_TC_IMMEDIATE	This message immediately times out if any portion of its delivery path is unavailable.
RDM_QMSG_TC_PROVIDER_DEFAULT	This message persists in the system for a duration set by the provider.

Table 203: RsslRDMQueueTimeoutCodes

8.6.6.4 Queue Data Encoding

The `RsslRDMQueueData` message allows users to encode both OMM and non-OMM/opaque content. There are several methods available to help with encoding.

FUNCTION NAME	DESCRIPTION
<code>rsslEncodeRDMQueueMsg</code>	When sending no payload or payload content is preencoded and specified on the <code>RsslRDMQueueData.encDataBody</code> buffer, this method encodes the <code>RsslRDMQueueData</code> message in a single call.
<code>rsslEncodeRDMQueueMsgComplete</code>	Completes the content encoding into this <code>RsslRDMQueueData</code> message.
<code>rsslEncodeRDMQueueMsgInit</code>	<p>Begins the process of encoding content into this <code>RsslRDMQueueData</code> message. This method takes an <code>EncodeIterator</code> as a parameter, where the <code>EncodeIterator</code> is associated with the buffer into which content is encoded.</p> <p>When this method returns, users should call additional methods required to encode the content. After all remaining encoding is completed, call the <code>encodeComplete</code> method.</p>

Table 204: Queue Data Message Encoding Methods

8.6.6.5 Queue Data Message Encoding Code Sample

```

RsslEncodeIterator _msgEncIter;
RsslRDMQueueData _queueData;

// initialize the QueueData encoding
rsslClearRDMQueueData(&_queueData);
_queueData.rdmMsgBase.streamId = QUEUE_MSG_STREAM_ID;
_queueData.identifier= 124;
_queueData.sourceName.data = "MY_QUEUE";
_queueData.sourceName.length = 8;
_queueData.destName.data = "DESTINATION_QUEUE";
_queueData.destName.length = 17;
_queueData.timeout = RDM_QMSG_TC_INFINITE;
_queueData.containerType = RSSL_DT_FIELD_LIST;

_msgEncIter.clear();
rsslClearEncodeIterator(&_msgEncIter);
rsslSetEncodeIteratorRWFVersion(&_msgEncIter, pTunnelStream-
    >classOfService.common.protocolMajorVersion, pTunnelStream-
    >classOfService.common.protocolMinorVersion);
rsslSetEncodeIteratorBuffer(&_msgEncIter, buffer);

// begin encoding content into RsslRDMQueueData message
if ((ret = rsslEncodeRDMQueueMsgInit(&_msgEncIter, &_queueData, &error)) < RSSL_RET_SUCCESS)
{
    printf("rsslEncodeRDMQueueMsgInit() failed");
    return;
}

// Start Content Encoding - follow standard field list encoding as shown in the
// Transport API C Edition Developers Guide examples.
// When content encoding is done, complete the RsslRDMQueueData encoding

if ((ret = rsslEncodeRDMQueueMsgComplete(&_msgEncIter, RSSL_TRUE, &buffer->length &error)) <
    RSSL_RET_SUCCESS)
{
    printf("rsslEncodeRDMQueueMsgComplete() failed");
    return;
}

```

Code Example 31: Queue Data Message Encoding Example

8.6.7 QueueDataExpired

If queue data messages sent on a queue stream cannot be successfully delivered, the queue provider sends **RsslRDMQueueDataExpired** messages on the queue stream to OMM consumer applications.

OMM consumer applications do not send this message.

8.6.7.1 RsslRDMQueueDataExpired Structure Members

MEMBER	DESCRIPTION
containerType	Required. Indicates the type of contents in the message.
destName	Required. destName specifies the name of the queue from which content is sourced (i.e., the value of sourceName as set in the original RsslRDMQueueData message).
encDataBody	Optional. Contains the payload contents (if any) of the original Queue Data message.
flags	Required. flags indicate more information about this message. For details, refer to Section 8.6.6.2.
identifier	Required. A user-specified, unique identifier for the message (which is the same as the identifier from the original RsslRDMQueueData message).
queueDepth	Required. Indicates how many Queue Data or Queue Data Expired messages are still inbound on this queue stream (following this message).
rdmMsgBase	Required. Specifies the queue message type (i.e., RDM_QMSG_MT_DATA_EXPIRED) and contains general message information, including the stream's ID (streamId) and domain type (domainType).
sourceName	Required. sourceName specifies the name of the queue to which content was sent (i.e., the value of destName as set in the original RsslRDMQueueData message).
undeliverableCode	Required. Specifies a code explaining why the content was undeliverable. For more information on undeliverable codes and their meanings, refer to Section 8.6.7.2.

Table 205: RsslRDMQueueDataExpired Structure Members

8.6.7.2 Queue Message Undeliverable Codes

Undeliverable codes are used in the **QueueDataExpired.undeliverable** member, and specify why the message could not be delivered.

ENUMERATION	REASON FOR DELIVERY FAILURE
RDM_QMSG_UC_EXPIRED	Indicates that the timeout value specified for this message has expired.
RDM_QMSG_UC_INVALID_SENDER	Indicates that the sender of this message has now become invalid.
RDM_QMSG_UC_INVALID_TARGET	Indicates that the specified destination of this message does not exist.
RDM_QMSG_UC_MAX_MSG_SIZE	Indicates that the message was too large.
RDM_QMSG_UC_NO_PERMISSION	Indicates that the source/sender of this message is not permitted to send or is not permitted to send to the specified destination.
RDM_QMSG_UC_QUEUE_DISABLED	Indicates that the specified destination of this message has a disabled queue.

Table 206: RsslRDMQueueDataUndeliverableCodes

ENUMERATION	REASON FOR DELIVERY FAILURE
RDM_QMSG_UC_QUEUE_FULL	Indicates that the specified destination of this message has a full queue and cannot receive any additional content.
RDM_QMSG_UC_TARGET_DELETED	Indicates that the target queue was deleted before the message was delivered.
RDM_QMSG_UC_UNSPECIFIED	Indicates that the delivery failed for unspecified reasons.

Table 206: `RsslRDMQueueDataUndeliverableCodes` (Continued)

8.6.8 Queue Ack

A Queue Provider encodes and sends a **Queue Ack** message to OMM applications, acknowledging that a Queue Data message is persisted on the Queue Provider. After a Queue Provider acknowledges persistence, the application no longer needs to persist the acknowledged content.

MEMBER	DESCRIPTION
destName	Optional. Specifies the name of the queue from which content is sourced (i.e., the value of sourceName as set in the original Queue Data message).
identifier	Required. The identifier of the message being acknowledged. This should match the RsslRDMQueueData.identifier for the message being acknowledged.
sourceName	Required. Specifies the name of the queue to which content was originally sent (i.e., the value of destName as set in the original Queue Data message).
rdmMsgBase	Required. Sets the message type (i.e., RDM_QMSG_MT_ACK) and contains general message information, including the stream's ID (streamId) and domain type (domainType).

