

Enterprise Transport API

C Edition

3.7.1.L1

VALUE ADDED COMPONENTS

Document Version: 3.7.1
Date of issue: June 2023
Document ID: ETAC371UMVAC.230



© Refinitiv 2015 - 2023. All rights reserved.

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

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.

Refinitiv, 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. Refinitiv, 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	2
1.6	Documentation Feedback	4
1.7	Document Conventions	4
1.7.1	<i>Optional, Conditional, and Required</i>	4
1.7.2	<i>Typographic</i>	4
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	9
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>	27
6.3	Reactor Channel Use	28
6.3.1	<i>Reactor Channel Roles</i>	29
6.3.2	<i>Reactor Channel Role: OMM Consumer</i>	30
6.3.3	<i>Reactor Channel Role: OMM Provider</i>	32
6.3.4	<i>Reactor Channel Role: OMM Non-Interactive Provider</i>	33
6.3.5	<i>Reactor Channel: Role Union</i>	35
6.4	Managing Reactor Channels.....	36
6.4.1	<i>Adding Reactor Channels</i>	36
6.4.2	<i>Removing Reactor Channels</i>	41
6.5	OMM Consumer Multi-Credential Configuration	42
6.5.1	<i>Login Message Configuration and Callback</i>	42
6.6	Reporting on Channel Statistics	42
6.7	Dispatching Data	43
6.7.1	<i>rssl/ReactorDispatch Function</i>	43
6.7.2	<i>Reactor Callback Functions</i>	44
6.7.3	<i>Reactor Callback: Channel Event</i>	45
6.7.4	<i>Reactor Callback: Default Message</i>	48
6.7.5	<i>Reactor Callback: RDM Login Message</i>	49
6.7.6	<i>Reactor Callback: RDM Directory Message</i>	51
6.7.7	<i>Reactor Callback: RDM Dictionary Message</i>	52
6.8	Writing Data	54
6.8.1	<i>Writing Data using rssl/ReactorSubmitMsg()</i>	54
6.8.2	<i>Writing data using rssl/ReactorSubmit()</i>	57
6.9	Creating and Using Tunnel Streams	64
6.9.1	<i>Authenticating a Tunnel Stream</i>	64
6.9.2	<i>Opening a Tunnel Stream</i>	65
6.9.3	<i>Negotiating Stream Behaviors: Class of Service</i>	67
6.9.4	<i>Tunnel Stream Callback Functions and Event Types</i>	70
6.9.5	<i>Opening a Tunnel Stream Code Sample</i>	73
6.9.6	<i>Accepting Tunnel Streams</i>	74
6.9.7	<i>Receiving Content on a TunnelStream</i>	79
6.9.8	<i>Sending Content on a TunnelStream</i>	79
6.10	Cloud Connectivity	83
6.10.1	<i>rssl/ReactorQueryServiceDiscovery</i>	83
6.10.2	<i>OAuth Credential Management</i>	86
6.11	JSON to RWF Protocol Conversion for WebSocket Support	90
6.11.1	<i>Consumer and WebSocket Support</i>	90
6.11.2	<i>Interactive Provider and WebSocket Support</i>	92

6.11.3	<i>RsslReactorJsonConverterOptions Structure</i>	94
6.11.4	<i>RsslReactorJsonConversionEventCallback Function</i>	95
6.11.5	<i>RsslReactorServiceNameToldCallback Function</i>	95
6.11.6	<i>RsslReactorServiceNameToldEvent Structure</i>	96
6.11.7	<i>RsslReactorJsonConversionEvent Structure</i>	96
6.12	Reactor Utility Functions	97
6.12.1	<i>General Reactor Utility Functions</i>	97
6.12.2	<i>RsslReactorChannelInfo Structure Members</i>	97
6.12.3	<i>rsslReactorloctl Option Values</i>	97
6.12.4	<i>Reactor Debug Functions</i>	97
7	Consuming Data from the Cloud	99
7.1	Overview	99
7.2	Encrypted Connections	99
7.3	Credential Management.....	99
7.4	Version 1 Authentication Using oAuth Password and Refresh_Token	100
7.4.1	<i>Client_ID (AppKey)</i>	100
7.4.2	<i>Obtaining Initial Access and Refresh Tokens</i>	100
7.4.3	<i>Refreshing the Access Token and Sending a Login Reissue</i>	101
7.4.4	<i>Managing the Password and Client Secret</i>	102
7.4.5	<i>Session Management per User Credential</i>	102
7.5	Version 2 Authentication Using oAuth Client Credentials	102
7.5.1	<i>Configuring and Managing Version 2 Credentials</i>	102
7.5.2	<i>Version 2 oAuth Client Credentials Token Lifespan</i>	103
7.6	Service Discovery	104
7.7	Consuming Market Data	105
7.8	HTTP Error Handling for Reactor Token Reissues	105
7.9	Cloud Connection Use Cases	106
7.9.1	<i>Session Management Use Case</i>	106
7.9.2	<i>Disabling the Watchlist</i>	106
7.9.3	<i>Query Service Discovery</i>	106
7.10	Logging of Authentication and Service Discovery Interaction	107
7.10.1	<i>Logged Request Information</i>	107
7.10.2	<i>Logged Response Information</i>	107
8	Administration Domain Models Detailed View	108
8.1	Concepts	108
8.2	RDM Message Base	109
8.2.1	<i>RSSL RDM Message Base Structure Members</i>	109
8.2.2	<i>RSSL RDM Message Types</i>	109
8.2.3	<i>RSSL RDM Encoding and Decoding Functions</i>	110
8.3	RDM Login Domain	110
8.3.1	<i>RSSL RDM Login Request</i>	110
8.3.2	<i>RSSL RDM Login Refresh</i>	115
8.3.3	<i>RSSL RDM Login Status</i>	121
8.3.4	<i>RSSL RDM Login Close</i>	123
8.3.5	<i>RSSL RDM Consumer Connection Status</i>	123
8.3.6	<i>Login Round Trip Time Message Use</i>	126
8.3.7	<i>Login Post Message Use</i>	127
8.3.8	<i>Login Ack Message Use</i>	127
8.3.9	<i>RSSL RDM Login Message Union</i>	127
8.3.10	<i>RSSL RDM Login Encoding and Decoding</i>	128
8.4	RSSL RDM Source Directory Domain	133
8.4.1	<i>RSSL RDM Directory Request</i>	133
8.4.2	<i>RSSL RDM Directory Refresh</i>	134

8.4.3	<i>RSSL RDM Directory Update</i>	136
8.4.4	<i>RSSL RDM Directory Status</i>	137
8.4.5	<i>RSSL RDM Directory Close</i>	138
8.4.6	<i>RSSL RDM Consumer Status</i>	139
8.4.7	<i>Source Directory RDM Service</i>	140
8.4.8	<i>Source Directory RDM Service Info</i>	141
8.4.9	<i>Source Directory RDM Service State</i>	143
8.4.10	<i>Source Directory RDM Service Group State</i>	144
8.4.11	<i>Source Directory RDM Service Load</i>	145
8.4.12	<i>Source Directory RDM Service Data</i>	146
8.4.13	<i>Source Directory RDM Service Link Information</i>	147
8.4.14	<i>Source Directory RDM Service Link</i>	148
8.4.15	<i>Source Directory RDM Sequenced Multicast Information</i>	149
8.4.16	<i>RSSL RDM Directory Message Union</i>	150
8.4.17	<i>Source Directory Encoding and Decoding</i>	151
8.5	Dictionary Domain	158
8.5.1	<i>RSSL RDM Dictionary Request</i>	158
8.5.2	<i>RSSL RDM Dictionary Refresh</i>	159
8.5.3	<i>RSSL RDM Dictionary Status</i>	161
8.5.4	<i>RSSL RDM Dictionary Close</i>	163
8.5.5	<i>RSSL RDM Dictionary Message Union</i>	163
8.5.6	<i>Dictionary Encoding and Decoding</i>	164
8.6	RDM Queue Messages	169
8.6.1	<i>Queue Data Message Persistence</i>	169
8.6.2	<i>Queue Request</i>	169
8.6.3	<i>Queue Refresh</i>	170
8.6.4	<i>Queue Status</i>	170
8.6.5	<i>Queue Close</i>	170
8.6.6	<i>Queue Data</i>	171
8.6.7	<i>QueueDataExpired</i>	174
8.6.8	<i>Queue Ack</i>	175
9	Warm Standby Feature	176
9.1	Overview	176
9.2	Warm Standby Modes	176
9.3	Warm Standby Configuration and Feature Details	178
9.3.1	<i>Configuration Example for a Starting Server and a Standby Server</i>	180
9.4		180
10	Value Added Utilities	181
10.1	Utility Overview	181
10.2	Memory Buffer	181
10.3	Using the Memory Buffer	182
11	Payload Cache Detailed View	183
11.1	Concepts	183
11.2	Payload Cache	184
11.2.1	<i>Managing the Payload Cache</i>	184
11.2.2	<i>Cache Error Handling</i>	184
11.2.3	<i>Payload Cache Instances</i>	185
11.2.4	<i>Managing RDM Field Dictionaries for Payload Cache</i>	186
11.2.5	<i>Payload Cache Utilities</i>	188
11.3	Payload Cache Entries	189
11.3.1	<i>Managing Payload Cache Entries</i>	189

11.3.2	<i>Applying Data</i>	190
11.3.3	<i>Retrieving Data</i>	191

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 via rsslReactorSubmit	57
Figure 8.	Tunnel Stream Illustration	64
Figure 9.	Obtaining an Authentication Token	100
Figure 10.	Login Reissue	101
Figure 11.	Service Discovery	104
Figure 12.	Login Based Warm Standby Order of Events in a Cutover from Active to Standby.....	177
Figure 13.	Service Based Warm Standby Order of Events in a Cutover from Active to Standby	178
Figure 14.	Consumer Application using Cache to Store Payload Data for Item Streams	183

List of Tables

Table 1:	Acronyms and Abbreviations	1
Table 2:	ETA Functionality and ETA Reactor Comparison	22
Table 3:	Reactor Error Info Codes	23
Table 4:	25
Table 5:	Reactor Creation Function.....	25
Table 6:	ReactorOptions Structure Members	26
Table 7:	ReactorOptions Utility Function	27
Table 8:	Reactor Debugging Levels	27
Table 9:	Reactor Destruction Function	28
Table 10:	ReactorChannel Class Members.....	29
Table 11:	ReactorRole class Members	30
Table 12:	ReactorRoleType Enumerated Values.....	30
Table 13:	ConsumerRole Class Properties	31
Table 14:	ConsumerRole.DictionaryDownloadMode Enumerated Values.....	32
Table 15:	OMM Consumer Role Watchlist Options	32
Table 16:	ConsumerRole Utility Function.....	32
Table 17:	ProviderRole Structure Members.....	33
Table 18:	ProviderRole Utility Function.....	33
Table 19:	NIProviderRole Structure Members	34
Table 20:	NIProviderRole Utility Function	34
Table 21:	RsslReactorChannelRole Union Members	35
Table 22:	RsslReactorChannelRole Utility Function	35
Table 23:	ReactorConnect Function	36
Table 24:	ReactorConnectOption CLASS Members.....	37
Table 25:	ReactorConnectInfo Class Members	37
Table 26:	new ReactorConnectOptions Utility Function	38
Table 27:	Reactor.Accept Function	40
Table 28:	ReactorAcceptOptions Class Members	40
Table 29:	ReactorAcceptOptions Utility Function	40
Table 30:	Function.....	41
Table 31:	RsslReactorLoginRequestMsgCredential Structure Members.....	42
Table 32:	rsslReactorDispatch	43
Table 33:	RsslReactorDispatchOptions Class Properties	43
Table 34:	RsslReactorDispatchOptions Utility Function	43
Table 35:	ReactorCallbackReturnCodeRsslReactorCallbackReturnCodes Callback Return Codes.....	44
Table 36:	RsslReactorChannelEvent Members	45
Table 37:	RsslReactorChannelEventType Enumeration Values	45
Table 38:	RsslReactorChannelEvent Utility Functions	46
Table 39:	RsslMsgEvent Structure Members	48
Table 40:	RsslMsgEvent Utility Function.....	48
Table 41:	RsslRDMLoginMsgEvent Structure Members	49
Table 42:	RsslRDMLoginMsgEvent Utility Function	49
Table 43:	RsslRDMDirectoryMsgEvent Structure Members	51
Table 44:	RsslRDMDirectoryMsgEvent Utility Function	51
Table 45:	RsslRDMDictionaryMsgEvent Structure Members	52
Table 46:	RsslRDMDictionaryMsgEvent Utility Function	53
Table 47:	rsslReactorSubmitMsg Function	54
Table 48:	RsslReactorSubmitMsgOptions Structure Members	54
Table 49:	RsslReactorRequestMsgOptions Structure Members	55
Table 50:	RsslReactorSubmitMsgOptions Utility Function	55
Table 51:	rsslReactorSubmitMsg Return Codes.....	55

Table 52:	Reactor Buffer Management Functions	58
Table 53:	rss1ReactorGetBuffer Return Values	59
Table 54:	rss1ReactorSubmit Function	59
Table 55:	Rss1ReactorSubmitOptions Structure Members	60
Table 56:	rss1ReactorSubmit Return Codes	60
Table 57:	Rss1ReactorsSubmitOptions Utility Function	61
Table 58:	rss1ReactorPackBuffer Function.....	62
Table 59:	rss1ReactorPackBuffer Return Values	62
Table 60:	Rss1TunnelStreamAuthInfo Structure Members	65
Table 61:	rss1ReactorOpenTunnelStream Method.....	65
Table 62:	Rss1TunnelStreamOpenOptions	66
Table 63:	Rss1ClassOfService.common Structure Members	67
Table 64:	Rss1ClassOfService.authentication Structure Members	68
Table 65:	Rss1ClassOfService.flowControl Structure Members	68
Table 66:	Rss1ClassOfService.dataIntegrity Structure Members	69
Table 67:	Rss1ClassOfService.guarantee Structure Members	69
Table 68:	Tunnel Stream Callback Event Types.....	70
Table 69:	Tunnel Stream Callback Functions	71
Table 70:	Tunnel Stream Callback Event Types.....	72
Table 71:	Rss1TunnelStreamRequestEvent Structure Members.....	75
Table 72:	rss1ReactorAcceptTunnelStream Function	75
Table 73:	Rss1ReactorAcceptTunnelStreamOptions Options.....	76
Table 74:	rss1ReactorRejectTunnelStream Function	76
Table 75:	Rss1ReactorRejectTunnelStreamOptions Options.....	76
Table 76:	Tunnel Stream Buffer Methods	79
Table 77:	Tunnel Stream Submit Method	79
Table 78:	Rss1TunnelStreamSubmitOptions Structure Members.....	80
Table 79:	Rss1TunnelStreamSubmitMsgOptions Structure Members	80
Table 80:	Rss1TunnelStreamInfo Structure Members	80
Table 81:	rss1ReactorCloseTunnelStream Method	81
Table 82:	Rss1TunnelStreamCloseOptions Structure Members	82
Table 83:	rss1ReactorQueryServiceDiscovery Method.....	83
Table 84:	Rss1ReactorServiceDiscoveryOptions Structure Members	83
Table 85:	Rss1ReactorDiscoveryTransportProtocol Enumerations.....	84
Table 86:	Rss1ReactorDiscoveryDataFormatProtocol Enumerations	84
Table 87:	Rss1ReactorServiceEndpointEvent Structure Members	85
Table 88:	Rss1ReactorServiceEndpointEvent Structure Members	86
Table 89:	Rss1ReactorOAuthCredential Structure Members	86
Table 90:	Rss1ReactorOAuthCredentialEvent Structure Members	87
Table 91:	Rss1ReactorOAuthCredentialRenewal Members.....	87
Table 92:	rss1ReactorSubmitOAuthCredentialRenewal	88
Table 93:	rss1ReactorSubmitOAuthCredentialRenewal Options	88
Table 94:	Rss1ReactorOAuthCredentialRenewalMode Enums	89
Table 95:	Rss1ReactorJsonConverterOptions Structure Members	94
Table 96:	Rss1ReactorServiceNameToIdCallback Parameters	95
Table 97:	Rss1ReactorServiceNameToIdEvent Structure Members	96
Table 98:	Rss1ReactorJsonConversionEvent Structure Members	96
Table 99:	Reactor Utility Functions	97
Table 100:	Rss1ReactorChannelInfo Structure Members	97
Table 101:	Domains Representations in the Administration Domain Model Value Added Component.....	108
Table 102:	Rss1RDMMMsgBase Structure Members	109
Table 103:	Rss1RDMMMsg	109
Table 104:	RDM Encoding and Decoding Functions	110
Table 105:	Rss1RDMLLoginRequest Structure Members	111
Table 106:	Rss1RDMLLoginRequest Flags	113

Table 107: Rss1RDMLoginRequest Utility Functions.....	114
Table 108: Rss1RDMLoginRefresh Structure Members	115
Table 109: Rss1RDMLoginRefresh Flags	118
Table 110: Rss1RDMLoginRefresh Utility Functions.....	120
Table 111: Rss1RDMServerInfo Structure Members.....	120
Table 112: Rss1RDMServerInfo Flags	120
Table 113: Rss1RDMServerInfo Utility Functions.....	120
Table 114: Rss1RDMLoginStatus Structure Members.....	121
Table 115: Rss1RDMLoginStatus Flags	122
Table 116: Rss1RDMLoginStatus Utility Functions.....	123
Table 117: Rss1RDMLoginClose Structure Member.....	123
Table 118: Rss1RDMLoginClose Utility Functions.....	123
Table 119: Rss1RDMLoginConsumerConnectionStatus Structure Members	124
Table 120: Rss1RDMLoginConsumerConnectionStatus Flags	124
Table 121: Rss1RDMLoginWarmStandbyInfo Structure Members	124
Table 122: RDMLoginServerTypes Enumeration Values.....	124
Table 123: Rss1RDMLoginConsumerConnectionStatus Utility Functions.....	125
Table 124: Login Round Trip Time Members.....	126
Table 125: Login Round Trip Time Flag Enumeration Values	126
Table 126: Login Round Trip Time Utility Functions	126
Table 127: Rss1RDMLoginMsg Union Members	127
Table 128: Rss1RDMLoginMsg Utility Functions	127
Table 129: RDM Login Encoding and Decoding Functions	128
Table 130: Rss1RDMDirectoryRequest Structure Members.....	133
Table 131: Rss1RDMDirectoryRequest Flags	134
Table 132: Rss1RDMDirectoryRequest Utility Functions.....	134
Table 133: Rss1RDMDirectoryRefresh Structure Members.....	134
Table 134: Rss1RDMDirectoryRefresh Flags	135
Table 135: Rss1RDMDirectoryRefresh Utility Functions	135
Table 136: Rss1RDMDirectoryUpdate Structure Members.....	136
Table 137: Rss1RDMDirectoryUpdate Flags	136
Table 138: Rss1RDMDirectoryUpdate Utility Functions	137
Table 139: Rss1RDMDirectoryStatus Structure Members	137
Table 140: Rss1RDMDirectoryStatus Flags	138
Table 141: Rss1RDMDirectoryStatus Utility Functions	138
Table 142: Rss1RDMDirectoryClose Structure Member.....	138
Table 143: Rss1RDMDirectoryClose Utility Functions	138
Table 144: Rss1RDMDirectoryConsumerStatus Structure Members	139
Table 145: Rss1RDMConsumerStatusService Structure Members.....	139
Table 146: Rss1RDMDirectoryConsumerStatus Utility Functions	139
Table 147: Rss1RDMService Structure Members	140
Table 148: Rss1RDMService Flags	140
Table 149: Rss1RDMService Utility Function	141
Table 150: Rss1RDMServiceInfo Structure Members	141
Table 151: Rss1RDMServiceInfo Flags	142
Table 152: Rss1RDMServiceInfo Utility Functions	143
Table 153: Rss1RDMServiceState Structure Members	143
Table 154: Rss1RDMServiceState Flags	144
Table 155: Rss1RDMServiceState Utility Functions	144
Table 156: Rss1RDMServiceGroupState Structure Members	144
Table 157: Rss1RDMServiceGroupState Flags	145
Table 158: Rss1RDMServiceGroupState Utility Functions	145
Table 159: Rss1RDMServiceLoad Structure Members.....	145
Table 160: Rss1RDMServiceLoad Flags	146
Table 161: Rss1RDMServiceLoad Utility Functions	146

Table 162: Rss1RDMServiceData Structure Members.....	146
Table 163: Rss1RDMServiceData Flags	147
Table 164: Rss1RDMServiceData Utility Functions.....	147
Table 165: Rss1RDMServiceLinkInfo Structure Members.....	147
Table 166: Rss1RDMServiceLinkInfo Utility Functions	147
Table 167: Rss1RDMServiceLink Structure Members.....	148
Table 168: Rss1RDMServiceLink Flags	148
Table 169: Rss1RDMServiceLink Utility Functions.....	149
Table 170: Rss1RDMServiceSeqMcastInfo Structure Members	149
Table 171: RSSL RDM Service Sequenced Multicast Info Enumeration Values	149
Table 172: RSSL RDM Address/Port Information Structure Members	150
Table 173: Rss1RDMServiceSeqMcastInfo Utility Functions	150
Table 174: Rss1RDMDirectoryMsg Union Members.....	150
Table 175: Rss1RDMDirectoryMsg Utility Functions.....	150
Table 176: RDM Directory Encoding and Decoding Functions	151
Table 177: Rss1RDMDictionaryRequest Structure Members	158
Table 178: Rss1RDMDictionaryRequest Flag	158
Table 179: Rss1RDMDictionaryRequest Utility Functions.....	159
Table 180: Rss1RDMDictionaryRefresh Structure Members	159
Table 181: Rss1RDMDictionaryRefreshFlags	160
Table 182: Rss1RDMDictionaryRefresh Utility Functions.....	161
Table 183: Rss1RDMDictionaryStatus Structure Members.....	161
Table 184: Rss1RDMDictionaryStatus Flags	161
Table 185: Rss1RDMDictionaryStatus Utility Functions	162
Table 186: Rss1RDMDictionaryClose Structure Members.....	163
Table 187: Rss1RDMDictionaryClose Utility Functions	163
Table 188: Rss1RDMDictionaryMsg Union Members	163
Table 189: Rss1RDMDictionaryMsg Utility Functions	163
Table 190: RDM Dictionary Encoding and Decoding Functions	164
Table 191: Rss1RDMQueueRequest Members	169
Table 192: Rss1RDMQueueRefresh Members	170
Table 193: Rss1RDMQueueStatus Members	170
Table 194: Rss1RDMQueueClose Members	170
Table 195: Rss1RDMQueueData Members	171
Table 196: Queue Data Flag	171
Table 197: Rss1RDMQueueTimeoutCodes	172
Table 198: Queue Data Message Encoding Methods	172
Table 199: Rss1RDMQueueDataExpired Structure Members	174
Table 200: Rss1RDMQueueDataUndeliverableCodes	174
Table 201: Rss1RDMQueueAck	175
Table 202: Memory Buffer Functions	181
Table 203: Payload Cache Management Functions	184
Table 204: Rss1CacheError Structure Members	184
Table 205: Function for Cache Error Handling	185
Table 206: Functions for Managing Cache Instances	185
Table 207: Rss1PayloadCacheConfigOptions Structure Members	185
Table 208: Functions for Setting Dictionary to Cache	186
Table 209: Payload Cache Utility Functions	188
Table 210: Payload Cache Entry Management Functions	189
Table 211: Functions for Applying and Retrieving Cache Entry Data	191
Table 212: Functions for Using the Payload Cursor	192

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 Refinitiv 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 Refinitiv 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	Refinitiv Real-Time Advanced Data Hub is the horizontally scalable service component within the Refinitiv Real-Time Distribution System providing high availability for publication and contribution messaging, subscription management with optional persistence, conflation and delay capabilities.
ADS	Refinitiv Real-Time Advanced Distribution Server is the horizontally scalable distribution component within the Refinitiv Real-Time Distribution System providing highly available services for tailored streaming and snapshot data, publication and contribution messaging with optional persistence, conflation and delay capabilities.
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
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 Refinitiv Real-Time SDK. It consists of transport, buffer management, compression, fragmentation and packing over each transport and encoders and decoders that implement the Open Message Model. Applications written to this layer achieve the highest throughput, lowest latency, low memory utilization, and low CPU utilization using a binary Refinitiv Wire Format when publishing or consuming content to/from Refinitiv Real-Time Distribution Systems.
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol (Secure)
JWK	JSON Web Key. Defined by RFC 7517, a JWK is a JSON formatted public or private key.
JWKS	JSON Web Key Set, This is a set of JWK, placed in a JSON array.
JWT	JSON Web Token. Defined by RFC 7519, JWT allows users to create a signed claim token that can be used to validate a user.
OMM	Open Message Model
QoS	Quality of Service
RDM	Refinitiv Domain Model
Refinitiv Real-Time Distribution System	Refinitiv Real-Time Distribution System is Refinitiv's financial market data distribution platform. It consists of the Refinitiv Real-Time Advanced Distribution Server and Refinitiv Real-Time Advanced Data Hub. Applications written to the Refinitiv Real-Time SDK can connect to this distribution system.
Reactor	The Reactor is a low-level, open-source, easy-to-use layer above the Enterprise Transport API. It offers heartbeat management, connection and item recovery, and many other features to help simplify application code for users.
RFA	Robust Foundation API
RMTES	A multi-lingual text encoding standard
RSSL	Refinitiv Source Sink Library
RTT	Round Trip Time, this definition is used for round trip latency monitoring feature.
RWF	Refinitiv Wire Format, a Refinitiv proprietary binary format for data representation.
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

1. Enterprise Transport API C Edition *Refinitiv Domain Model Usage Guide*
2. *API Concepts Guide*

3. *RMTES Specification*
4. Enterprise Transport API C Edition *Developers Guide*
5. The [Refinitiv Developer Community](#)
- 6.

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

1.7 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:

- Structuresmethods, Methods, in-line code snippets, and types are shown in **Courier New** font.
- Parameters, filenames, tools, utilities, and directories are shown in **Bold** font.
- Document titles and variable values are shown in *italics*.
- When initially introduced, concepts are shown in ***Bold, Italic***.
- 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, Refinitiv Real-Time server, or other application		Network of multiple servers
	Enterprise Transport API application		Point-to-point connection showing direction of primary data flow
			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 Refinitiv 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.

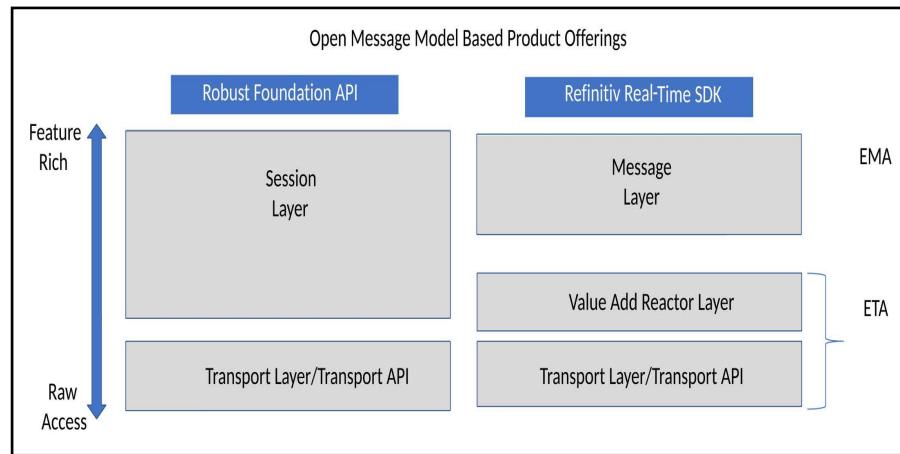


Figure 3. OMM APIs with Value Added Components

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.

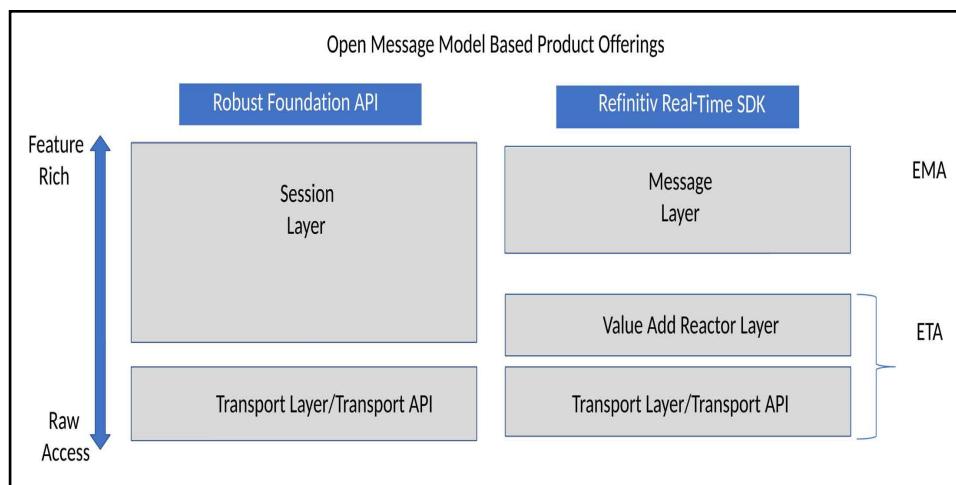


Figure 4. Enterprise Transport API Value Added Components

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.

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

1. Refinitiv 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 Refinitiv, 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_SOCKET**) and multicast networks (**RSSL_CONN_TYPE_RELIABLE_MULTICAST**). The reactor uses the watchlist to provide the same interaction model for both TCP and Multicast communications, so that application developers need not write code specific to either system.

For details on configuring the **RsslReactor** to enable the consumer watchlist, refer to Section 6.3.2 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 *Refinitiv 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 *Refinitiv 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., Refinitiv 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 `rssIRDMMMsg.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 Refinitiv Real-Time Distribution System, Refinitiv Data Feed Direct, and Refinitiv 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 **Reactor** can manage one or multiple **RsslReactorChannel** structures. This functionality allows the application to associate OMM consumer connections with an existing **Reactor**, having it manage more than one connection, or to create a new **Reactor** to use with the connection.

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

For more information about the **Reactor** 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 **ReactorChannel** basis so each channel can have its own unique callback functions or existing callback functions can be specified and shared across multiple **ReactorChannels**.

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

- **RsslReactorChannelEventCallback**, which returns information about the **ReactorChannel** 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:

- **RsslRDMLoginMsgCallback**, which processes all data for the RDM Login domain.
- **IDirectoryMsgCallback**, which processes all data for the RDM Source Directory domain.

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

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

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 **Reactor** will use this information when starting up the **ReactorChannel**.

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 **ConsumerRole**, 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 **ConsumerRole** 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 **ReactorChannel** 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 Refinitiv are defined in the Enterprise Transport API C Edition *Refinitiv Domain Model Usage Guide*. This content will be returned to the application via the **IDefaultMsgCallback**.

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 **ReactorChannel** by calling **ReactorChannel.Close**. This will close all item streams and log out the user. Prior to closing the **ReactorChannel**, the application should release any unwritten pool buffers to ensure proper memory cleanup.

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

- Closing a **ReactorChannel** is described in Section 6.4.2.
- Shutting down a **Reactor** 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 **Consumer** 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 *Refinitiv 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 **Reactor**, or create a new one
- Create a **Server**
- Implement callbacks and populate role
- Associate incoming connections using **Reactor.Accept**
- 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 **Provider** 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 RssIReactor

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

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

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

4.3 Create an RssIServer

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 **Transport.Bind** 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 **Server** and associate the incoming connections with an **Reactor**, 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 **ReactorChannel** basis so each channel can have its own unique callback functions or existing callback functions can be specified and shared across multiple **ReactorChannels**.