Table 207: `RsslRDMQueueAck`

9 Warm Standby Feature

9.1 Overview

The Warm Standby feature is implemented at the Value Add Watchlist layer of Enterprise Transport API and is client-side feature. This feature works by providing the application the capability to failover from an active to one or more standby server(s) in the event that the primary/active fails. Application must configure the active and standby servers to use this API feature. After the connections are established with the provided servers which form a Warm Standby group, the client-side or consumer sends messages to the standby server connections to change their mode to Standby. Requested items are opened on all servers by the consumer but the active server responds with messages such as refresh, updates, status, etc. to the consumer. Standby servers respond with blank/empty refreshes. When primary fails, consumer notifies the next server in standby list that it is now Active. The new active server responds with refresh as needed resumes updates for all open items. This process of cut-over is transparent to the application.

The watchlist feature must be enabled to use the Warm Standby feature. A server qualifies to be a standby only if it advertises support for Warm Standby, supports similar features over login and offers an identical service (supported domains, quality of service, etc.) as the active server.

Warm Standby not only reduces overall recovery time, but also network traffic by not inducing a “packet storm” with a flurry of re-requests to a standby server. Because the standby server is already aware of items an application has subscribed for, during a failover Enterprise Transport API does not need to re-subscribe open items between a provider and consumer.

9.2 Warm Standby Modes

The Enterprise Transport API Value Add layer supports two Warm Standby modes:

- Login based Warm Standby
- Service based Warm Standby

The login based Warm Standby uses the connection lost event to switch from a primary server to a standby server from the standby server list. The service based Warm Standby uses the service down event OR connection lost event to switch all subscribe items from a primary service to a standby service.

The service based Warm Standby mode offers better resiliency than the login based mode as it can switch from primary to standby if an upstream service is down but the connection to both servers remains intact. A particular server may be the primary for one service and standby for another service as a result. This ability to failover in the event of service down or channel down events makes the service based Warm Standby the recommended mode.

The following figure illustrates the sequence of events when using the Login Based Warm Standby feature:

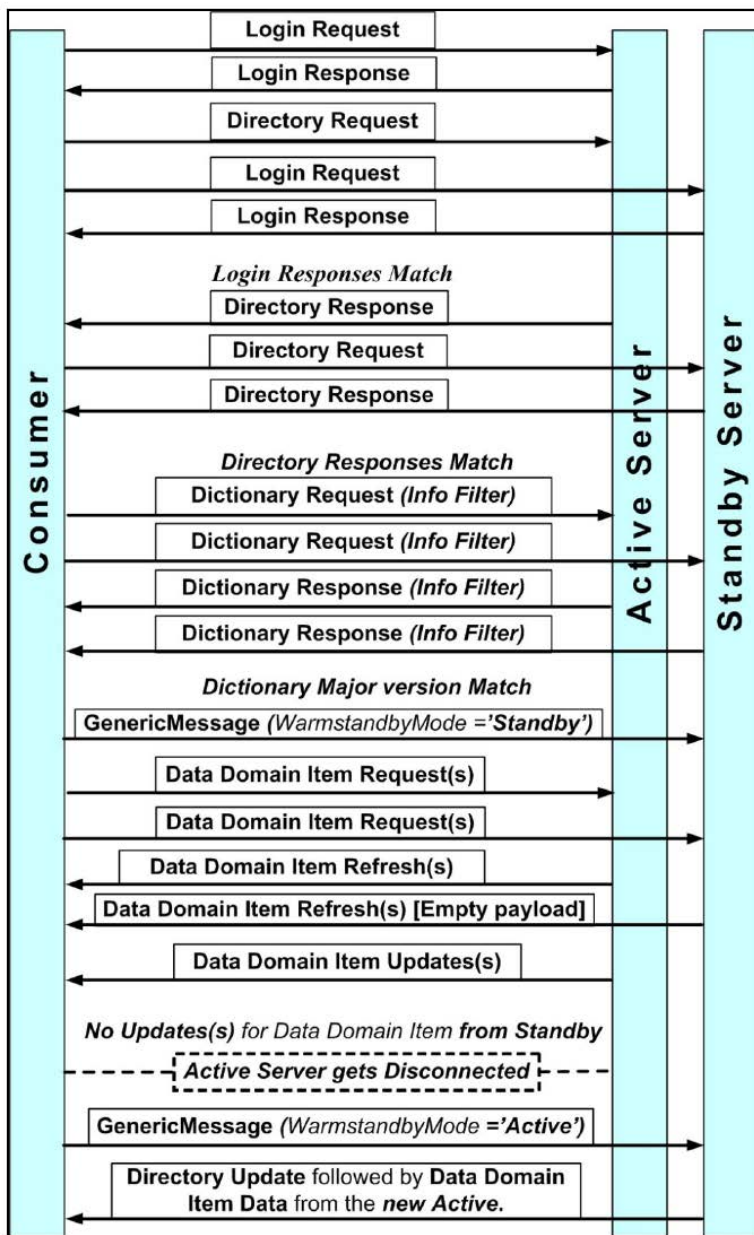


Figure 12. Login Based Warm Standby Order of Events in a Cutover from Active to Standby

The following figure illustrates the sequence of events when using the Service Based Warm Standby feature:

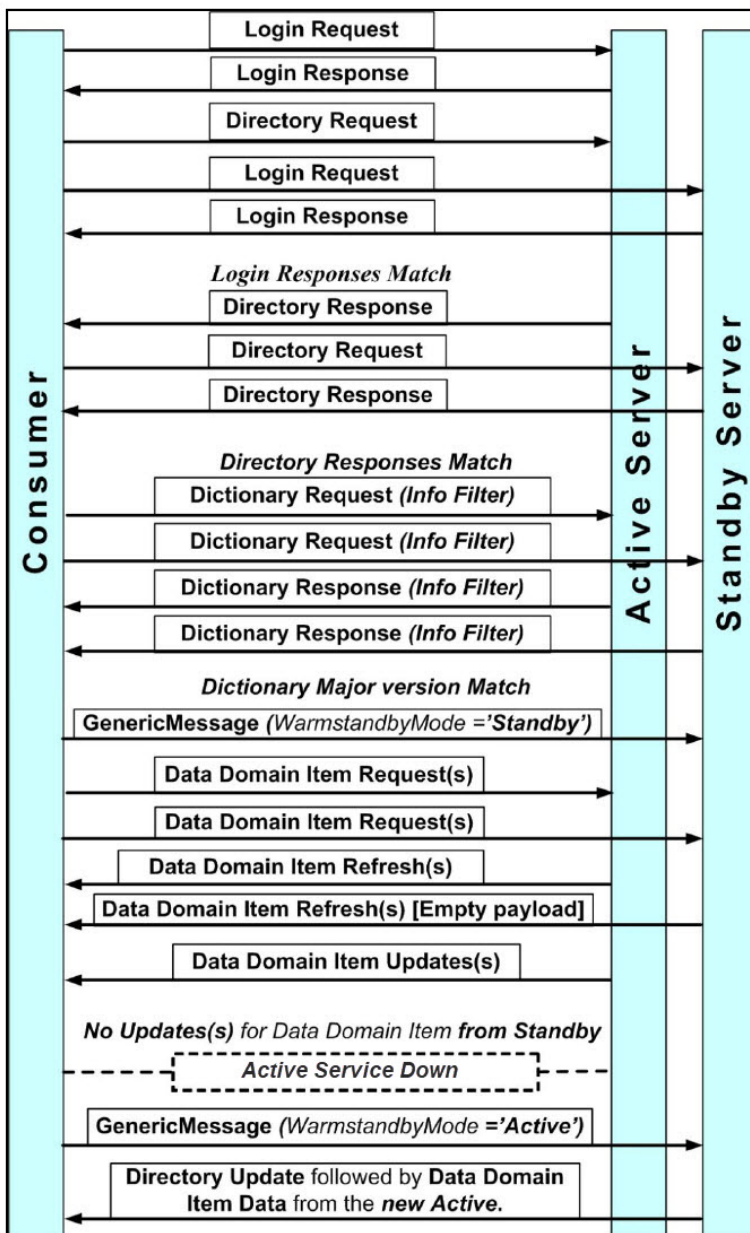


Figure 13. Service Based Warm Standby Order of Events in a Cutover from Active to Standby

9.3 Warm Standby Configuration and Feature Details

Application written to the Enterprise Transport API VA layer with watchlist enabled may be configured to use a particular stand by mode. Application must configure a group of servers, or Warm Standby group in which a starting active server and one or more standby servers are specified. Using this configuration, the Enterprise Transport API library will internally request the same items from both active and standby servers.

For the sample code for specifying a starting active server and a standby server, see Section 9.3.1.

The consumer application receives updates only on the item streams opened on the active server or active service; it does not get updates, status, unsolicited refreshes, and Generic Messages from the standby servers(s) or standby service(s). If the active server or active service is down, Enterprise Transport API notifies one of the standby servers that it is a new active for the requested service. The new active then begins sending data without the consumer application needing to re-request the items. After cut-over, the new active responds by sending

any conflated updates containing all changed data for conflatable domains and sending unsolicited refresh messages followed by updates for any non-conflated domains. This process brings active data streams back to their pre-failover state.

If the failed server or service comes back online, Enterprise Transport API does not switch back to it. It becomes one of the servers in the standby list. Internally, the Enterprise Transport API consumer-side sends a *ConsumerConnectionStatus GenericMsg* on the Login domain for login based and on the Directory domain for service based to indicate to the provider applications whether they should operate as an active server or service based on the Warm Standby mode. Refer to the *RDM Usage Guide* for more details on this message on the Login and Directory domains.

After establishing a connection to the standby server, Enterprise Transport API internally obeys the following rules for Warm Standby:

- The Standby server's login response must match the following login elements and state support for standby and the same standby mode as active server:
 - **ApplicationId**
 - **PositionId**
 - **ProvidePermissionProfile**: If requested by the user, the values received from all servers must match what was requested.
 - **ProvidePermissionExpressions**: If requested by the user, the values received from all servers must match what was requested.
 - **SingleOpen**
 - **AllowSuspectData**
 - **SupportPost**
 - **SupportBatchRequests**
 - **SupportOMMPost**
 - **SupportStandby**
 - **SupportViewRequests**

If any of these elements does not match, then Enterprise Transport API disconnects the standby server.

- The standby server's Directory response must match all attributes except **Vendor** and **IsSource** of the Directory Info Filter for services common to both active and standby servers. Otherwise Enterprise Transport API disconnects the standby server.
- The standby server's Dictionary response must match the major version. Otherwise Enterprise Transport API disconnects the standby server.
- The provider application can optionally send the **SupportStandbyMode** element in login response to advertise supported Warm Standby mode to consumer applications. Enterprise Transport API disconnects the server if the mode does not match with the configured Warm Standby mode on consumer side.
- Enterprise Transport API routes Generic Messages only on streams in administrative domains. Generic Message on data streams is not routed to the standby server(s).
- Post Message is routed to the active and all standby servers to ensure data on all servers is synchronized.

If the active server or service fails when switching to the new active server or service, Enterprise Transport API does the following:

- For all services that were offered by the old active server but are not offered by the new active server, Enterprise Transport API sends a directory update message stating that the service is down for login based.
- For an item on the failed server but no longer open on the newly active server, Enterprise Transport API sends an internally-generated status message with the *ClosedRecover* stream state to the consumer application.

9.3.1 Configuration Example for a Starting Server and a Standby Server

```

RsslReactorWarmStandbyGroup      reactorWarmStandbyGroup;
RsslReactorWarmStandbyServerInfo  standbyServerInfo;
RsslReactorConnectOptions        reactorConnectOpts;

reactorWarmStandbyGroup.startingActiveServer.reactorConnectInfo.rsslConnectOptions.connectionType =
    RSSL_CONN_TYPE_SOCKET;
reactorWarmStandbyGroup.startingActiveServer.reactorConnectInfo.rsslConnectOptions.connectionInfo.unified.address = "dataserver1";
reactorWarmStandbyGroup.startingActiveServer.reactorConnectInfo.rsslConnectOptions.connectionInfo.unified.serviceName = "14002";

standbyServerInfo.reactorConnectInfo.rsslConnectOptions.connectionType = RSSL_CONN_TYPE_SOCKET;
standbyServerInfo.reactorConnectInfo.rsslConnectOptions.connectionInfo.unified.address =
    "dataserver2";
standbyServerInfo.reactorConnectInfo.rsslConnectOptions.connectionInfo.unified.serviceName = "14002";

reactorWarmStandbyGroup.standbyServerCount = 1;
reactorWarmStandbyGroup.standbyServerList = &standbyServerInfo;
reactorWarmStandbyGroup.warmStandbyMode = RSSL_RWSB_MODE_SERVICE_BASED;
reactorConnectOpts.warmStandbyGroupCount = 1;
reactorConnectOpts.reactorWarmStandbyGroupList = &reactorWarmStandbyGroup;

```

Example 32: Warm Standby Configuration Example

10 Preferred Host Feature

The Preferred Host feature adds the ability for Consumer applications to select a specific connection or warm standby group as the preferred connection, and, if configured, the Reactor will switch back to that connection or warm standby group on either a timer, or directly from a function call. This allows an application to switch to a lower cost provider during periods of low traffic, or to switch back to a preferred provider in the case that a previous connection was interrupted, and the Reactor has moved to a secondary connection. During the Preferred Host operation, the reactor will not close the currently active connections, and the application will continue to get data via dispatch.

This feature works with or without the watchlist, and also supports Warm Standby configurations.

For Warm Standby functionality, the Preferred Host feature has the option to do the fallback strictly within a currently connected Warm Standby Group, allowing applications to effectively reset their requests to their configured preferred data providers.

All Preferred Host options can be modified during runtime via **`rsslReactorChannelIoctl`** calls.

The Preferred Host and Warm Standby group are configured via configuration parameters **`RsslPreferredHostOptions`** before making a connection for **`RsslReactorChannel`**, see Section 6.4.

This feature is not supported with Non-Interactive Providers and Interactive Providers.

10.1 Preferred Host Operation Triggers

The Preferred Host feature, when enabled, can be triggered either through configuring a timer or via direct function call.

10.1.1 Function Call

When the application directly calls **`rsslReactorFallbackToPreferredHost`**, the Reactor will immediately start a Preferred Host operation. This will return an error if Preferred Host is not enabled on the channel. If a Preferred Host operation is currently in progress, this will immediately return success. If the Reactor Channel is currently connected to the configured Preferred Host channel or Warm Standby group, a **`PREFERRED_HOST_COMPLETE`** channel event will be queued up for dispatch, indicating that the operation is complete. If the channel is currently in reconnection, or is switching Warm Standby Groups, this call will return success, but will not start a Preferred Host operation.