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

- **IReactorChannelEventCallback**, which returns information about the **ReactorChannel** and its state (e.g., connection up)
- **IDefaultMsgCallback**, 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:

- **IRDMLLoginMsgCallback**, which processes all data for the RDM Login domain.
- **IDirectoryMsgCallback**, which processes all data for the RDM Source Directory domain.
- **IDictionaryMsgCallback**, which processes all data for the RDM Dictionary domain.

The **ProviderRole** structure should be populated with all callback information for the **ReactorChannel**.

Detailed information about the **ProviderRole** is in Section 6.3.1. Information about the various callback functions and their specifications are available in Section 6.6.2.

4.5 Associate Incoming Connections Using **rsslReactorAccept**

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

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

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

- For more details on the **Reactor.Accept** function, refer to.
- For more details on dispatching, refer to.

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 **IRDMLLoginMsgCallback**, when specified on the **ProviderRole**.

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 **LoginRefresh** 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 **LoginStatus**, closing the connection and informing the user of the reason for denial.

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

All content requested, received, or posted is encoded and decoded using the ETA C Message 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 *Refinitiv 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. Refinitiv 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 **serviceId** 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 C Message 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 *Refinitiv 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 **FieldList** type requires the use of a field dictionary (usually the Refinitiv **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 a Refinitiv Real-Time Advanced Data 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. A Refinitiv Real-Time Advanced Data 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 *Refinitiv Domain Model Usage Guide*.

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

Whether loading a dictionary from file or requesting it from a Refinitiv Real-Time Advanced Data 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 C Message Package and the ETA C 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 *Refinitiv 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 **msgKey** 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 **StatusMsg** 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 *Refinitiv Domain Model Usage Guide* defines all domains provided by Refinitiv. This content will be returned to the application via the **IDefaultMsgCallback**.

The provider can specify that it supports post messages via the **LoginRefresh**. 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 **IDefaultMsgCallback**.

All content requested, received, or posted is encoded and decoded using the ETA CMessage Package and the ETA C 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 **Reactor** 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 **StatusMsg** 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 **ReactorChannels** by calling **ReactorChannel.Close**. Prior to closing the **ReactorChannel**, the application should release any unwritten pool buffers to ensure proper memory cleanup.

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

- Closing an **ReactorChannel** is described in Section 6.4.2.
- Shutting down an **Reactor** 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 **rssIVAPeriodic** 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 *Refinitiv 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 a Refinitiv Real-Time Advanced Data Hub. After the reactor indicates that the connection is ready, an OMM non-interactive provider can publish information into the Refinitiv Real-Time Advanced Data Hub cache without needing to handle requests for the information. The Refinitiv Real-Time Advanced Data Hub and other Refinitiv 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 **Reactor**
- Implement callbacks and populate role
- Establish connection using **ReactorConnect**
- Perform dictionary download
- Provide content
- Log out and shut down

The **NIProvider** 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 RssIRreactor

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

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

Detailed information about the **Reactor** 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 **ReactorChannel** basis so each channel can have its own unique callback functions or existing callback functions can be specified and shared across multiple **ReactorChannels**.

A **Reactor** requires the use of the following callback functions:

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

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

The **NIProviderRole** structure should be populated with all callback information for the **ReactorChannel**. **NIProviderRole** 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 **NIPrviderRole** structure or with specific information according to the application and user. The **Reactor** will use this information when starting up the **ReactorChannel**.

- For detailed information on the **NIPrviderRole**, 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 **rssIReactorConnect**

After populating the **NIPrviderRole**, the application can use **Reactor.Connect** to create a new outbound connection.

Reactor.Connect 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 **IReactorChannelEventCallback**, which provides the **ReactorChannel** and indicates the current connection state. At this point, the application can begin using the **Reactor.Dispatch** function to dispatch directly on this **ReactorChannel**, or continue using to dispatch across all channels associated with the **Reactor**.

The **Reactor** will use the login and directory information specified on the **NIPrviderRole** 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 **Reactor.Connect** 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 Refinitiv Real-Time Advanced Data 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 Refinitiv Real-Time Advanced Data 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 *Refinitiv Domain Model Usage Guide*.

5.6 Provide Content

After the **ReactorChannel** is established, it can begin pushing content to the Refinitiv Real-Time Advanced Data Hub. Each unique information stream should begin with an **RefreshMsg**, 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 CSharp 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 `ReactorChannel` by calling `ReactorChannel.Close`. This will close all item streams and log out the user. Prior to closing the `ReactorChannel`, the application should release any unwritten pool buffers to ensure proper memory cleanup.

If the application is done with the `Reactor`, the `Reactor.Shutdown` function can be used to shutdown and cleanup any `Reactor` resources.

- For details on closing an `ReactorChannel`, refer to Section 6.4.2.
- Shutting down an `Reactor` 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 `rssIVANIProvider` 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 *Refinitiv 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. 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.

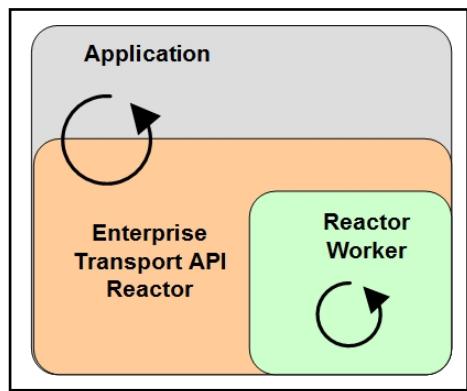


Figure 5. ETA Reactor Thread Model

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: • 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

6.1.3 Reactor Error Info Codes

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

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

Table 3: 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 **Reactor**. The **Reactor** manages **ReactorChannels** (described in Section 6.3). An understanding of both constructs is necessary for application writers.

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

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

PROPERTY	DESCRIPTION
EventSocket	Socket object used for Reactor-specific events to communicate with the worker thread. This socket can be polled for incoming data.
userSpecPtr	A pointer that can be set by the user of the Reactor . This value can be set directly or via the creation options. This information can be useful for identifying a specific instance of an Reactor or coupling this Reactor with other user-defined information.

Table 4:

6.2.1 Creating a Reactor

The lifecycle of an **Reactor** 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 **Reactor** 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
Reactor.CreateReactor	Creates an Reactor instance, including all necessary internal memory and threads. After creating the Reactor , ReactorChannels can be associated, as described in Section 6.3. Options are passed in via the ReactorOptions , as defined in Section 6.2.1.2.

Table 5: Reactor Creation Function

6.2.1.2 RssICreateReactorOptions Structure Members

CLASS MEMBER/PROPERTY	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 rss1BindThread method to bind threads. Reactor creation will fail when rss1BindThread 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 Reactor 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. Reactor now chooses an ephemeral port upon creation. Any values specified in this parameter are ignored.
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.
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 Reactor.Connect() method, if the request times out, the Reactor does not retry.</p> <p>If set to 0, there is no timeout. Be default, the Enterprise Transport API behaves as if set to 90 seconds.</p>
serviceDiscoveryURL	Specifies the URL of Refinitiv Data 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 Refinitiv 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 Reactor waits before retrieving a new access token and refreshing its connection to Refinitiv 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 .

Table 6: ReactorOptions Structure Members

CLASS MEMBER/PROPERTY	DESCRIPTION
tokenServiceURL	Specifies the URL of Refinitiv Data Platform that the RTSDK API uses to obtain an authentication token.
UserSpecObj	An object that can be set by the application. This value is preserved and stored in the UserSpecObj of the Reactor returned from Reactor.CreateReactor . This information can be useful for coupling this RsslChannel with other user-created information, such as a watch list associated with this connection.

Table 6: ReactorOptions Structure Members(Continued)**6.2.1.3 RsslCreateReactorOptions Utility Function**

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

FUNCTION NAME	DESCRIPTION
	Clears the ReactorOptions structure. Useful for structure reuse.

Table 7: ReactorOptions 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 8: 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 **Reactor** instance is controlled by the application, which controls creation and destruction of each reactor instance. The following sections describe destruction functionality in more detail.

6.2.2.1 Reactor Destruction

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

FUNCTION NAME	DESCRIPTION
rsslDestroyReactor	Destroys an Reactor instance, including all internal memory and threads. This also sends ReactorChannelEvents , indicating channel down, to all ReactorChannels associated with this Reactor .

Table 9: Reactor 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 **ReactorChannel** class is used to represent a connection that can send or receive information across a network. This class is used to represent a connection, regardless of whether it is an outbound connection or a connection accepted by a listening socket via a **Server**. The **ReactorChannel** 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 **ReactorChannel** and how to associate with a **Reactor**.

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

The following table describes the properties of the **ReactorChannel** class.

STRUCTURE MEMBER	DESCRIPTION
hostname	Provides the name of the host to which a consumer or NIP application connects.
majorVersion	When an ReactorChannel is up (ReactorChannelEventType.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 ReactorChannel is up (ReactorChannelEventType.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> .
	It is possible for a socket 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 ReactorChannelEventType.FD_CHANGE . The previous socket is stored in so the application can properly handle the change.
port	Provides the server port number to which the consumer or NIP application connects.
protocolType	When an ReactorChannel is up (ReactorChannelEventType.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. 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> .
	The A pointer to the underlying IChannel 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 Channel instance directly with any RSSL Transport functionality.
	The A pointer to the underlying Server instance, 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 ReactorAccept function, as described in Section 6.4.1.7.
state	The state of the .
UserSpecObj	An object pointer that can be set by the user of the Channel . This value can be set directly or via the ReactorConnectOptions and ReactorAcceptOptions . This information can be useful for coupling this with other user-created information, such as a watch list associated with this connection.

Table 10: ReactorChannel Class Members

6.3.1 Reactor Channel Roles

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

- **ConsumerRole** for OMM consumer applications

- **ProviderRole** for OMM interactive provider applications
- **NIProviderRole** for OMM non-interactive provider applications

All roles have the same common element, the **ReactorRole** class.

6.3.1.1 RsslReactorChannelRoleBase Structure

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

CLASS PROPERTY	DESCRIPTION
channelEventCallback	This ReactorChannel 's user-defined callback function to handle all ReactorChannel specific events, like ReactorChannelEventType.CHANNEL_UP or ReactorChannelEventType.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 ReactorChannel 's user-defined callback function to handle Msg 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 11: **ReactorRole** class Members

6.3.1.2 roleType Enumerations

ENUMERATED NAME	DESCRIPTION
RC_RT_INIT	Role is not specified. This is intended for structure initialization only.
RC_RT_OMM_CONSUMER	Indicates that the ReactorChannel should act as a consumer.
RC_RT_OMM_NI_PROVIDER	Indicates that the ReactorChannel should act as a non-interactive provider.
RC_RT_OMM_PROVIDER	Indicates that the ReactorChannel should act as a interactive provider.

Table 12: **ReactorRoleType** Enumerated Values

6.3.2 Reactor Channel Role: OMM Consumer

When an **ReactorChannel** 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 *Refinitiv Domain Model Usage Guide*.

An **ReactorChannel** 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 **ReactorChannel**, this information can be specified with the **ConsumerRole** structure as follows:

CLASS PROPERTY	DESCRIPTION
base	The role base structure, as defined in Section 6.3.1.1.
dictionaryDownloadMode	Informs the ReactorChannel of the method to use when requesting dictionaries. Allowable modes are defined in Section 6.3.2.2.
dictionaryMsgCallback	This ReactorChannel '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 ReactorChannel '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 ReactorChannel '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 DirectoryRequest (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 InitDefDirectoryRequest function to populate with default information. <ul style="list-style-type: none"> If this parameter is specified, a RdmLoginRequest 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 Refinitiv Data Platform token service request. This is required when connecting to a Refinitiv Real-Time Advanced Distribution Server in the cloud. See Section 6.9.2.1.
pOAuthCredentialList	This is an array of ReactorOAuthCredential 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 ReactorConnectInfo.oAuthCredentialIndex .
pLoginRequestList	This is an array of ReactorLoginRequestMsgCredential 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 ReactorConnectOptions .

Table 13: ConsumerRole Class Properties

6.3.2.2 OMM Consumer Role Dictionary Download Modes

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

ENUMERATED NAME	DESCRIPTION
RSSL_RC_DICTIONARY_DOWNLOAD_FIRST_AVAILABLE	The Reactor will search received directory messages for the RDMFieldDictionary (RWFFld) and the enumtype.def (RWFEnum) dictionaries. Once found, the Reactor 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 Reactor will not request dictionaries for this ReactorChannel . This is typically used when the application has loaded a file-based dictionary or has acquired the dictionary elsewhere.

Table 14: ConsumerRole.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 15: 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 **ConsumerRole**.

FUNCTION NAME	DESCRIPTION
	Clears the ConsumerRole structure. Useful for structure reuse.

Table 16: ConsumerRole Utility Function

6.3.3 Reactor Channel Role: OMM Provider

When **ReactorChannel** 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 *Refinitiv Domain Model Usage Guide*

A **ReactorChannel** 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 **ReactorChannel**, this information can be specified with the **ProviderRole** structure, as follows:

STRUCTURE MEMBER	DESCRIPTION
base	The role base structure, as defined in Section 6.3.1.1.
DictionaryMsgCallback	This ReactorChannel '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.
DirectoryMsgCallback	This ReactorChannel '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.
LoginMsgCallback	This ReactorChannel '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.

Table 17: ProviderRole Structure Members

6.3.3.2 OMM Provider Role Utility Function

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

FUNCTION NAME	DESCRIPTION
	Clears the ProviderRole structure. Useful for structure reuse.

Table 18: ProviderRole Utility Function

6.3.4 Reactor Channel Role: OMM Non-Interactive Provider

When a **ReactorChannel** acts as an OMM non-interactive provider application, it connects to a Refinitiv Real-Time Advanced Data 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 *Refinitiv Domain Model Usage Guide*

A **ReactorChannel** 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 **ReactorChannel**, this information can be specified with the **NIProviderRole** structure, as follows:

PROPERTY	DESCRIPTION
base	The role base structure, as defined in Section 6.3.1.1.
RdmLoginRequest	The LoginRequest , defined in Section 8.3.1, sent when establishing a connection. You can populate this with a user's specific information or invoke the initDefaultRDMLoginRequest 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.
RdmDirectoryRefresh	The DirectoryRefresh , 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 pLoginRequest is required. • If this parameter is left empty, a refresh is not automatically sent to the system.
LoginMsgCallback	The ReactorChannel '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 19: NIProviderRole 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 **NIProviderRole**.

FUNCTION NAME	DESCRIPTION
	Clears the NIProviderRole structure. Useful for structure reuse.

Table 20: NIProviderRole 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 11.
ommConsumerRole	The RsslReactorOMMConsumerRole , as defined in Section 6.3.2.
ommProviderRole	The RsslReactorOMMProviderRole , as defined in Section Section 6.3.3.
ommNIProviderRole	The RsslReactorOMMNIProviderRole , as defined in Section 6.3.4.

Table 21: **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 22: **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 **ReactorConnect** function will create a new **ReactorChannel** and associate it with an **Reactor**. This function creates a new outbound connection. The **ReactorChannel** 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 **Reactor** automatically reconnect an **ReactorChannel** whenever a connection fails. To enable this, the application sets the appropriate members of the **ReactorConnectOption** structure. The application can specify that **Reactor** reconnect the **ReactorChannel** to the same host, or to one from among multiple hosts.

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

Table 23: **ReactorConnect** Function

6.4.1.2 RsslReactorConnectOptions Structure Members

CLASS MEMBER/PROPERTY	DESCRIPTION
connectionCount	Specifies the number of connections listed in <code>reactorConnectionList</code> . If set to 0 , <code>rsslConnectOptions</code> is used.
connectionDebugFlags	Specifies a set of <code>RsslDebugFlags</code> for use when calling user-set debug callbacks as set by <code>rsslSetDebugFunctions</code> . If set to 0 , debug callbacks are not used
initializationTimeout	Specifies the amount of time (in seconds) to wait to successfully establish an <code>RsslReactorChannel</code> . If a <code>RsslReactorChannel</code> is not established in this timeframe, an event is dispatched to the application to indicate that the <code>RsslReactorChannel</code> is down.
reactorConnectionList	Specifies an array of connection information. When used with <code>reconnectAttemptLimit</code> , the <code>Reactor</code> attempts to connect to each host in the list with each reconnection attempt.
reconnectAttemptLimit	The maximum number of times the <code>Reactor</code> attempts to reconnect a channel when it fails. If set to -1 , there is no limit.
reconnectMinDelay	Specifies the minimum length of time the <code>Reactor</code> waits (in milliseconds) before attempting to reconnect a failed channel. The time increases with each reconnection attempt, from <code>reconnectMinDelay</code> to <code>reconnectMaxDelay</code> .
reconnectMaxDelay	Specifies the maximum length of time the <code>Reactor</code> waits (in milliseconds) before attempting to reconnect a failed channel. The time increases with each reconnection attempt, from <code>reconnectMinDelay</code> to <code>reconnectMaxDelay</code> .
rsslConnectOptions	Specifies information (<code>rsslConnectOptions</code>) 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 <code>rsslConnect</code> function. This is described in more detail in the Enterprise Transport API C Edition <i>Developers Guide</i> .

Table 24: ReactorConnectOption CLASS Members

6.4.1.3 RsslReactorConnectInfo Structure Members

CLASS MEMBER	DESCRIPTION
rsslConnectOptions	Specifies information 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 <code>Connect</code> function. This is described in more detail in the Enterprise Transport API C Edition <i>Developers Guide</i> .
enableSessionManagement	Specifies whether the channel manages the authentication token on behalf of the user used to keep the session alive. Boolean. If set to <code>true</code> , the channel obtains the authentication token and refreshes it on behalf of user to keep session active. The default setting is <code>false</code> .
initializationTimeoutinitTimeout	Specifies the amount of time (in seconds) to wait to successfully establish an <code>ReactorChannel</code> . If a <code>ReactorChannel</code> is not established in this timeframe, an event is dispatched to the application to indicate that the <code>ReactorChannel</code> is down.

Table 25: ReactorConnectInfo Class Members

CLASS MEMBER	DESCRIPTION
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]).
pAuthTokenEventCallback	A callback function that receives ReactorAuthTokenEvents . The Reactor requests a token for the Consumer and NiProvider applications to send login requests and reissues with the token.

Table 25: ReactorConnectInfo Class Members (Continued)

6.4.1.4 RsslReactorConnectOptions Utility Function

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

FUNCTION NAME	DESCRIPTION
	Clears the new ReactorConnectOptions structure. Useful for structure reuse.

Table 26: new ReactorConnectOptions Utility Function

6.4.1.5 rsslReactorConnect Example

```
RsslReactorConnectOptions reactorConnectOpts;
RsslReactorOMMConsumerRole consumerRole;

RsslRDMLoginRequest 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. */
rsslInitDefaultRDMLoginRequest(&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

6.4.1.6 rsslReactorConnect Segmented Multicast Consumer Example

```

RsslReactorConnectOptions reactorConnectOpts;
RsslReactorOMMConsumerRole consumerRole;

RsslRDMLoginRequest loginRequest;
RsslRDMDirectoryRequest directoryRequest;

/* Configure connection options.*/
rsslClearReactorConnectOptions(&reactorConnectOpts);
reactorConnectInfo.rsslConnectOptions.connectionType = RSSL_CONN_TYPE_RELIABLE_MCAST;

/* Configure outgoing network */
reactorConnectOpts.rsslConnectOptions.connectionInfo.segmented.sendAddress = "232.6.6.1";
reactorConnectOpts.rsslConnectOptions.connectionInfo.segmented.sendServiceName = "30010";

/* Configure incoming network. This example listens to two multicast networks. */
reactorConnectOpts.rsslConnectOptions.connectionInfo.segmented.recvAddress = "232.6.6.2,232.6.6.4";
reactorConnectOpts.rsslConnectOptions.connectionInfo.segmented.recvServiceName = "30011";
reactorConnectOpts.rsslConnectOptions.connectionInfo.unicastServiceName = "55555";

/* Enable filtering of incoming multicast traffic. */
reactorConnectOpts.rsslConnectOptions.multicastOpts.flags = RSSL_MCAST_FILTERING_ON;

/* 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. */
rsslInitDefaultRDMLoginRequest(&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 3: `rsslReactorConnect` Segmented Multicast Consumer Example

6.4.1.7 Reactor Accept

The `Reactor.Accept()` function creates a new `ReactorChannel` and associates it with an `Reactor`. This function accepts the connection from an already running `IServer`. The `ReactorChannel` 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
ReactorAccept	<p>Creates an <code>ReactorChannel</code> by accepting it from an <code>Server</code>. This establishes a connection in a manner similar to the <code>Reactor.Accept()</code> 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 <code>ReactorAcceptOptions</code>, as defined in Section 6.4.1.8. <code>ReactorChannel</code>-specific information (such as the per-channel callback functions, the type of behavior, default RDM messages, and etc.) are passed in via the <code>ReactorRole</code>, as defined in Section 6.3.1.

Table 27: `Reactor.Accept` Function

6.4.1.8 RsslReactorAcceptOptions Structure Members

CLASS MEMBER / PROPERTY	DESCRIPTION
	The <code>AcceptOptions</code> associated with the underlying <code>Accept</code> function. This includes an option to reject the connection as well as a <code>UserSpecObject</code> . This is described in more detail in the Enterprise Transport API C Edition <i>Developers Guide</i> .
ialization	The amount of time (in seconds) to wait for the successful connection establishment of an <code>ReactorChannel</code> . If a timeout occurs, an event is dispatched to the application to indicate that the <code>ReactorChannel</code> is down.

Table 28: `ReactorAcceptOptions` Class Members

6.4.1.9 RsslReactorAcceptOptions Utility Function

The Enterprise Transport API provides the following utility function for use with the `ReactorAcceptOptions`.

FUNCTION NAME	DESCRIPTION
ClearReactorAcceptOptions	Clears the <code>ReactorAcceptOptions</code> instance. Useful for object reuse.

Table 29: `ReactorAcceptOptions` 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 4: rsslReactorAccept Example

6.4.2 Removing Reactor Channels

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

Table 30: Function

6.4.2.2 **rsslReactorClose Example**

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

Code Example 5: rsslReactorClose 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 RsslRDMLoginRequest . All login messages configured in pLoginRequestList must have the same SingleOpen and allow Suspect Data Verification flags and values on the RsslRDMLoginRequest .
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 31: **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** via **Reactor.Dispatch** method by either not setting **ReactorChannel** property of the supplied **ReactorDispatchOptions** instance or setting it to a specific **ReactorChannel** instance. 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. **.eventFd** (e.g., select, poll) Section 6.7.1

6.7.1 rsslReactorDispatch Function

NOTE: Applications should not call `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**, a deep copy of the object should be made if the application wishes to preserve it. To copy an **RsslMsg**, refer to the 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.

6.7.1.1 Reactor Dispatch Options

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

ReactorDispatch	
	DESCRIPTION
pReactorChannel	The specific RsslReactorChannel to dispatch on in this call. If NULL , rsslReactorDispatch will process across all RsslReactorChannels associated with the passed in RsslReactor .

Table 33: **RsslReactorDispatchOptions** Class Properties

6.7.1.2 RsslReactorDispatchOptions Utility Function

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

FUNCTION NAME	DESCRIPTION
rsslClearReactorDispatchOptions	Clears the RsslReactorDispatchOptions . Useful for reuse.

Table 34: **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 6: 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
RSSL_RC_CRET_SUCCESS	Indicates that the callback function was successful and the message or event has been handled.
RSSL_RC_CRET_FAILURE	Indicates that the message or event has failed to be handled. Returning this code from any callback function will cause the RsslReactor to shutdown.
RSSL_RC_CRET_RAISE	Can be returned from any domain-specific callback (e.g., IRDMLoginMsgCallback). This will cause the RsslReactor to invoke the RsslDefaultMsgCallback for this message upon the domain-specific callbacks return.

Table 35: ReactorCallbackReturnCodeRsslReactorCallbackReturnCodes Callback Return Codes

6.7.3 Reactor Callback: Channel Event

callback function prototype:

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

6.7.3.1 Reactor Channel Event

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

	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 36: **RsslReactorChannelEvent** 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. will contain more detailed information about what occurred. To clean up the failed RsslReactorChannel the application should call rsslReactorCloseChannel .
RSSL_RC_CET_CHANNEL_DOWN_REC ONNECTING	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 was called. Before exiting the channelEventCallback , the application should release any resources associated with the channel. remove file-descriptor 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 the watchlist is enabled at that a channel has been created via . Though the channel is still not ready for dispatch, the application can begin submitting request messages, which are sent after the channel successfully initializes.
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 .

Table 37: **RsslReactorChannelEventType** Enumeration Values

FLAG ENUMERATION	MEANING
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, replace with , 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_WARNING	Indicates that the RsslReactorChannel has experienced an event that did not result in connection failure, but may require the attention of the application. ReactorErrorInfo contains more detailed information about what occurred.

Table 37: **RsslReactorChannelEventType** Enumeration Values (Continued)

6.7.3.3 Reactor Channel Event Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearReactorChannelEvent	Clears an RsslReactorChannelEvent .

Table 38: **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 7: 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 **ReactorCallbackReturnCode.RAISE** value. callback function 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 **VAR_IDefaultMsgCallback**.

	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 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 39: RsslMsgEvent Structure Members

6.7.4.2 Reactor Message Event Utility Functions

FUNCTION NAME	DESCRIPTION
rsslClearMsgEvent	Clears an RsslMsgEvent structure.

Table 40: 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 8: 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:

```
RsslRDMLoginMsgCallback(RsslReactor*, RsslReactorChannel*, RsslRDMLoginMsgEvent*)
```

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

6.7.5.1 Reactor RDM Login Message Event

The **RsslRDMLoginMsgEvent** is returned to the application via the **RsslRDMLoginMsgCallback**.

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.
pRDMLoginMsg	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 RsslRDMLoginMsg , described in Section 8.3.

Table 41: **RsslRDMLoginMsgEvent** Structure Members

6.7.5.2 Reactor RDM Login Message Event Utility Function

FUNCTION NAME	DESCRIPTION
rssIClearRDMLoginMsgEvent	Clears an RsslRDMLoginMsgEvent structure.

Table 42: **RsslRDMLoginMsgEvent** Utility Function

6.7.5.3 Reactor RDM Login Message Event Callback Example

```
RsslReactorCallbackRet loginMsgCallback(RsslReactor *pReactor, RsslReactorChannel *pReactorChannel,
                                         RsslRDMLoginMsgEvent *pLoginMsgEvent)
{
    RsslRDMLoginMsg *pLoginMsg = pLoginMsgEvent->pRDMLoginMsg;

    /* Received an RsslRDMLoginMsg --- 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:
                RsslRDMLoginRefresh *pRefresh = &pLoginMsg->refresh;
                break;
            case RDM_LG_MT_STATUS:
                RsslRDMLoginStatus *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 9: 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 43: **RsslRDMDirectoryMsgEvent** Structure Members

6.7.6.2 Reactor RDM Directory Message Event Utility Function

FUNCTION NAME	DESCRIPTION
rssIClearRDMDirectoryMsgEvent	Clears an RsslRDMDirectoryMsgEvent structure.

Table 44: **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", pDirectoryEvent->baseMsgEvent.pErrorInfo->rsslError.text,
pDirectoryEvent->baseMsgEvent.pErrorInfo->errorLocation);
}

```

Code Example 10: 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 45: **RsslRDMDictionaryMsgEvent** Structure Members

6.7.7.2 Reactor RDM Dictionary Message Event Utility Function

FUNCTION NAME	DESCRIPTION
rssIClearRDMDictionaryMsgEvent	Clears an RsslRDMDictionaryMsgEvent structure.

Table 46: **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 11: 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 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()

rsslReactorSubmitMsg 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
rsslReactorSubmitMsg	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 47: **rsslReactorSubmitMsg** Function

6.8.1.2 Reactor Submit Message Options

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

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 RsslRDMMMsg structure to submit. Use only one instance of either pRsslMsg or pRDMMsg.

Table 48: **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 RsslRequestMsgs to send requests.

Table 48: **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 49: **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
rssIClearReactorSubmitMsgOptions	Clears the RsslReactorSubmitMsgOptions structure. Useful for structure reuse.

Table 50: **RsslReactorSubmitMsgOptions** Utility Function

6.8.1.5 rssICreateReactorSubmitMsg Return Codes

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

RETURN CODE	DESCRIPTION
RSSL_RET_SUCCESS	Indicates that the rssICreateReactorSubmitMsg 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 rssICreateReactorChannelIoctl 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 51: **rssICreateReactorSubmitMsg** 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 = RSSI_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 12: 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 `rsslReactorSubmitMsg`, 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.

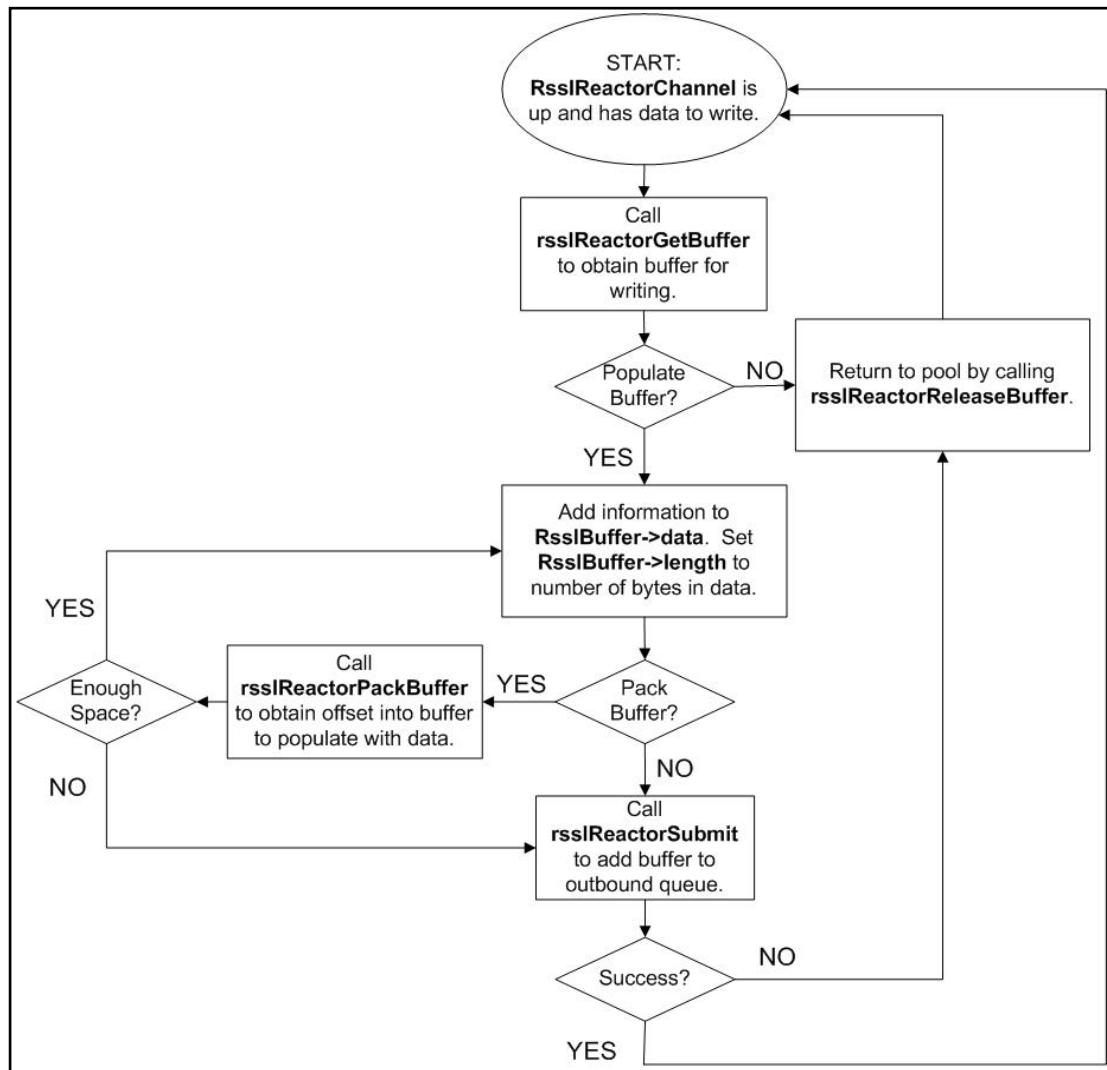


Figure 7. Flow Chart for writing data via `rsslReactorSubmit`

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 rsslReactorSubmit 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 rsslReactorSubmit call fails, the buffer must be returned to the pool using rsslReactorReleaseBuffer. If the rsslReactorSubmit 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 rsslReactorSubmit.</p> <p>This function calls the Enterprise Transport API rsslReleaseBuffer 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 52: 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 <code>RsslBuffer</code> is returned to the user. The <code>RsslBuffer.length</code> indicates the number of bytes available to populate and the <code>RsslBuffer.data</code> provides a starting location for population.
NULL buffer returned Error Code: RSSL_RET_BUFFER_NO_BUFFERS	NULL is returned to the user. This value indicates that there are no buffers available to the user. See <code>RsslErrorInfo</code> 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 <code>guaranteedOutputBuffers</code> (for details, refer to Section 6.12).
NULL buffer returned Error Code: RSSL_RET_FAILURE	NULL is returned to the user. This value indicates that some type of general failure has occurred. The <code>RsslReactorChannel</code> should be closed.
NULL buffer returned Error Code: RSSL_RET_INIT_NOT_INITIALIZED	Indicates that the underlying RSSL Transport has not been initialized. See the <code>RsslErrorInfo</code> content for more details.