10.1.2 Timer Trigger

The Preferred Host configuration allows applications to specify either a specific time of day/times of day using a cron string format, or a specific interval can be specified using the **`RsslPreferredHostOptions`** structure. The cron time of day configuration has higher priority over the interval configuration.

When the timer fires, it will check to see if the channel is currently on the configured Preferred Host channel or Warm Standby group. If so, a **`PREFERRED_HOST_COMPLETE`** channel event will be queued up for dispatch, indicating that the operation is complete. Otherwise, the Preferred Host operation will continue.

For the interval configuration, the timer will start whenever the channel is connected, and will be reset upon any reconnection succeeding or if **`RsslReactorChannelIoctl`** changes the interval configuration.

10.2 Preferred Host Reconnection Behavior Changes

When Preferred Host is enabled, the reconnection order will be changed to attempt the configured Preferred Host connection and Warm Standby group (if enabled) more aggressively by alternating between a configured preferred connection and a non-preferred connection. For more information about the specific ordering and differences from non-Preferred Host reconnection, see Section 6.4.3.

10.3 Preferred Host Operation Steps

When the Preferred Host operation is triggered (either by function call or timer), and Preferred Host is enabled for the Reactor Channel, the following checks and steps will occur. The Application can continue to dispatch and receive data from the current connection(s) while the Preferred Host operation is occurring.

If the Reactor Channel is currently on a Warm Standby group and the fallback within Warm Standby group option is enabled:

1. The fallback within a warm standby group will occur, and the Reactor Channel will not attempt to connect to a different Warm Standby group.

2. Upon dispatching, the Reactor will do the following operation based on the Warm Standby group's configuration:
 - a) If the current Warm Standby group is Login-based:
 - If the starting server connection is active and not the current active connection for the Warm Standby group, the Reactor will swap the current active server to the starting server connection. The application may see unsolicited refreshes to re-synchronize the item streams.
 - b) If the current Warm Standby group is service-based:
 - The Reactor will iterate through the configuration of the Warm Standby group, starting with the starting server, and going through each connection defined in the secondary server list. For each service name defined in each connection's `RsslReactorWarmStandbyServerInfo.perServiceBasedOptions.serviceNameList`, if the service name is in an ACTIVE state on that connection, that connection will become the ACTIVE for that service, and the previous active will become STANDBY.

NOTE: If a service name is defined multiple times in the Warm Standby group, the first time the service name is found on a connection as ACTIVE, it will switch the current ACTIVE service, and the service will not be switched for any subsequent connections.

- Any services that are not defined in the configuration will be ignored by this operation, and the current ACTIVE for those services will not be changed.

If the Reactor Channel is currently on the preferred channel, or if Warm Standby is enabled, on the preferred Warm Standby group:

1. The Reactor will dispatch a **PREFERRED_HOST_COMPLETE** event, and the Preferred Host operation will be finished.

If Warm Standby is enabled for the Reactor Channel.

1. The Reactor worker thread will attempt to establish a connection to the configured preferred Warm Standby group's starting connection.
2. Once that is established, the worker thread will send a channel event to be dispatched. This event will:
 - a) Close out all of the currently active secondary connections and the starting connection, and the application should expect to see **FD_CHANGE** events as well as Watchlist generated item and service state changes.
 - b) The Reactor will then send a **DOWN_RECONNECTING** channel event, indicating that the Warm Standby group is fully closed and that it is switching to the preferred Warm Standby group.
 - c) The Reactor will then swap the underlying transport channels, and signal the user by sending **CHANNEL_UP** and **CHANNEL_READY** events.
 - d) The Reactor will send a **PREFERRED_HOST_COMPLETE** event to the application.
 - e) When the reactor gets login and directory responses, it will connect to the configured secondary servers in the Warm Standby group.
3. If the connection attempt fails, the Reactor will send a **PREFERRED_HOST_COMPLETE** event to the application. The current connections will remain and any data that is flowing will continue to flow.

If Warm Standby is not enabled:

1. The Reactor worker thread will attempt to establish a connection to the configured preferred connection in the connection list.
2. Once that is established, the worker thread will send a channel event to be dispatched. This event will:
 - a) Send a **DOWN_RECONNECTING** channel event to the application, and if the watchlist is enabled, any Watchlist generated item and service state changes.
 - b) The Reactor will then swap the underlying transport channels, and signal the user by sending **CHANNEL_UP** and **CHANNEL_READY** events.
 - c) The Reactor will send a **PREFERRED_HOST_COMPLETE** event to the application, after the channel is fully closed by the Worker thread.
 - d) If the watchlist is enabled, it will handle all administrative messages and will re-request any open items.

10.4 Preferred Host Private Stream and Tunnel Stream Handling

If the Preferred Host operation switches connections, Private Steams and Tunnel Streams will be closed, and may need to be reopened.

10.5 Preferred Host IOCTL Functionality

RsslReactorChannelIoctl allows consuming applications to change the current Preferred Host configuration. If a Preferred Host operation is currently in progress when this is called, the configuration change will be applied after the operation finishes. For details, see Section 6.4.2.1.

11 Value Added Utilities

11.1 Utility Overview

The Value Added Utilities are a collection of helper constructs, mainly used by the Enterprise Transport API Reactor. Included is a multi-purpose memory buffer type that can help with flexible, reusable memory - this is leveraged by the Administration Domain Model Representations when encoding or decoding messages. Other Value Added Utilities include a simple queue, mutex locks, thread helper functionality, and a simple event alerting component.

Only the Memory Buffer utility is described in this document as it is leveraged by the provided example applications. The other Value Added Utilities are internally leveraged by the Enterprise Transport API Reactor so applications need not be familiar with their use.

11.2 Memory Buffer

The Memory Buffer utilities provide a simple method to apportion space from a block of memory space. This allows for the creation of complex objects without expensively requesting and releasing memory from the operating system. This also allows for easy reuse of the memory block. The memory is provided to the functions via an **RsslBuffer** that has its **data** member set to the memory block and **length** member indicating the byte length of the memory.

FUNCTION NAME	DESCRIPTION
<code>rsslReserveBufferMemory</code>	Reserves memory from an RsslBuffer . The buffer passed in is modified to point to the unused portion of the memory block. Subsequent calls reserve adjacent memory, so this can be called multiple times to generate a C-style array of objects without knowing the full length in advance.
<code>rsslReserveAlignedBufferMemory</code>	Reserves memory from an RsslBuffer . Similar to <code>rsslReserveBufferMemory</code> , but will ensure that the memory is aligned on an appropriate word boundary. Subsequent calls to the non-aligned <code>rsslReserveBufferMemory</code> will reserve adjacent memory, so this can be called multiple times to generate a C-style array of objects without knowing the full length in advance.
<code>rsslCopyBufferMemory</code>	Requires an input RsslBuffer , output RsslBuffer , and an RsslBuffer pointing to an available memory block. Sets the output buffer to a deep copy of the input RsslBuffer , using the space provided by the memory block.