Table 53: `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 `rsslReactorSubmit` 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 `rsslReactorSubmit` should be released to the pool using `rsslReactorReleaseBuffer`. Section 6.8.2.5 describes the `rsslReactorSubmit` 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 <code>length</code> reflects the actual number of bytes used. This function calls the Enterprise Transport API <code>rsslWrite</code> function and also triggers the <code>rsslFlush</code> 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 <code>RsslReactorSubmitOptions</code> structure, defined in Section 6.8.2.6. For a list of return codes, refer to Section 6.8.2.7.

Table 54: `rsslReactorSubmit` Function

6.8.2.6 Writing Data: Reactor Submit Options

The application uses **Rss1ReactorSubmitOptions** to control various aspects of the call to **rss1ReactorSubmit**.

STRUCTURE MEMBER	DESCRIPTION
Priority	<p>Controls the priority at which the data will be written. Valid priorities are</p> <ul style="list-style-type: none"> • RSSL_HIGH_PRIORITY • RSSL_MEDIUM_PRIORITY • RSSL_LOW_PRIORITY <p>More information about write priorities, including an example scenario, are available in the <i>Transport API C Edition Developers Guide</i>.</p>
writeFlags	<p>Flag values that allow the application to modify the behavior of this rss1ReactorSubmit call. This includes options to bypass queuing or compression.</p> <p>More information about the specific flag values are available in the <i>Transport API C Edition Developers Guide</i>.</p>
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 55: Rss1ReactorSubmitOptions Structure Members

6.8.2.7 Writing Data: rss1ReactorSubmit Return Codes

The following table defines the return codes that can occur when using **rss1ReactorSubmit**.

RETURN CODE	DESCRIPTION
RSSL_RET_SUCCESS	Indicates that the rss1ReactorSubmit function has succeeded. The RsslBuffer will be released by the Enterprise Transport API Reactor.
RSSL_RET_WRITE_CALL AGAIN	<p>Indicates that a large buffer could not be fully written with this rss1ReactorSubmit call. This is typically due to all pool buffers being unavailable. The RsslReactor will flush for the user to free up buffers. The application can optionally use rss1ReactorChannelIoctl to increase the number of available pool buffers. After pool buffers become available again, the same buffer should be used to call rss1ReactorSubmit 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> <p>If the application does not subsequently pass the buffer to rss1ReactorSubmit, the application should release it by calling rss1ReactorReleaseBuffer.</p>
RSSL_RET_FAILURE	<p>Indicates that a general write failure has occurred. The RsslReactorChannel should be closed.</p> <p>The application should release the RsslBuffer by calling rss1ReactorReleaseBuffer.</p>

Table 56: rss1ReactorSubmit 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
rsslClearReactorSubmitOptions	Clears the RsslReactorSubmitOptions structure. Useful for structure reuse.

Table 57: 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 13: 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
rsslReactorPackBuffer	<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 58: **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: RSSL_RET_FAILURE	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: RSSL_RET_INIT_NOT_INITIALIZED	Indicates that the underlying RSSL Transport has not been initialized. See the RsslErrorInfo content for more details.

Table 59: **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 14: 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 Refinitiv Real-Time Distribution System or a Refinitiv Real-Time Edge Device).

When creating a tunnel, the consumer indicates any additional behaviors to enforce, which is exchanged with the provider application endpoint. 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.

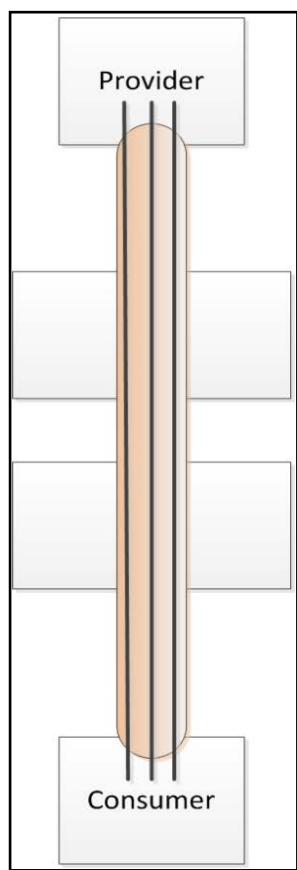


Figure 8. Tunnel Stream Illustration

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_OMP_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 60: 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
<code>rsslReactorOpenTunnelStream</code>	Begins the establishment of a tunnel stream. The <code>RsslTunnelStream</code> is returned via the <code>RsslTunnelStreamStatusEventCallback</code> as specified on the <code>RsslTunnelStreamOpenOptions</code> . For more details, refer to Section 6.9.2.2.

Table 61: 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 RsslRDMLoginRequest to send if RsslClassOfService.authentication.type is set to RDM_COS_AU_OMP_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 62: **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 **rss1ReactorOpenTunnelStream**, 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 **rss1TunnelStreamRequestGetCos** 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 **rssIRDM.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 63: 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 64: 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 65: 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	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. NOTE: At this time, RDM_COS_DI_RELIABLE is the only supported option.

Table 66: Rss1ClassofService.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	Indicates the level of guarantee that will be performed on this stream. RDM_COS_GU_PERSISTENT_QUEUE is not supported for provider applications. 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 .
persistLocally	RSSL_TRUE	RSSL_FALSE, RSSL_TRUE	Indicates whether messages are persisted locally on the tunnel stream. When type is RDM_COS_GU_NONE , this member has no effect.
persistenceFilePath	NULL	n/a	File path where files containing persistent messages may be stored. If set to NULL, the current working directory is used. When type is RDM_COS_GU_NONE , or when persistLocally is set to RSSL_FALSE , this member has no effect.

Table 67: Rss1ClassofService.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	<p>Indicates status information associated with the RsslTunnelStream. For example:</p> <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.
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.	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.
		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.
RsslTunnelStreamQueueMsgEvent	This event presents any queue message content received on the RsslTunnelStream .	pErrorInfo	Used to convey error information, when applicable.
		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 68: 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 69: 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.
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 70: 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 15: 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 71: **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 72: **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 73: **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 74: **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 75: **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 16: 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 17: 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
<code>rsslTunnelStreamGetBuffer</code>	Obtains a buffer from the <code>RsslTunnelStream</code> . 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.
<code>rsslTunnelStreamGetInfo</code>	Gets information about the Tunnel Stream by returning the <code>RsslTunnelStreamInfo</code> structure. For details on <code>RsslTunnelStreamInfo</code> methods, refer to Section 6.9.8.5.
<code>rsslTunnelStreamReleaseBuffer</code>	Releases a buffer back to the <code>RsslTunnelStream</code> 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 <code>RsslTunnelStream</code> .

Table 76: 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
<code>rsslTunnelStreamSubmitMsg</code>	Allows the user to pass in RDM message content, including Queue Messages, that will be processed and sent over the <code>RsslTunnelStream</code> . This method has additional options that can be specified via the <code>RsslTunnelStreamSubmitOptions</code> . Currently, the only available members of the option structure allow the user to pass in an RDM message or an <code>RsslMsg</code> structure containing their content.
<code>rsslTunnelStreamSubmit</code>	Allows the user to pass in a buffer populated with content that will be processed and sent over the <code>RsslTunnelStream</code> .

Table 77: Tunnel Stream Submit Method

6.9.8.3 RsslTunnelStreamSubmitOption

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

MEMBER	DESCRIPTION
containerType	<p>Specifies the type of data in the buffer being submitted. 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 78: `RsslTunnelStreamSubmitOptions` Structure Members

6.9.8.4 RsslTunnelStreamSubmitMsgOptions

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

MEMBER	DESCRIPTION
pRsslMsg	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> .
pRDMMsg	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 79: `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
buffersUsed	Returns the total number of buffers used by the Tunnel Stream for a user application.

Table 80: `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 18: 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 81: `rsslReactorCloseTunnelStream` Method

6.9.8.8 RsslTunnelStreamCloseOptions

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

MEMBER	DESCRIPTION
finalStatusEvent	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 82: `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 19: 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 Refinitiv Real-Time - Optimized service discovery.

6.10.1.1 rsslReactorQueryServiceDiscovery Method

METHOD	DESCRIPTION
rsslReactorQueryServiceDiscovery	<p>Uses the passed-in <code>RsslReactor</code> to query service endpoints from the Refinitiv Real-Time Optimized service according to the <code>rsslReactorQueryServiceDiscoveryOptions</code> that you specify (listed in Section 6.10.1.2).</p> <p>Error handling is managed by the <code>RsslErrorInfo</code> structure.</p> <p>NOTE: <code>rsslReactorQueryServiceDiscovery</code> 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, <code>rsslReactorQueryServiceDiscovery</code> retrieves a new access token by sending a token request using the username and password parameters.</p>

Table 83: `rsslReactorQueryServiceDiscovery` Method

6.10.1.2 RsslReactorServiceDiscoveryOptions

MEMBER	DESCRIPTION
audience	Optional and only used with Version 2 oAuth ClientCredentials with JWT logins. An <code>RsslBuffer</code> 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 <code>RsslBuffer</code> 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 <code>RsslBuffer</code> that contains the JWK-formatted private key associated with the Service Account. For further information, refer to clientSecret in this section.
clientSecret	An <code>RsslBuffer</code> 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 <code>RsslBuffer</code> that specifies a password for authorization with the token service.
proxyHostName	Optional. An <code>RsslBuffer</code> that specifies a proxy server hostname.
proxyPort	Optional. An <code>RsslBuffer</code> that specifies a proxy server port.
proxyUserName	Optional. An <code>RsslBuffer</code> that specifies a username to perform authorization with a proxy server.
proxyPasswd	Optional. An <code>RsslBuffer</code> that specifies a password to perform authorization with a proxy server.

Table 84: `RsslReactorServiceDiscoveryOptions` Structure Members

MEMBER	DESCRIPTION
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.
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 84: **RsslReactorServiceDiscoveryOptions** Structure Members (Continued)

6.10.1.3 RsslReactorDiscoveryTransportProtocol Enumerations

ENUMERATED NAME	DESCRIPTION
RSSL_RD_TP_INIT = 0	Specifies that the transport's protocol is unknown.
RSSL_RD_TP_TCP = 1	Specifies that the service discovery should use the TCP transport protocol.
RSSL_RD_TP_WEBSOCKET = 2	Specifies that the service discovery should use the Websocket transport protocol.

Table 85: **RsslReactorDiscoveryTransportProtocol** Enumerations

6.10.1.4 RsslReactorDiscoveryDataFormatProtocol Enumerations

ENUMERATED NAME	DESCRIPTION
RSSL_RD_DP_INIT = 0	Specifies that the transport's data format is unknown.
RSSL_RD_DP_RWF = 1	Specifies that the service discovery should use the RWF data format.
RSSL_RD_DP_JSON2 = 2	Specifies that the service discovery should use the tr_json2 data format

Table 86: **RsslReactorDiscoveryDataFormatProtocol** Enumerations

6.10.1.5 RsslReactorServiceEndpointEvent

MEMBER	DESCRIPTION
pErrorInfo	Returns any information about the error that occurred with the Refinitiv Data 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 87: **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 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 88: **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 2 authentication: a unique ID defined for the application that makes the request and is provisioned as part or a service account for the login. Version 1 authentication (Client ID usage with oAuth Password Credentials): refer to Section 7.4.1.
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.

Table 89: **RsslReactorOAuthCredential** Structure Members

MEMBER	DESCRIPTION
pOAuthCredentialEventCallback	A callback function that receives the RsslReactorOAuthCredentialEvent to specify the password and/or clientSecret . If pOAuthCredentialEventCallback 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 Refinitiv Data 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 Refinitiv Data Platform.

Table 89: RsslReactorOAuthCredential Structure Members (Continued)

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 Refinitiv Data Platform.

Table 90: 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 2 authentication: a unique ID defined for the application that makes the request and is provisioned as part or a service account for the login. Version 1 authentication (Client ID usage with oAuth Password Credentials): refer to Section 7.4.1.
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.

Table 91: RsslReactorOAuthCredentialRenewal Members

MEMBER	DESCRIPTION
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 Refinitiv Data 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 91: **RsslReactorOAuthCredentialRenewal** Members(Continued)

6.10.2.3 **rsslReactorSubmitOAuthCredentialRenewal** Method

MEMBERS	DESCRIPTION
rsslReactorSubmitOAuthCredentialRenewal	Uses the passed-in RsslReactor and RsslReactorOAuthCredentialRenewal to submit the application’s password (and client secret if available) to the Refinitiv Data Platform token service. An application can also use this method to change its password. For a list of options you can use with rsslReactorSubmitOAuthCredentialRenewal , refer to Section 6.10.2.4. If you call this method outside of the RsslReactorOAuthCredentialEventCallback , you should also include pAuthTokenEventCallback to receive a result response. Error handling is managed by the RsslErrorInfo structure.

Table 92: **rsslReactorSubmitOAuthCredentialRenewal**

6.10.2.4 **rsslReactorSubmitOAuthCredentialRenewal** Options

OPTION	DESCRIPTION
pAuthTokenEventCallback	A callback function (pAuthTokenEventCallback) 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. 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.
proxyDomain	An RsslBuffer that specifies the domain for authenticated proxies.

Table 93: **rsslReactorSubmitOAuthCredentialRenewal** Options

OPTION	DESCRIPTION
proxyHostName	An Rss1Buffer that specifies the proxy's host name.
proxyPasswd	An Rss1Buffer that specifies the password for authenticated proxies.
proxyPort	An Rss1Buffer that specifies the proxy's port.
proxyUserName	An Rss1Buffer that specifies the username for authenticated proxies.
renewalMode	A Rss1ReactorOAuthCredentialRenewalMode 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 Rss1Buffer that specifies a user name for authorization with the token service.

Table 93: rss1ReactorSubmitOAuthCredentialRenewal Options (Continued)**6.10.2.5 Rss1ReactorOAuthCredentialRenewalMode 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 only when changing the application's password.

Table 94: Rss1ReactorOAuthCredentialRenewalMode 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 (strcmp(&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 = (RsslUIInt16) 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 RsslUIInt16) 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 95: **RsslReactorJsonConverterOptions** Structure Members

MEMBER	DESCRIPTION
outputBufferSize	Sets the size (an RsslUInt32) that the converter allocates for its output buffer. Defaults to 65535 .  WARNING! If the output buffer is not large enough, JSON/RWF conversion will fail.
jsonTokenIncrementSize	Sets the number (an RsslUInt32) of JSON token increment size for parsing JSON messages.
pDictionary	Sets the data dictionary (RsslDataDictionary) that Enterprise Transport API uses to initialize the RWF/JSON converter. pDictionary defaults to 0 .
pJsonConversionEventCallback	Specifies the callback function (RsslReactorJsonConversionEventCallback) that receives the RsslReactorJsonConversionEvent if the JSON converter fails to convert a message. pJsonConversionEventCallback defaults to 0 . <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	Specifies the callback function (RsslReactorServiceNameToIdCallback) that handles conversion of the ServiceName to ServiceId . pServiceNameToIdCallback defaults to 0 . For further details on RsslReactorServiceNameToIdCallback , refer to Section 6.11.5.
userSpecPtr	Specifies a user-defined pointer which is retrieved in the callback function. pUserSpec is a void* type and defaults to 0 .

Table 95: **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 96: **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 97: **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 98: **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
rsslReactorGetChannelInfo	<p>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.</p> <p>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>.</p>
rsslReactorIoctl	<p>Allows the application to change various settings associated with the RsslReactorChannel. The available options are defined in Section 6.12.3.</p> <p>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>.</p>

Table 99: 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
rsslChannelInfo	<p>Returns the underlying RsslChannel information. This includes maxFragmentSize, number of output buffers, compression information, and more.</p> <p>The RsslChannelInfo function structure is fully described in the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>

Table 100: RsslReactorChannelInfo Structure Members

6.12.3 rsslReactorIoctl Option Values

There are currently no **Reactor** or **ReactorChannel** specific codes for use with the **IOCTL** method. Reactor-specific codes may be added in the future. The application can still use any of the codes allowed with **IOCTL**, 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
rsslReactorGetDebugLevel	Gets the level of the debug information.
rsslReactorSetDebugLevel	Sets the level of the debug information.

FUNCTION NAME	DESCRIPTION
rsslReactorGetDebugInfo	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. 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.

7 Consuming Data from the Cloud

7.1 Overview

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

- Credential Management (for details, refer to Section 7.4)
- 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 Refinitiv Data 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 Refinitiv **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 Refinitiv Real-Time as it functions in the cloud, refer to the *Refinitiv Real-Time - Optimized: Installation and Configuration for Client Use*.

7.2 Encrypted Connections

When connecting to a Refinitiv 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.

When configuring the **sslReactorChannel**, 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 Refinitiv 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 Refinitiv, 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 *Refinitiv 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 **Rss1ReactorOAuthCredential** 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 Refinitiv Data 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 Refinitiv Data 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:

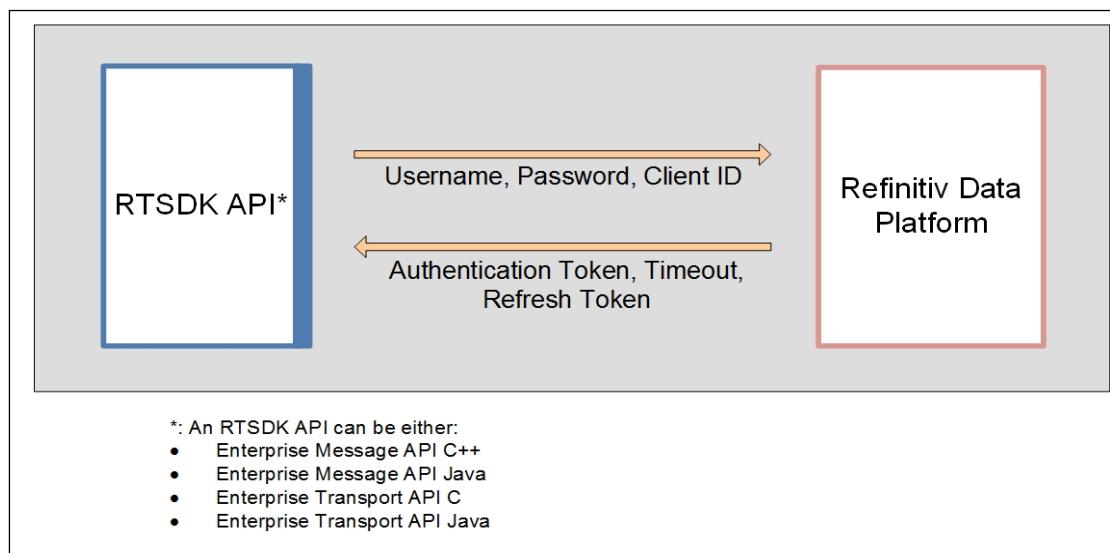


Figure 9. Obtaining an Authentication Token

7.4.3 Refreshing the Access Token and Sending a Login Reissue

In response to the API's token request, the Refinitiv Data 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 Refinitiv Real-Time Advanced Distribution Server does not receive a new access token before the end of the expiration timeout, the Refinitiv 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 Refinitiv Data Platform sends a new access token with an associated timeout to the API. After receiving the new access token from the Refinitiv Data Platform, the API renews its connection by sending a Login Reissue with the new access token to the Refinitiv Real-Time Advanced Distribution Server. The process of renewing the access token and refreshing the Refinitiv 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:

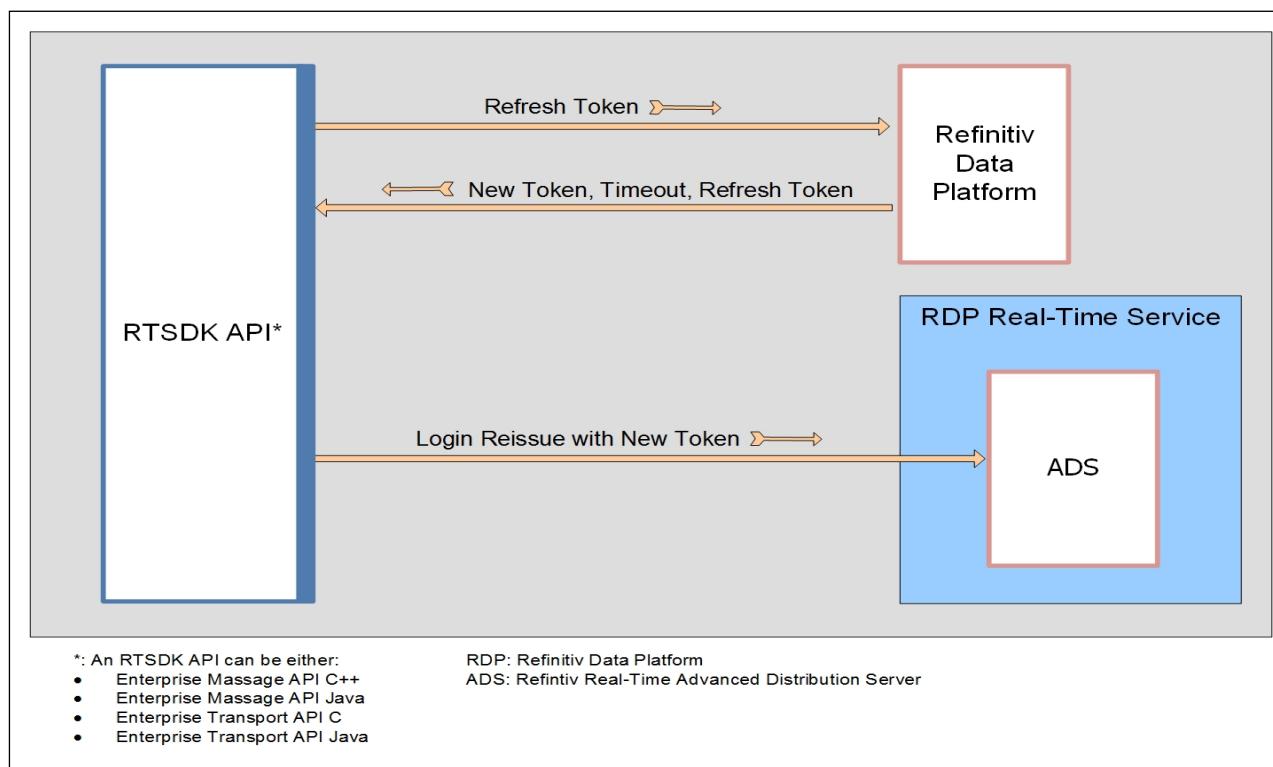


Figure 10. Login Reissue

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 .

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 Refinitiv Data 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 the Refinitiv Real-Time Optimized ADS, there is no need to renew the Access Token. The login session to the ADS will remain valid until the consumer disconnects or is disconnected from RTO. The API will only re-request an Access Token in the following scenarios:

- 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 6.10.2.1Section 6.9.2.1 of the *Enterprise Transport API C Edition Value Added Developers Guide*. The `Reactor` will handle the credentials the same way as Version 1, with an `RsslReactorOAuthCredentialEvent` callback for credentials if the user does not wish for the `Reactor` 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 Refinitiv 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 as a client secret above. For more information about the Platform Admin UI, reference the RRT Optimized documentation in the Refinitiv Developers portal.

NOTE: Please 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 Refinitiv Real-Time - Optimized (RTO).

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 Refinitiv Data Platform to obtain connection details for the Refinitiv Real-Time - Optimized. The Enterprise Transport API C Edition uses the **`rsslReactorQueryServiceDiscovery`** function (refer to Section 6.2.1 for a description of this reactor method) to submit a service discovery.

In response to a service discovery, the Refinitiv Data Platform returns transport and data format protocols and a list of hosts and associated ports for the requested service(s) (i.e., a Refinitiv Real-Time Advanced Distribution Server running in the cloud or endpoint). Refinitiv 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 Refinitiv Data Platform, the Enterprise Transport API identifies an RTO 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.

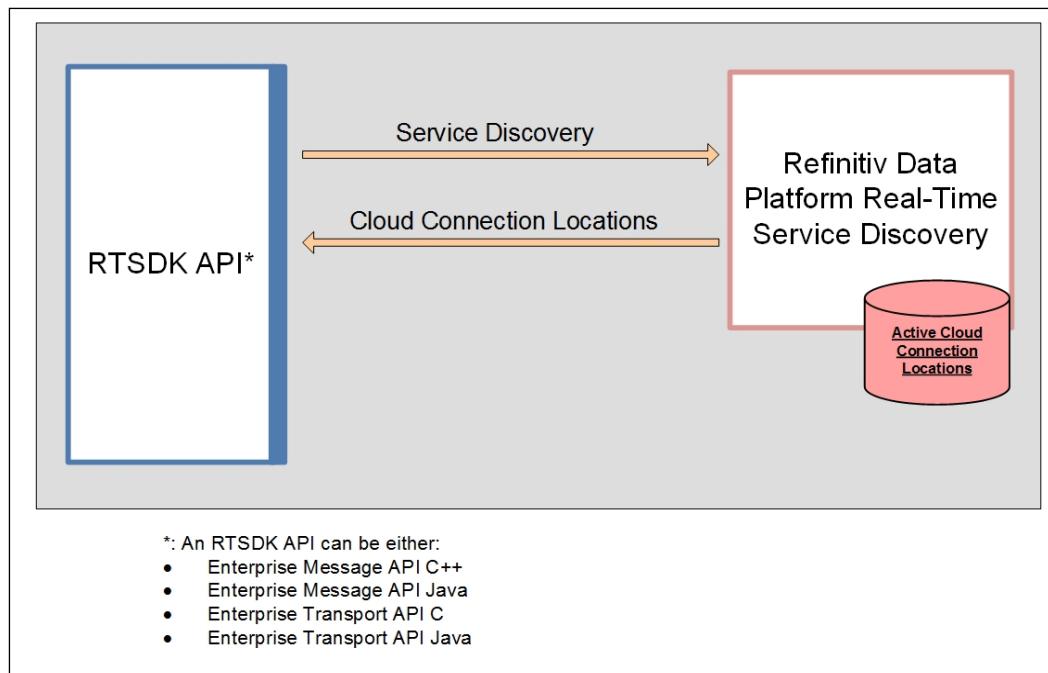
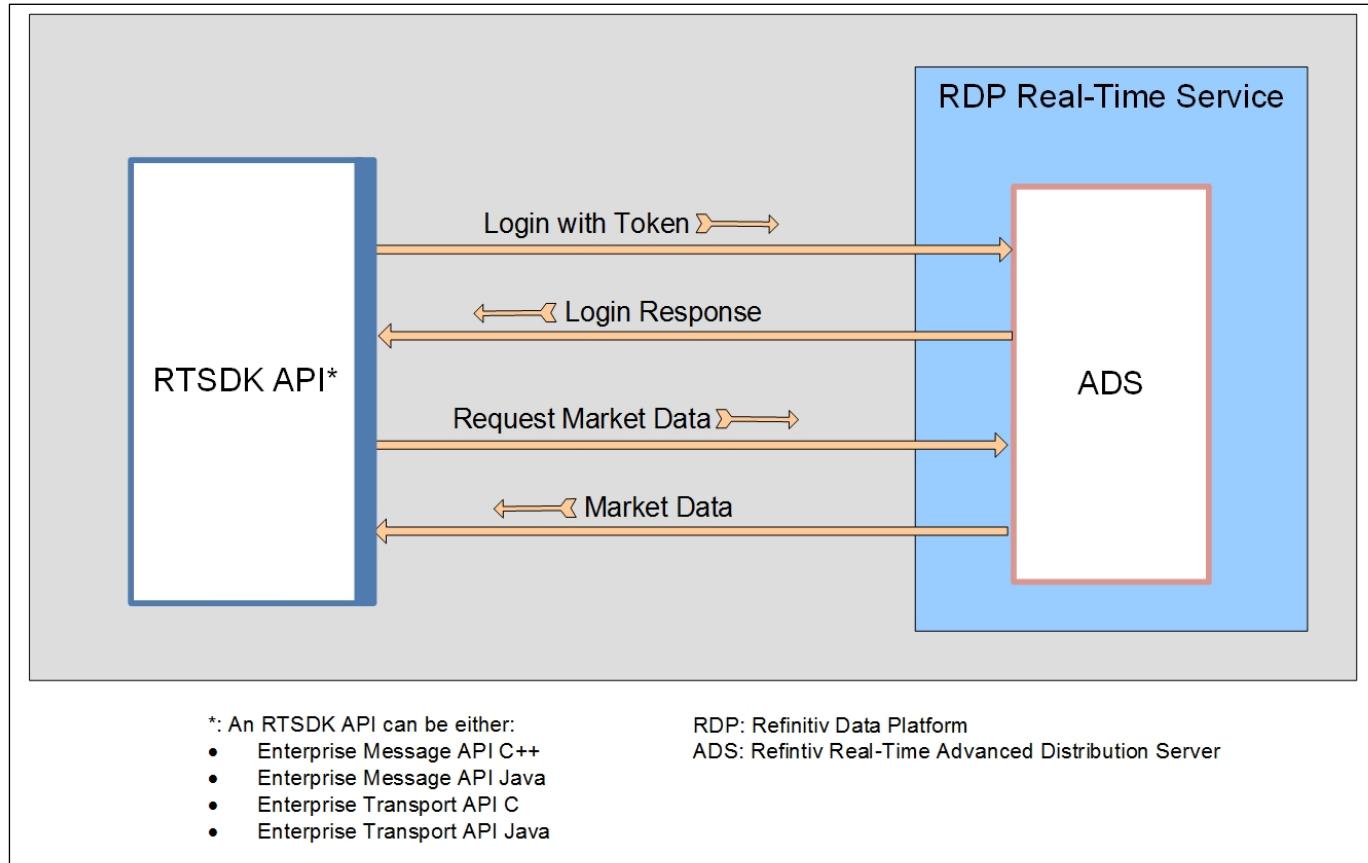


Figure 11. Service Discovery

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 Refinitiv 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 (for details, refer to Section 7.9.1)
- Disabling the watchlist (for details, refer to Section 7.9.2)
- Query service discovery (for details, refer to Section 7.9.3)

7.9.1 Session Management Use Case

In the session management use case, the Enterprise Transport API manages the entire connection from start to finish. To use session management, you need to configure the API to enable the watchlist and session management (i.e., in the `RsslReactorConnectInfo` object, set `enableSessionManagement`).