Table 208: Memory Buffer Functions

11.3 Using the Memory Buffer

The following example reserves an aligned block of memory to represent an array of five user-defined **MyStruct** structures. The memory for the first structure is reserved and aligned. Each subsequent member of the array is then reserved in a loop, wherein memory is reserved based on the initial aligned offset. The memory associated with each **MyStruct** is initialized after it is reserved. Once completed, **myStructArray** can be accessed like an array of **MyStructs** (**myStructArray[index]**, etc.).

```
/* Represents some complex user-defined struct.
 * This example will create an array of these structs. */
typedef struct
{
    int number;
    char letter;
} MyStruct;

int i = 0;

/* The block of memory that we will use as storage of the array. */
char memoryBlock[128];
RsslBuffer memoryBuffer;

MyStruct *myStructArray, *myStructElem;

memoryBuffer.data = memoryBlock;
memoryBuffer.length = 128;

/* Create first element on a word boundary, then initialize */
myStructArray = (MyStruct*)rsslReserveAlignedBufferMemory(&memoryBuffer, 1, sizeof(MyStruct));
myStructArray->number = i;
myStructArray->letter = 'a';

for(i = 1; i < 5; ++i)
{
    /* Reserve space for subsequent elements and initialize them in. */
    myStructElem = (MyStruct*)rsslReserveBufferMemory(&memoryBuffer, 1, sizeof(MyStruct));

    myStructElem->number = i;
    myStructElem->letter = 'a';
}

/* Change the letter of MyStruct in position 2, can access just like an array */
myStructArray[2]->letter = 'b';
```

Code Example 33: Memory Buffer Example

12 Payload Cache Detailed View

12.1 Concepts

The Value Added Payload Cache component provides a facility for storing OMM containers (the message's data payload). Typical use of a payload cache is to store the current image of OMM data streams, where each entry in the cache corresponds to a single data stream. The initial content of a cache entry is defined by the payload of a refresh message. The current (or last) value of the entry is defined by the cumulative application of all refresh and update messages applied to the cache entry container. Values are stored in and retrieved from the cache as encoded OMM containers.

A cache is defined as a collection of OMM data containers. An application may create multiple cache collections, or instances, depending on how it wants to organize the data. The only restriction on cache organization is that all entries in a cache must use the same RDM Field Dictionary to define the set of field definitions it will use. At minimum, a separate cache would be required for each field dictionary in use by the application. However, because cache instances can also share the same field dictionary, partitioning is not restricted to dictionary usage. Some examples of how cache instances can be organized in an application include: all item streams on an RSSL connection; all items belonging to a particular service; all items across the entire application.

The application is responsible for organizing cache instances, managing the lifecycle of all entries in each cache, and applying and retrieving data from the cache. Figure 14 shows an example consumer type application which has created two cache instances to store data from two services on an OMM provider.

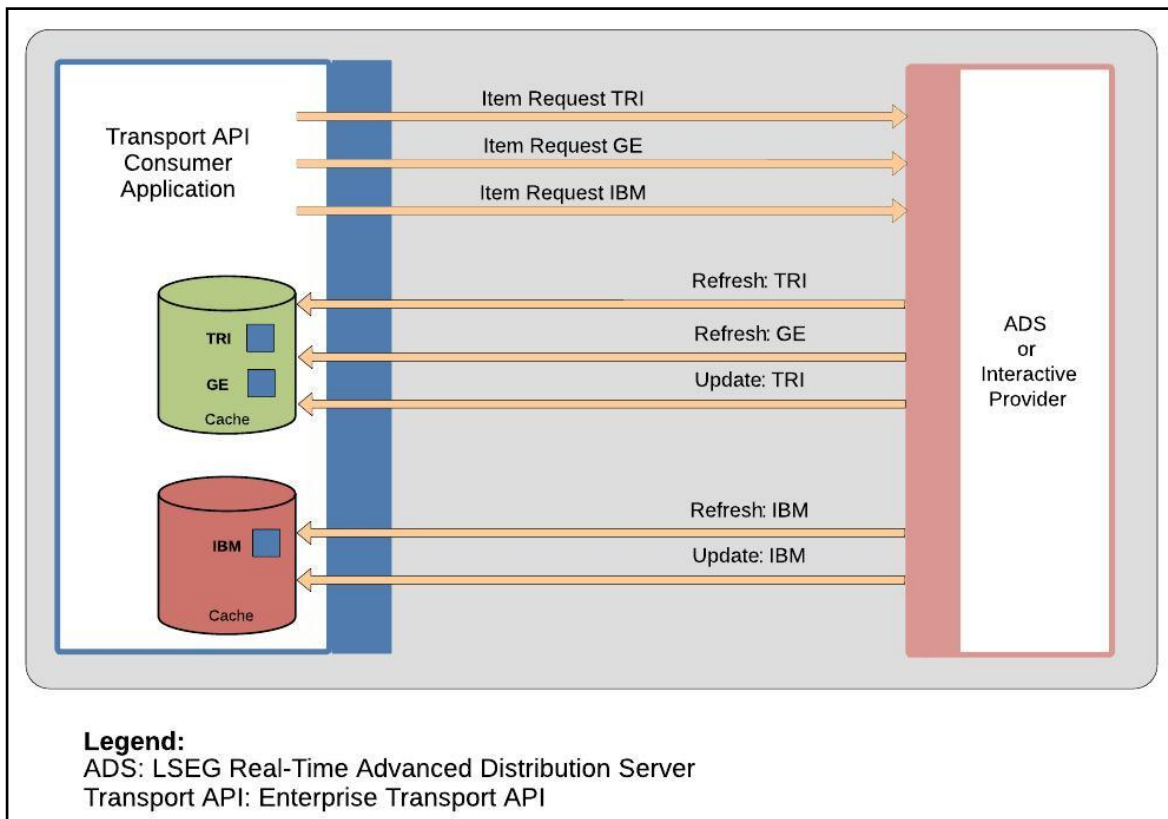


Figure 14. Consumer Application using Cache to Store Payload Data for Item Streams

12.2 Payload Cache

This section describes how the payload cache is managed (initialization and uninitialization), and how instances of cache (collections of payload entries) are created and destroyed.

12.2.1 Managing the Payload Cache

To use the Value Added Payload Cache, the application must first call the **rsslPayloadCacheInitialize** function for global static resource initialization. When the payload cache is no longer needed, the application should call **rsslPayloadCacheUninitialize** to cleanup and release all resources used by the cache.

Use the following functions to manage the cache:

FUNCTION NAME	DESCRIPTION
rsslPayloadCacheInitialize	The first function the application must call prior to using the payload cache. The method only needs to be called one time by the application, but may be called more than once. A reference count is incremented for each call to this function. An equal number of calls to rsslPayloadCacheUninitialize must be made before the component is uninitialized.
rsslPayloadCacheUninitialize	The last call an application should make when it is finished using the payload cache interface. The initialization reference count is decremented for each call to this function. Uninitialization only occurs if the initialization reference count is zero. During uninitialization, all remaining cache instances, entries, and resources will be destroyed.
rsslPayloadCacheIsInitialized	This function can be used by an application to determine if the payload cache component has already been initialized (by a call to rsslPayloadCacheInitialize).

Table 209: Payload Cache Management Functions

12.2.2 Cache Error Handling

Some of the functions on the payload cache interface use the **RsslCacheError** structure to return error information. This structure will be populated with additional information if an error occurs during the function call. The application should check the return value from functions. The application can optionally provide the **RsslCacheError** structure to obtain additional information.