The API exhibits the following behavior (listed in order) when operating in a session management use case:

- Obtains a token (according to the details in Section 7.4.2)
- Queries service discovery (according to the details in Section 7.6)
- Consumes market data (according to the details in Section 7.7)
- Manages login reissues for Version 1 authentication when needed on a cyclical basis (according to the details in Section 7.4.3)

A special use case exists for connecting to a specific (i.e., non-default) host. As described in Section 7.6, by default the Enterprise Transport API connects to whichever host is setup for failover in the location specified by the API. If you want to connect to a specific, non-default host, you must set this in the `RsslConnectOptions.connectionInfo` options. In this case, the Enterprise Transport API exhibits the same behavior listed above, but ignores the endpoints it receives from the service discovery.

7.9.2 Disabling the Watchlist

When connecting to a Refinitiv 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 Query Service Discovery

In the query service discovery use case, the API user wants to connect to the Refinitiv Data Platform only for a service discovery, and does not necessarily want to consume market data. The API exhibits the following behavior (listed in order) when operating in a query service discovery use case:

- Obtains a token (according to the details in Section 7.4.2)
- 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 Refinitiv Data 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:

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).

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`

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 101). All structures follow the formatting and naming specified in the *Enterprise Transport API C Edition Refinitiv 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., Refinitiv 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 **Data Type** 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 **Map** or **ElementList**, 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 Refinitiv Domain Model Usage Guide* defines and describes all domain-specific behaviors, usage, and details.

DOMAIN	PURPOSE
Dictionary	<p>Provides dictionaries that may be needed when decoding data. Though use of the Dictionary domain is optional, Refinitiv recommends that provider applications support the domain's use.</p> <p>The Dictionary domain is considered an administrative domain. Many Refinitiv components require this content and expect it to follow the domain model definition.</p> <p>For further details refer to Section 8.5.</p>
Login	<p>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.</p> <p>Login is considered an administrative domain. Many Refinitiv components require this content and expect it to conform to the domain model definition.</p> <p>For further details refer to Section 8.3.</p>
Source Directory	<p>Advertises information about available services and their state, quality of service, and capabilities. This domain also conveys any group status and group merge information.</p> <p>Interactive and non-interactive provider applications require support for this domain. Refinitiv strongly recommends that consumers request this domain.</p> <p>Source Directory is considered an administrative domain, and many Refinitiv components expect this content and require it to conform to the domain model definition.</p> <p>For further details, refer to Section 8.4.</p>

Table 101: 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 **Rss1RDMMsgBase** 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 Refinitiv-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 Refinitiv-defined domain models in the Enterprise Transport API C Edition <i>Refinitiv 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 102: **Rss1RDMMsgBase** 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 (Rss1RDMLoginMsg) Refer to Section 8.3	RDM_LG_MT_REQUEST	Rss1RDMLoginRequest
	RDM_LG_MT_REFRESH	Rss1RDMLoginRefresh
	RDM_LG_MT_STATUS	Rss1RDMLoginStatus
	RDM_LG_MT_CLOSE	Rss1RDMLoginClose
	RDM_LG_MT_CONSUMER_CONNECTION_STATUS	Rss1RDMLoginConsumerConnectionStatus
RSSL_DMT_SOURCE (Rss1RDMDirectoryMsg) Refer to Section 8.4	RDM_DR_MT_REQUEST	Rss1RDMDirectoryRequest
	RDM_DR_MT_REFRESH	Rss1RDMDirectoryRefresh
	RDM_DR_MT_UPDATE	Rss1RDMDirectoryUpdate
	RDM_DR_MT_STATUS	Rss1RDMDirectoryStatus
	RDM_DR_MT_CLOSE	Rss1RDMDirectoryClose
	RDM_DR_MT_CONSUMER_STATUS	Rss1RDMDirectoryConsumerStatus

Table 103: **Rss1RDMMsg**

DOMAIN TYPE	RDM: MESSAGE TYPE	RDM: MESSAGE STRUCTURE
RSSL_DMT_DICTIONARY (Rss1RDMDictionaryMsg) Refer to Section 8.5	RDM_DC_MT_REQUEST	Rss1RDMDictionaryRequest
	RDM_DC_MT_REFRESH	Rss1RDMDictionaryRefresh
	RDM_DC_MT_STATUS	Rss1RDMDictionaryStatus
	RDM_DC_MT_CLOSE	Rss1RDMDictionaryClose

Table 103: Rss1RDMMsg(Continued)

8.2.3 RSSL RDM Encoding and Decoding Functions

Encode and decode functionality is provided that can take the **Rss1RDMMsg** 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
rss1EncodeRDMMMsg	Used to encode any message that the Rss1RDMMsg can represent. This function takes the Rss1RDMMsg as a parameter.
rss1DecodeRDMMMsg	Used to decode any message that the Rss1RDMMsg can represent. This function populates the Rss1RDMMsg and leverages the Value Added Utility message buffer (refer to Section 10.2). NOTE: The decoded message may refer to encoded data from the original Rss1Msg . If the message is to be stored, the appropriate copy function for the decoded Rss1RDMMsg should be used to create a full copy.

Table 104: 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.

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 **Rss1RDMLoginRequest** 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 *Refinitiv Domain Model Usage Guide*.

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.

8.3.1.1 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 RsslRDMLoginRequest using rsslInitDefaultRDMLoginRequest, 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 RsslRDMLoginRequest using rsslInitDefaultRDMLoginRequest, applicationName is set to upa.</p>
AuthenticationExtended	<p>Optional. If present, a flags value of RDM_LG_RQF_HAS_AUTHN_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>
password	<p>Optional. If present, a Flags value of RDM_LG_RQF_HAS_PASSWORD should be specified.</p> <p>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.</p>

Table 105: **RsslRDMLoginRequest** Structure Members

STRUCTURE MEMBER	DESCRIPTION
position	<p>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.</p> <p>If initializing Rss1RDMLoginRequest using rss1InitDefaultRDMLoginRequest, the IP address of the system the application is running on will be used.</p>
providePermissionExpressions	<p>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.</p> <p>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.</p>
providePermissionProfile	<p>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.</p> <p>When 1, this indicates that a consumer desires the permission profile. The permission profile can be used by an application to perform proxy permissioning.</p>
rdmMsgBase	<p>Required. Contains general message information like streamId and domainType. For more information, refer to Section 8.2.</p>
role	<p>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.</p> <p>Indicates the role of the application logging onto the system.</p> <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	<p>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.</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.
supportProviderDictionaryDownload	<p>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.</p> <p>Indicates whether the Refinitiv Real-Time Advanced Data Hub supports the Provider Dictionary Download feature, which allows the application to request RWFFId and RWFEnum dictionaries from a Refinitiv Real-Time Advanced Data Hub.</p> <ul style="list-style-type: none"> • 1: The Refinitiv Real-Time Advanced Data Hub supports the Provider Dictionary Download feature. • 0: The Refinitiv Real-Time Advanced Data Hub does not support the Provider Dictionary Download feature. <p>For details on the Provider Dictionary Download feature, refer to the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>

Table 105: Rss1RDMLoginRequest Structure Members (Continued)

STRUCTURE MEMBER	DESCRIPTION
UserName	<p>Required. Populate this member with the username, email address, or user token based on the UserNameType specification.</p> <p>If you initialize Rss1RDMLoginRequest using rss1InitDefaultRDMLoginRequest, it uses the name of the user currently logged into the system on which the application runs.</p>
UserNameType	<p>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.</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 Refinitiv 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>Refinitiv Real-Time Distribution System Authentication User Manual</i>.^a</p>

Table 105: Rss1RDMLoginRequest Structure Members (Continued)

a. For further details on Refinitiv Real-Time Distribution System Authentication, refer to the *Refinitiv Real-Time Distribution System Authentication User Manual*, accessible on [MyRefinitiv](#) 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 106: Rss1RDMLoginRequest 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 106: **Rss1RDMLoginRequest** Flags (Continued)

8.3.1.3 RSSL RDM Login Request Utility Functions

FUNCTION NAME	DESCRIPTION
rssIClearRDMLoginRequest	Clears an Rss1RDMLoginRequest structure. Useful for structure reuse.
rssIInitDefaultRDMLoginRequest	Clears an Rss1RDMLoginRequest structure and populates UserName , Position , ApplicationId , and ApplicationName with default values.
rssICopyRDMLoginRequest	Performs a deep copy of an Rss1RDMLoginRequest structure.

Table 107: **Rss1RDMLoginRequest** 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 **Rss1RDMLoginRefresh** 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 *Refinitiv 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 a Refinitiv Real-Time Distribution System Authentication environment, where 0 indicates an error-free condition. For further information, refer to the <i>Refinitiv 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>Refinitiv 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>Refinitiv 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>Refinitiv 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 108: **Rss1RDMLoginRefresh** 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	<p>Required. Contains general message information like streamId and domainType. (i.e.,</p>
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.
RsslState	<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.</p> <p>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 108: 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>Refinitiv 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 Refinitiv Real-Time Advanced Data 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 Refinitiv Real-Time Advanced Data Hub supports the Provider Dictionary Download feature. • 0: The Refinitiv Real-Time Advanced Data 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 108: Rss1RDMLoginRefresh 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 Refinitiv 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>Refinitiv Real-Time Distribution System Authentication User Manual</i>.^a</p>

Table 108: Rss1RDMLoginRefresh Structure Members (Continued)

a. For further details on Refinitiv Real-Time Distribution System Authentication, refer to the *Refinitiv Real-Time Distribution System Authentication User Manual*, accessible on [MyRefinitiv](#) 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 109: Rss1RDMLoginRefresh 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	<p>Indicates the presence of supportBatchRequests. If absent, a value of 0 should be assumed.</p> <p>For more information on Batch functionality, refer to the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>
RDM_LG_RFF_HAS_SUPP_POST	<p>Indicates the presence of supportOMMPost. If absent, a value of 0 should be assumed.</p> <p>For more information on Posting, refer to the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>
RDM_LG_RFF_HAS_SUPPORT_PROV_DIC_DOWNLOAD	<p>Indicates the presence of supportProviderDictionaryDownload. If absent, a value of 0 should be assumed.</p> <p>For more information on Provider Dictionary Download, refer to the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>
RDM_LG_RFF_HAS_SUPP_OPT_PAR	<p>Indicates the presence of supportOptimizedPauseResume. If absent, a value of 0 should be assumed.</p> <p>For more information on Pause and Resume, refer to the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>
RDM_LG_RFF_HAS_SUPP_VIEW	<p>Indicates the presence of supportViewRequests. If absent, a value of 0 should be assumed.</p> <p>For more information on View functionality, refer to the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>
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 109: Rss\RDMLoginRefresh Flags (Continued)

8.3.2.3 Login Refresh Utility Functions

FUNCTION NAME	DESCRIPTION
rssiClearRDMLoginRefresh	Clears an Rss1RDMLoginRefresh structure. Useful for structure reuse.
rssiCopyRDMLoginRefresh	Performs a deep copy of an Rss1RDMLoginRefresh structure.

Table 110: **Rss1RDMLoginRefresh** Utility Functions

8.3.2.4 Server Info Structure Members

STRUCTURE MEMBER	DESCRIPTION
Flags	Required. Indicates the presence of optional server information members. For details, refer to Section 8.3.2.5.
hostname	Required. Indicates the server's hostname .
loadFactor	Optional. Indicates the load information for this server. If present, a Flags value of RDM_LG_SIF_HAS_LOAD_FACTOR should be specified.
port	Required. Indicates the server's port number for connections.
RsslServerIndex	Required. Provides the index value to this server.
RsslServerType	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 RDMLoginServerTypes .

Table 111: **Rss1RDMServerInfo** Structure Members

8.3.2.5 Server Info Flag Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_LG_SIF_HAS_LOAD_FACTOR	Indicates presence of loadFactor information.
RDM_LG_SIF_HAS_TYPE	Indicates presence of RsslServerType .

Table 112: **Rss1RDMServerInfo** Flags

8.3.2.6 Server Info Utility Functions

FUNCTION NAME	DESCRIPTION
rssiClearRDMServerInfo	Clears an Rss1RDMServerInfo structure. Useful for structure reuse.

Table 113: **Rss1RDMServerInfo** 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 *Refinitiv Domain Models Usage Guide*.

8.3.3.1 Login Status Structure Members

STRUCTURE MEMBER	DESCRIPTION
AuthenticationErrorCode	<p>Optional. If present, a Flags value of RDM_LG_STF_HAS_AUTHN_ERROR_CODE should be specified.</p> <p>AuthenticationErrorCode is specific to deployments using Refinitiv 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>Refinitiv Real-Time Distribution System Authentication User Manual</i>.^a</p>
AuthenticationErrorText	<p>Optional. If present, a Flags value of RDM_LG_STF_HAS_AUTHN_ERROR_TEXT should be specified.</p> <p>Specifies any text associated with the specified AuthenticationErrorCode. For further information, refer to the <i>Refinitiv Real-Time Distribution System Authentication User Manual</i>.^a</p>
Flags	<p>Required. Indicates the presence of optional login status members. For details, refer to Section 8.3.3.2.</p>
rdmMsgBase	<p>Required. Contains general message information, such as streamId and domainType.</p>
RsslState	<p>Optional. If present, a flags value of RDM_LG_STF_HAS_STATE should be specified.</p> <p>Indicates the state of the login stream. When rejecting a login the state should be:</p> <ul style="list-style-type: none"> • streamState = RSSL_STREAM_CLOSED or RSSL_STREAM_CLOSED_RECOVER • DataState = RSSL_DATA_SUSPECT • RsslStateCode = RSSL_SC_NOT_ENTITLED <p>For more information on RsslState, refer to the Enterprise Transport API C Edition <i>Developers Guide</i>.</p>

Table 114: **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 Refinitiv 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>Refinitiv 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 114: Rss1RDMLoginStatus Structure Members (Continued)

a. For further details on Refinitiv Real-Time Distribution System Authentication, refer to the *Refinitiv Real-Time Distribution System Authentication User Manual*, accessible on [MyRefinitiv](#) 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	<p>Indicates the presence of RssiState.</p> <p>If absent, any previously conveyed state continues to apply.</p>
RDM_LG_STF_HAS_USERNAME	Indicates the presence of UserName .
RDM_LG_STF_HAS_USERNAME_TYPE	<p>Indicates the presence of UserNameType.</p> <p>If absent a UserNameType of RDM_LOGIN_USER_NAME is assumed.</p>

Table 115: Rss1RDMLoginStatus Flags

8.3.3.3 Login Status Utility Functions

FUNCTION NAME	DESCRIPTION
rssiClearRDMLoginStatus	Clears an Rss1RDMLoginStatus structure. Useful for structure reuse.
rssiCopyRDMLoginStatus	Performs a deep copy of an Rss1RDMLoginStatus structure.

Table 116: **Rss1RDMLoginStatus** 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
rdmMsgBase	Contains general message information like streamId and domainType .

Table 117: **Rss1RDMLoginClose** Structure Member

8.3.4.2 Login Close Utility Functions

FUNCTION NAME	DESCRIPTION
rssiClearRDMLoginClose	Clears an Rss1RDMLoginClose structure. Useful for structure reuse.
rssiCopyRDMLoginClose	Performs a deep copy of an Rss1RDMLoginClose structure.