12.2.2.1 Cache Error Structure Members

The **RsslCacheError** has the following structure members:

STRUCTURE MEMBER	DESCRIPTION
rsslErrorId	Specifies an error ID. The range of values is defined by the set of Enterprise Transport API return codes (from the RsslReturnCodes enumeration).
text	This char[] will contain text with additional information when a function call returns a failed result. The size of the buffer is fixed to MAX_OMM_CACHE_ERROR_TEXT as defined on the cache interface.

Table 210: RsslCacheError Structure Members

12.2.2.2 Clearing a Cache Error

The following function clears the **RsslCacheError**.

STRUCTURE MEMBER	DESCRIPTION
<code>rsslCacheErrorClear</code>	Clears the RsslCacheError structure. Use this function prior to passing the structure to a cache interface function.

Table 211: Function for Cache Error Handling

12.2.3 Payload Cache Instances

A payload cache instance is a collection of payload data containers. An empty cache instance must be created before any data can be stored in the cache. When a cache or its entries are no longer needed, it can be destroyed. For functions used to create and destroy a cache, refer to Section 12.2.3.1. Before using payload caches, you must first have initialized this function using **rsslPayloadCacheInitialize** as described in Section 12.2.1.

12.2.3.1 Managing Payload Instances

FUNCTION NAME	DESCRIPTION
<code>rsslPayloadCacheCreate</code>	Creates a payload cache instance, and returns the RsslPayloadCacheHandle . All operations on the cache require this handle. Options are passed in via the RsslPayloadCacheConfigOptions defined in Section 12.2.3.2.
<code>rsslPayloadCacheDestroy</code>	Destroys a payload cache instance. Any entries remaining in the cache are also destroyed at this time.

Table 212: Functions for Managing Cache Instances

12.2.3.2 Payload Cache Structure Member

STRUCTURE MEMBER	DESCRIPTION
<code>maxItems</code>	Sets the maximum number of entries allowed in the cache. When the maximum number of items is reached, the cache refuses new entries until existing entries are removed. The rsslPayloadEntryCreate function will return a null RsslPayloadEntryHandle when the maximum number of items is reached. When set to zero, the cache allows an unlimited number of items. Refer to Section 12.3.1.

Table 213: RsslPayloadCacheConfigOptions Structure Members

12.2.4 Managing RDM Field Dictionaries for Payload Cache

Each cache instance requires an RDM Field Dictionary, to define the set of fields that may be encoded in the OMM containers stored in the cache.

A cache is associated with a field dictionary through a binding process, which requires an **RsslDataDictionary** structure loaded with the field dictionary. The dictionary structure can be loaded from a file (using the **rsslLoadFieldDictionary** function) or from an encoded dictionary message from a provider (using the **rsslDecodeFieldDictionary** function). The cache does not use the enumerated dictionary content, so loading the enumeration dictionary is not required. For more information on using **RsslDataDictionary**, refer to the *Enterprise Transport API Reference Manual*.

After the **RsslDataDictionary** loads, it is bound to a cache instance using a key (an arbitrary string identifier assigned by the application to name the dictionary). The key allows multiple cache instances to share the same dictionary. After the first binding of a dictionary, it can be bound to additional cache instances by simply providing the same key on additional bindings. For a list of functions used in binding a dictionary to a cache, refer to Section 12.2.4.1.

The cache builds its own field definition database from the **RsslDataDictionary** definitions. After binding, the application does not need to retain the dictionary structure, because the cache does not refer to the **RsslDataDictionary** used during the binding. In typical usage, the application will likely retain the dictionary for use with other encoding and decoding operations.

NOTE: A cache can be bound to a dictionary only once during its lifetime. While a cache cannot be switched to a new dictionary, the dictionary in use can be extended with new definitions. Refer to Section 12.2.4.3.

12.2.4.1 Setting Functions

FUNCTION NAME	DESCRIPTION
<code>rsslPayloadCacheBindDictionary</code>	Deprecated. For details on using this function, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> . Otherwise, for equivalent functionality, refer to rsslPayloadCacheSetDictionary in this table.
<code>rsslPayloadCacheBindSharedDictionaryKey</code>	Deprecated. For details on using this function, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> . Otherwise, for equivalent functionality, refer to rsslPayloadCacheSetSharedDictionaryKey in this table.
<code>rsslPayloadCacheSetDictionary</code>	<p>This function sets an RsslDataDictionary to a cache instance (identified by RsslPayloadCacheHandle). Use this function the first time a dictionary is set to a cache. The application must provide a key parameter to this function to name the dictionary for future reference. This key is used in future setting operations when the application wants to share a dictionary between cache instances or to extend the definitions in the dictionary.</p> <p>The first time a particular key is used with this function will be the initial setting of that dictionary to a cache. The second time the same key is used in this function; it will reload the field definitions from the given RsslDataDictionary structure, enabling the dictionary to be extended. Refer to Section 12.2.4.3.</p>
<code>rsslPayloadCacheSetSharedDictionaryKey</code>	<p>Use this function when sharing a dictionary among multiple caches. This function sets a cache (identified by the RsslPayloadCacheHandle) to a previously set dictionary (identified by the dictionary key name). To share a dictionary, the dictionary named by the key passed to this function must have previously had an initial setting to another cache using the rsslPayloadCacheSetDictionary function.</p> <p>This function does not require the RsslDataDictionary structure, since that was already loaded during the initial setting with this dictionary key.</p>

Table 214: Functions for Setting Dictionary to Cache

12.2.4.2 Setting Example

In the following example, two cache instances are created and set to a single shared field dictionary.

```
RsslRet ret;
RsslCacheError cacheError;
RsslPayloadCacheConfigOptions cacheConfig;
const char* dictionaryKey = "SharedKey1";
char errorDataArray[256];
RsslBuffer errorBuffer = {256, &errorDataArray[0]};
RsslPayloadCacheHandle cacheHandle1 = 0;
RsslPayloadCacheHandle cacheHandle2 = 0;
RsslDataDictionary dataDictionary;

/* For simplicity in this code fragment, CHK is assumed to be a macro for error handling
   (performing cleanup and returning from function). */

/* Initialize cache component and create cache instances */
ret = rsslPayloadCacheInitialize(); CHK(ret)
cacheConfig.maxItems = 0; /* unlimited */
cacheHandle1 = rsslPayloadCacheCreate(&cacheConfig, &cacheError);
if (cacheHandle1 == 0)
{
    printf("rsslPayloadCacheCreate failure: %s\n", cacheError.text);
    CHK(cacheError.rsslErrorId)
}
cacheHandle2 = rsslPayloadCacheCreate(&cacheConfig, &cacheError);
if (cacheHandle2 == 0)
{
    printf("rsslPayloadCacheCreate failure: %s\n", cacheError.text);
    CHK(cacheError.rsslErrorId)
}

/* Load an RDM Field Dictionary structure from file: set to each cache. */
rsslClearDataDictionary(&dataDictionary);
ret = rsslLoadFieldDictionary("RDMFieldDictionary", &dataDictionary, &errorBuffer); CHK(ret)
/* Initial setting of the dictionary to the first cache */
ret = rsslPayloadCacheSetDictionary(cacheHandle1, &dataDictionary, dictionaryKey, &cacheError);
    CHK(ret)
/* Shared setting of the same dictionary to the second cache */
ret = rsslPayloadCacheSetSharedDictionaryKey(cacheHandle2, dictionaryKey, &cacheError); CHK(ret)
/* The dataDictionary can be destroyed after setting, but is typically retained by the application
   for encoding and decoding. */

/* Two cache instances are now ready for applying and retrieving data */
/* ... */

/* Cleanup */
rsslPayloadCacheDestroy(cacheHandle1); /* destroys all entries and the cache instance */
rsslPayloadCacheDestroy(cacheHandle2);
```

```

/* After all cache instances bound to a dictionary are destroyed, the cache API will clean up the
   internal field dictionary database used by the cache. */
rsslPayloadCacheUninitialize(); /* final call to cache interface */
rsslDeleteDataDictionary(&dataDictionary);

```

Code Example 34: Creating Cache and Setting to Dictionary

12.2.4.3 Extending the Cache Field Dictionary

While a cache can only be set to a single dictionary during its lifetime, the set of field definitions defined by the dictionary can be extended. This is accomplished by reloading the cache field definition database with another call to the **rsslPayloadCacheSetDictionary** function. When extending the field dictionary, the **RsslDataDictionary** must contain the original field definitions and any new definitions the application wishes to use. Changes or deletions to the original field definitions are not supported; only additions are allowed. Using the same **RsslPayloadCacheHandle** and dictionary key that were previously set, call the **rsslPayloadCacheSetDictionary** function again with extended dictionary structure.

NOTE: When extending a field dictionary that is shared, all caches sharing that same dictionary key will see the extension with only a single call to **rsslPayloadCacheSetDictionary**. There is no need to set the shared dictionary key again to each cache after a dictionary is extended.

12.2.5 Payload Cache Utilities

Use the following functions for managing cache instances. These utilities provide a count of the cache entries and a list of handles to each cache entry.

FUNCTION NAME	DESCRIPTION
rsslPayloadCacheGetEntryCount	Returns the number of item payload entries in this cache instance (RsslPayloadCacheHandle).
rsslPayloadCacheGetEntryList	Populates an array provided by the caller with entry handles (RsslPayloadEntryHandle) for this cache instance (RsslPayloadCacheHandle). Because each cache entry is likely associated with an entry in the application's item list, an application would typically manage the set of entry handles. This utility provides access to the entire entry handle list if needed.
rsslPayloadCacheClearAll	Destroys all entries in the cache instance (RsslPayloadCacheHandle). The empty cache can be reused and remains bound to its data dictionary.

Table 215: Payload Cache Utility Functions

12.3 Payload Cache Entries

A payload cache entry stores a single OMM container (whose data types are defined by **RsslContainerType**). While a cache entry can store any arbitrary OMM data, the primary use case is to maintain the last known value of an item data stream by applying the sequence of refresh and update messages in the stream to the cache entry. Initial data applied to a container must be a refresh message payload, which will define the container type to be stored (e.g. Map). As refresh and update messages from the item stream are applied to the cache entry, the cache decodes the OMM data and sets the current value by following the OMM rules for the container (e.g., adding, deleting, or updating map entries in a Map, or updating fields in a field list). The last value of the data stream can be retrieved from cache at any time as an encoded OMM container.

12.3.1 Managing Payload Cache Entries

Payload cache entries are created within a cache instance. A cache entry is defined by the **RsslPayloadEntryHandle** returned from the **rsslPayloadEntryCreate** function. You cannot move entries between different cache instances, due to their dependency on the field dictionary bound to the cache where they are created.

Cache entries only store the payload container of an item. Maintain other item data (e.g. message key attributes, domain, state) as needed in an item list managed by the application, which will identify the source or sink associated with the cache entry data. This item list will likely include the **RsslPayloadEntryHandle** if the payload of the item is cached.

For a list of basic utilities provided by the payload cache to manage the collection of entries in the cache, refer to Section 12.2.5.

Use the following functions to manage cache entries:

FUNCTION NAME	DESCRIPTION
rsslPayloadEntryCreate	This method returns a RsslPayloadEntryHandle to the newly created entry in the cache defined by the given RsslPayloadCacheHandle . The RsslPayloadEntryHandle is required for all operations on this entry. This function will return a null handle if it cannot create the entry (e.g., if the maximum number of entries as defined in RsslPayloadCacheConfigOptions would be exceeded).
rsslPayloadEntryDestroy	This method destroys the cache entry defined by RsslPayloadEntryHandle and removes it from its cache.
rsslPayloadEntryClear	This method deletes any data in the cache entry RsslPayloadEntryHandle and returns the entry to its initial state. The entry itself remains in the cache and can be re-used.

Table 216: Payload Cache Entry Management Functions

12.3.2 Applying Data

Data is applied to a cache entry from the payload of an OMM message by using the **rsslPayloadEntryApply** function. The decoded **RsslMsg** and an **RsslDecodeIterator** are passed to the apply function. The iterator (positioned at the start of the encoded payload data **RsslMsgBase.encDataBody**) will be used to decode the OMM data so that the cache entry data can be set or updated.

Some caching behaviors are controlled by flags in the **RsslMsg**. When an **RsslRefreshMsg** is applied to the cache entry, the following **RsslRefreshFlags** take effect:

- **RSSL_RFMF_CLEAR_CACHE**: Cache entry data will be cleared prior to applying this message.
- **RSSL_RFMF_DO_NOT_CACHE**: The payload will not be applied to the cache entry.

When an **RsslUpdateMsg** is applied to cache, the following **RsslUpdateFlags** take effect:

- **RSSL_UPMF_DO_NOT_CACHE**: The payload data will not be applied to the cache entry.
- **RSSL_UPMF_DO_NOT_RIPPLE**: When applying the data, entry rippling is not performed.

The following example demonstrates how to create a payload entry in a cache instance and apply the payload of an **RsslMsg** to the cache entry.

```
/* Apply buffer containing an encoded RsslMsg to cache entry */
RsslRet applyBufferToCache(RsslChannel *pChannel, RsslBuffer *pBuffer, RsslPayloadCacheHandle
    cacheHandle, RsslPayloadEntryHandle *pEntryHandle)
{
    RsslDecodeIterator dIter;
    RsslMsg msg;
    RsslRet ret;
    RsslCacheError cacheError;

    /* If the caller did not provide a cache entry handle, create a new entry */
    if (*pEntryHandle == 0)
    {
        rsslCacheErrorClear(&cacheError);
        *pEntryHandle = rsslPayloadEntryCreate(cacheHandle, &cacheError);
        if (*pEntryHandle == 0)
        {
            printf("Error (%d) creating cache entry: %s\n", cacheError.rsslErrorId, cacheError.text);
            return cacheError.rsslErrorId;
        }
    }

    /* Perform message decoding. */
    rsslClearDecodeIterator(&dIter);
    rsslSetDecodeIteratorRwfVersion(&dIter, pChannel->majorVersion, pChannel->minorVersion);
    rsslSetDecodeIteratorBuffer(&dIter, pBuffer);
    rsslClearMsg(&msg);
    ret = rsslDecodeMsg(&dIter, &msg);
    if (ret < RSSL_RET_SUCCESS)
    {
        printf("Failure (%d) decoding message from buffer\n");
        return ret;
    }
}
```

```

/* Apply the decoded RsslMsg to cache, with iterator positioned at the start of the payload */
rsslCacheErrorClear(&cacheError);
ret = rsslPayloadEntryApply(*pEntryHandle, &dIter, &msg, &cacheError);
if (ret < RSSL_RET_SUCCESS)
{
    printf("Error (%d) applying data to cache entry: %s\n", cacheError.rsslErrorId,
        cacheError.text);
    return ret;
}

return ret;
}

```

Code Example 35: Applying Data to a Payload Cache Entry

12.3.3 Retrieving Data

Data is retrieved from a cache entry as an encoded OMM container by using the **rsslPayloadEntryRetrieve** function. The application provides the data buffer (via an **RsslEncodeIterator**) where the container will be encoded. The retrieve function supports both encoding scenarios. When using **rsslEncodeMsg**, the encoded content retrieved from the cache entry can be set on the **RsslMsgBase.encDataBody**. If using **rsslEncodeMsgInit** and **rsslEncodeMsgComplete** encoding, the cache retrieve function can encode the message payload prior to **rsslEncodeMsgComplete**.

There are two options for using the **rsslPayloadEntryRetrieve** function. For single-part retrieval, the buffer provided by the application must be large enough to hold the entire encoded container. For multi-part retrieval, the application makes a series of calls to **rsslPayloadEntryRetrieve** to get the OMM container in fragments (e.g., a sequence of maps are retrieved which together contain the entire set of map entries for the container). In this usage, the optional **RsslPayloadCursorHandle** is required to maintain the state of the multi-part retrieval. Container types **FieldList** and **ElementList** cannot be fragmented, so the buffer size must be large enough to retrieve the entire container.

The following functions describe data-related operations on a cache entry.

FUNCTION NAME	DESCRIPTION
rsslPayloadEntryGetDataType	Returns the RsslContainerType stored in the cache entry (RsslPayloadEntryHandle). When initially created (or after the entry is cleared), the data type will be RSSL_DT_UNKNOWN . The data type is defined by the container type of the first refresh message applied to the entry.
rsslPayloadEntryApply	Applies the OMM data in the payload of the RsslMsg to the cache entry (RsslPayloadEntryHandle). The first message applied must be a refresh message (class RSSL_MC_REFRESH).
rsslPayloadEntryRetrieve	Retrieves data from the cache entry by encoding the OMM container into the buffer provided with the RsslEncodeIterator given by the application. For single-part retrieval, the RsslPayloadCursorHandle parameter is optional. For details on multi-part retrieval, refer to Section 12.3.3.1.

Table 217: Functions for Applying and Retrieving Cache Entry Data

12.3.3.1 Multi-Part Retrieval

For data types that support fragmentation, the container can be retrieved in multiple parts by calling **rsslPayloadEntryRetrieve** until the complete container is returned. To support multi-part retrieval, the optional **RsslPayloadCursorHandle** parameter is required when calling **rsslPayloadEntryRetrieve**. The cursor is used to maintain the position where the next retrieval will resume. The application must check the state of the cursor after each call to **rsslPayloadEntryRetrieve** to determine when the retrieval is complete. The following functions are needed when using the payload cursor.

FUNCTION NAME	DESCRIPTION
rsslPayloadCursorCreate	Creates a cursor for optional use in the rsslPayloadEntryRetrieve function (required for multi-part retrieval). Returns the RsslPayloadCursorHandle .
rsslPayloadCursorDestroy	Destroys the cursor referenced by the RsslPayloadCursorHandle .
rsslPayloadCursorClear	Clears the state of the cursor for the given RsslPayloadCursorHandle . Whenever retrieving data from a cache entry, the cursor must be cleared prior to the first call to rsslPayloadEntryRetrieve . Clearing the cursor also allows it to be reused with a retrieval on a different container.
rsslPayloadCursorsIsComplete	Returns the completion state of a retrieval where the RsslPayloadCursorHandle was used. The state must be checked after each call to rsslPayloadEntryRetrieve to determine whether additional data needs to be encoded for the cache entry container. When the cursor state is complete, the entire container of the cache entry has been retrieved.

Table 218: Functions for Using the Payload Cursor

12.3.3.2 Buffer Management

In multi-part usage, the size of the buffer used in the calls to **rsslPayloadEntryRetrieve** will affect how many fragments are required to retrieve the entire image of the cache entry. The retrieve function will continue to encode OMM entries from the cache container until it runs out of room in the buffer to encode the next entry. To progress during a multi-part retrieval, the buffer size must be at least large enough to encode a single OMM entry from the payload container. For example, if retrieving a map in multiple parts, the buffer must be large enough to encode at least one **MapEntry** on each retrieval.

There are three general outcomes when using the **rsslPayloadEntryRetrieve** function:

- Full cache container is encoded into the buffer. This can occur with or without the use of the optional **RsslPayloadCursorHandle**. If used in this scenario, the cursor state would indicate the retrieval is complete.
- Partial container encoded into the buffer. This is only possible when using the **RsslPayloadCursorHandle** for container types that support fragmentation. The application must check the cursor to test whether this is the final part.
- No data encoded into container due to insufficient buffer size. This can occur with or without the use of the optional **RsslPayloadCursorHandle**. The application may retrieve again with a larger buffer.

12.3.3.3 Example: Cache Retrieval with Multi-Part Support

The following example illustrates data retrieval from a cache entry, which supports multi-part encoding of a container.

```
/* Code fragment showing use of rsslPayloadEntryRetrieve for multi-part retrieval. */

RsslRet ret;
RsslCacheError cacheError;
RsslBuffer buffer;
RsslEncodeIterator eIter;
int arraySize = DEFAULT_BUFFER_SIZE;
unsigned char* bufferArray = (char*) malloc(arraySize);
buffer.data = bufferArray;
buffer.length = arraySize;
RsslPayloadCursorHandle cursorHandle = rsslPayloadCursorCreate();
rsslPayloadCursorClear(cursorHandle);
while (!rsslPayloadCursorIsComplete(cursorHandle))
{
    buffer.length = arraySize;
    rsslClearEncodeIterator(&eIter);
    rsslSetEncodeIteratorBuffer(&eIter, &buffer);
    rsslCacheErrorClear(&cacheError);
    /* _entryHandle created outside the scope of this code fragment */
    ret = rsslPayloadEntryRetrieve(_entryHandle, &eIter, cursorHandle, &cacheError);
    if (ret == RSSL_RET_SUCCESS)
        /* Number of bytes encoded is buffer.length. Application can use encoded data, e.g. set the
           payload on RsslMsgBase.encDataBody and encode a message to be transmitted. */
    else if (ret == RSSL_RET_BUFFER_TOO_SMALL)
        /* Increase arraySize and reallocate bufferArray. */
    else
        /* Handle terminal error condition. See cacheError.text[] for additional information. */
}
rsslPayloadCursorDestroy(cursorHandle);
free(bufferArray);
```

Code Example 36: Cache Retrieval with Multi-Part Support

© LSEG 2015 - 2024. All rights reserved.

Republication or redistribution of LSEG Data & Analytics content, including by framing or similar means, is prohibited without the prior written consent of LSEG Data & Analytics. 'LSEG Data & Analytics' and the LSEG Data & Analytics logo are registered trademarks and trademarks of LSEG Data & Analytics.

Any third party names or marks are the trademarks or registered trademarks of the relevant third party.

Document ID: ETAC382UMVAC.240

Date of issue: September 2024



LSEG DATA &
ANALYTICS