Table 118: **Rss1RDMLoginClose** 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 **Rss1RDMLoginConsumerConnectionStatus** relies on the **Rss1GenericMsg** 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 *Refinitiv 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 <code>streamId</code> and <code>domainType</code> . Indicates the Login Message type (for login connection status, set to <code>LoginMsgType.CONSUMER_CONNECTION_STATUS</code>).
warmStandbyInfo	Optional. Includes <code>Rss1RDMLoginWarmStandbyInfo</code> to convey the state of the upstream provider. For details, refer to Section 8.3.5.3. If present, a flags value of <code>RDM_LG_CCSF_HAS_WARM_STANDBY_INFO</code> should be specified.

Table 119: `Rss1RDMLoginConsumerConnectionStatus` Structure Members

8.3.5.2 Login Consumer Connection Status Flag Enumeration Value

FLAG ENUMERATION	DESCRIPTION
<code>RDM_LG_CCSF_HAS_WARM_STANDBY_INFO</code>	Indicates presence of <code>warmStandbyInfo</code> .

Table 120: `Rss1RDMLoginConsumerConnectionStatus` 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 <code>Rss1MapEntry</code> actions, refer to the Enterprise Transport API C Edition Developers Guide.
warmStandbyMode	Required. Indicates the presence of optional login consumer connection status members. For details, refer to Section 8.3.5.4.

Table 121: `Rss1RDMLoginWarmStandbyInfo` Structure Members

8.3.5.4 Login Warm Standby Mode Enumeration Values

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

Table 122: `RDMLoginServerTypes` Enumeration Values

8.3.5.5 Login Consumer Connection Status Utility Functions

FUNCTION NAME	DESCRIPTION
rssiClearRDMLoginConsumerConnectionStatus	Clears an Rss1RDMLoginConsumerConnectionStatus structure. Useful for structure reuse.
rssiClearRDMLoginWarmStandbyInfo	Clears the Rss1RDMLoginWarmStandbyInfo structure.
rssiCopyRDMLoginConsumerConnectionStatusopy	Performs a deep copy of an Rss1RDMLoginConsumerConnectionStatus structure.

Table 123: **Rss1RDMLoginConsumerConnectionStatus** 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 124: 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 125: 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
rssIClearRDMRTT	Clears an RsslRDMLoginRTT structure for reuse.
rssICopyRDMRTT	Performs a deep copy of an RsslRDMLoginRTT structure.

Table 126: 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 **Rss1PostMsg**. 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 Rss1RDMLoginClose as described in Section 8.3.4.
consumerConnectionStatus	The Rss1RDMLoginConsumerConnectionStatus as described in Section 8.3.5.
rdmMsgBase	The message base information.
refresh	The Rss1RDMLoginRefresh as described in Section 8.3.2.
request	The Rss1RDMLoginRequest as described in Section 8.3.1.
RTT	The Rss1RDMLoginRTT as described in Section 8.3.6.
status	The Rss1RDMLoginStatus as described in Section 8.3.3.

Table 127: **Rss1RDMLoginMsg** Union Members

8.3.9.2 Login Message Utility Functions

FUNCTION NAME	DESCRIPTION
rssiClearRDMLoginMsg	Clears an Rss1RDMLoginMsg union. Useful for reuse.
rssiCopyRDMLoginMsg	Performs a deep copy of an Rss1RDMLoginMsg structure.

Table 128: **Rss1RDMLoginMsg** Utility Functions

8.3.10 RSSL RDM Login Encoding and Decoding

8.3.10.1 Directory Login Encoding and Decoding Functions

FUNCTION NAME	DESCRIPTION
rsslDecodeRDMLoginMsg	Decodes an RDM Login message. This function populates the RsslRDMLoginMsg and leverages the Value Added Utility message buffer (refer to Section 10.2). Alternatively, rsslDecodeRDMMsg can be used to decode into an RsslRDMMsg .
rsslEncodeRDMLoginMsg	Encodes an RDM Login message. This function takes the RsslRDMLoginMsg as a parameter. Alternately, rsslEncodeRDMMsg can be used if encoding from an RsslRDMMsg .

Table 129: RDM Login Encoding and Decoding Functions

8.3.10.2 Encoding a Login Request

```
RsslEncodeIterator encodeIter;
RsslRDMLoginRequest loginRequest;

/* Clear the Login Request structure. */
rsslClearRDMLoginRequest(&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 20: 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 21: Login Request Decoding Example

8.3.10.4 Encoding a Login Refresh

```
RsslEncodeIterator encodeIter;
RsslRDMLoginRefresh loginRefresh;

/* Clear the Login Refresh structure. */
rsslClearRDMLoginRefresh(&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, (RsslRDMMMsg*)&loginRefresh, &msgBuffer.length, &rsslErrorInfo);
```

Code Example 22: 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;
RsslRDMLoginRefresh *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 RsslRDMLoginRefresh. */
    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 23: 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 a Refinitiv Real-Time Advanced Data 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 **Rss1RDMDirectoryRequest** 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 *Refinitiv Domain Model Usage Guide*.

8.4.1.1 Directory Request Structure Members

STRUCTURE MEMBER	DESCRIPTION
filter	<p>Required. Indicates the service information in which the consumer is interested. The available flags are:</p> <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 <p>In most cases, you should set the RDM_DIRECTORY_SERVICE_INFO_FILTER, RDM_DIRECTORY_SERVICE_STATE_FILTER, and RDM_DIRECTORY_SERVICE_GROUP_FILTER.</p>
Flags	<p>Required. Indicates the presence of optional directory request members. For details, refer to Section 8.4.1.2.</p>
rdmMsgBase	<p>Required. Contains general message information like streamId and domainType.</p>
Rss1RDMServiceId	<p>Optional.</p> <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 130: **Rss1RDMDirectoryRequest** 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 131: **Rss1RDMDirectoryRequest** Flags

8.4.1.3 Directory Request Utility Functions

FUNCTION NAME	DESCRIPTION
rssiClearRDMDirectoryRequest	Clears an Rss1RDMDirectoryRequest structure. Useful for structure reuse.
rssiInitDefaultRDMDirectoryRequest	Clears an Rss1RDMDirectoryRequest , sets the structure to request all services and receive updates for them, and populates filter with default values.
rssiCopyRDMDirectoryRequest	Performs a deep copy of an Rss1RDMDirectoryRequest structure.

Table 132: **Rss1RDMDirectoryRequest** 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 **Rss1RDMDirectoryRefresh** 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 Refinitiv 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 Rss1RDMDirectoryRequest . 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 133: **Rss1RDMDirectoryRefresh** 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 RsslRDMServiceList .
RsslRDMServiceId	Optional. If present, a Flags value of RDM_DR_RFF_HAS_SERVICE_ID should be specified, which should match the RsslRDMServiceId of the consumer's RsslRDMDirectoryRequest .
RsslRDMServiceList	Optional. Presence indicated by serviceCount . Contains an array of information about available services.
RsslState	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 133: 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 RsslRDMServiceId .
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 134: RsslRDMDirectoryRefresh Flags

8.4.2.3 Directory Refresh Utility Functions

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

Table 135: 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 *Refinitiv Domain Model Usage Guide*.

8.4.3.1 Directory Update Structure Members

STRUCTURE MEMBER	DESCRIPTION
filter	<p>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.</p> <p>Available flags are:</p> <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	<p>Required. Indicates the presence of optional directory update members. For details refer to Section 8.4.3.2.</p>
SequenceNumber	<p>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.</p>
serviceCount	<p>Required. Indicates the number of services present in the RsslRDMServiceList.</p>
RsslRDMServiceId	<p>Optional. This member's value must match the RsslRDMServiceId of the consumer's RsslRDMDirectoryRequest. If present, a Flags value of RDM_DR_UPF_HAS_SERVICE_ID should be specified.</p>
RsslRDMServiceList	<p>Optional. Presence indicated by serviceCount. Contains an array of information about available services.</p>
rdmMsgBase	<p>Required. Contains general message information like streamId and domainType.</p>

Table 136: **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 RsslRDMServiceId .

Table 137: **RsslRDMDirectoryUpdate** Flags

8.4.3.3 Directory Update Utility Functions

FUNCTION NAME	DESCRIPTION
rssIClearRDMDirectoryUpdate	Clears an Rss1RDMDirectoryUpdate structure. Useful for structure reuse.
rssICopyRDMDirectoryUpdate	Performs a deep copy of an Rss1RDMDirectoryUpdate structure.

Table 138: **Rss1RDMDirectoryUpdate** 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 **Rss1RDMDirectoryStatus** 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 *Refinitiv Domain Model Usage Guide*.

8.4.4.1 Directory Status Structure Members

STRUCTURE MEMBER	DESCRIPTION
filter	Optional. If present, a Flags value of RDM_DR_STF_HAS_FILTER should be specified. Indicates what information is being provided about supported services. This should match the filter of the consumer's Rss1RDMDirectoryRequest . 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 status members. For details, refer to Section 8.4.4.2.
RsslRDMServiceId	Optional. If present, a Flags value of RDM_DR_STF_HAS_SERVICE_ID should be specified. This member should match the RsslRDMServiceId of the consumer's Rss1RDMDirectoryRequest .
RsslState	Optional. Indicates the state of the directory stream. If present, a Flags value of RDM_DR_STF_HAS_STATE should be specified. For more information on RsslState , refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
rdmMsgBase	Required. Contains general message information like streamId and domainType .

Table 139: **Rss1RDMDirectoryStatus** 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 Rss1RDMServiceId .
RDM_DR_STF_HAS_STATE	Indicates the presence of Rss1State . If not present, any previously conveyed state should continue to apply.

Table 140: **Rss1RDMDirectoryStatus Flags**

8.4.4.3 Directory Status Utility Functions

FUNCTION NAME	DESCRIPTION
rssiClearRDMDirectoryStatus	Clears an Rss1RDMDirectoryStatus structure. Useful for structure reuse.
rssiCopyRDMDirectoryStatusopy	Performs a deep copy of an Rss1RDMDirectoryStatus structure.

Table 141: **Rss1RDMDirectoryStatus 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 142: **Rss1RDMDirectoryClose Structure Member**

8.4.5.2 Directory Close Utility Functions

FUNCTION NAME	DESCRIPTION
rssiClearRDMDirectoryClose	Clears an Rss1RDMDirectoryClose structure. Useful for structure reuse.
rssiCopyRDMDirectoryClose	Performs a deep copy of an Rss1RDMDirectoryClose structure.

Table 143: **Rss1RDMDirectoryClose 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 *Refinitiv 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 144: **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> .
RsslRDMServiceId	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 145: **RsslRDMConsumerStatusService** Structure Members

8.4.6.3 Directory Consumer Status Utility Functions

FUNCTION NAME	DESCRIPTION
rssiClearRDMDirectoryConsumerStatus	Clears an RsslRDMDirectoryConsumerStatus structure. Useful for structure reuse.
rssiClearRDMConsumerStatusService	Clears the RsslRDMConsumerStatusService structure.
rssiCopyRDMDirectoryConsumerStatus	Performs a deep copy of an RsslRDMDirectoryConsumerStatus structure.

Table 146: **RsslRDMDirectoryConsumerStatus** Utility Functions

8.4.7 Source Directory RDM Service

An **RsslRDMService** structure conveys information about a service. An array of **RsslRDMServices** forms the **RsslRDMServiceList** 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 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.
RsslRDMServiceId	Required . Indicates the service associated with this RsslRDMService .
RsslState	Optional . Contains information related to the Source Directory State Filter. If present, a Flags value of RDM_SVCF_HAS_STATE should be specified.

Table 147: **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 148: **RsslRDMService** Flags

8.4.7.3 RSSL Service Utility Functions

FUNCTION NAME	DESCRIPTION
rssIClearRDMService	Clears an Rss1RDMService structure. Useful for structure reuse.

Table 149: **Rss1RDMService** Utility Function

8.4.8 Source Directory RDM Service Info

An **Rss1RDMServiceInfo** structure conveys information that identifies the service and the content it provides. The **Rss1RDMServiceInfo** structure represents the Source Directory Info filter. More information about the Info filter is available in the Enterprise Transport API C Edition *Refinitiv Domain Model Usage Guide*.

8.4.8.1 RSSL Service Info Members

STRUCTURE MEMBER	DESCRIPTION
acceptingConsumerStatus	Optional. Indicates whether this service supports accepting Rss1RDMDirectoryConsumerStatus 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.
Action	Required. Indicates how a service info cache should apply this information. For information on Rss1FilterEntry actions, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
capabilitiesCount	Required. Indicates the number of capabilities present in the capabilitiesList .
capabilitiesList	Required. Contains a list of capabilities that the service supports. Populated by domain types.
dictionariesProvidedCount	Optional. Indicates the number of elements present in DictionariesProvided . If present, a Flags value of RDM_SVC_IFF_HAS_DICTS_PROVIDED should be specified.
DictionariesProvidedList	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.
dictionariesUsedCount	Optional. Indicates the number of elements present in DictionariesUsed . If present, a Flags value of RDM_SVC_IFF_HAS_DICTS_USED should be specified.
DictionariesUsedList	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.
Flags	Required. Indicates the presence of optional service info members. For details, refer to Section 8.4.8.2.

Table 150: **Rss1RDMServiceInfo** Structure Members

STRUCTURE MEMBER	DESCRIPTION
isSource	<p>Optional. Indicates whether the service is provided directly by a source or represents a group of sources.</p> <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. <p>If present, a Flags value of RDM_SVC_IFF_HAS_IS_SOURCE should be specified.</p>
itemList	<p>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.</p> <p>If present, a Flags value of RDM_SVC_IFF_HAS_ITEM_LIST should be specified.</p>
qosCount	<p>Optional. Indicates the number of elements present in RsslQosList.</p> <p>If present, a Flags value of RDM_SVC_IFF_HAS_QOS should be specified.</p>
RsslQosList	<p>Optional. Contains an array of elements that identify the available Qualities of Service.</p> <p>If present, a Flags value of RDM_SVC_IFF_HAS_QOS and the qosCount should be specified.</p>
RsslRDMServiceName	<p>Required. Indicates the name of the service.</p>
SupportsOutOfBandSnapshots	<p>Optional. Indicates whether this service supports making snapshot requests even when the OpenLimit is reached. Available values are:</p> <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. <p>If present, a Flags value of RDM_SVC_IFF_HAS_SUPPORT_OOB_SNAPSHOTS should be specified.</p>
SupportsQosRange	<p>Optional. Indicates whether this service supports specifying a range of Qualities of Service when requesting an item. For further information, refer to the RsslQos and worstQos members of the RsslRequestMsg in the Enterprise Transport API C Edition <i>Developers Guide</i>. Available values are:</p> <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. <p>If present, a Flags value of RDM_SVC_IFF_HAS_SUPPORT_QOS_RANGE should be specified.</p>
vendor	<p>Optional. Identifies the vendor of the data.</p> <p>If present, a Flags value of RDM_SVC_IFF_HAS_VENDOR should be specified.</p>

Table 150: **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 151: **RsslRDMServiceInfo** Flags

FLAG ENUMERATION	DESCRIPTION
RDM_SVC_IFF_HAS_ITEM_LIST	Indicates the presence of <code>itemList</code> .
RDM_SVC_IFF_HAS_QOS	Indicates the presence of <code>Rss1QosList</code> and <code>qosCount</code> .
RDM_SVC_IFF_HAS_SUPPORT_OOB_SNAPSHOTS	Indicates the presence of <code>SupportsOutOfBandSnapshots</code> .
RDM_SVC_IFF_HAS_SUPPORT_QOS_RANGE	Indicates the presence of <code>SupportsQosRange</code> .
RDM_SVC_IFF_HAS_VENDOR	Indicates the presence of <code>vendor</code> .

Table 151: `Rss1RDMServiceInfo` Flags (Continued)

8.4.8.3 RSSL Service Info Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rssiClearRDMServiceInfo</code>	Clears an <code>Rss1RDMServiceInfo</code> structure. Useful for structure reuse.

Table 152: `Rss1RDMServiceInfo` Utility Functions

8.4.9 Source Directory RDM Service State

An `Rss1RDMServiceState` 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 *Refinitiv Domain Model Usage Guide*.

8.4.9.1 RSSL Service State Members

STRUCTURE MEMBER	DESCRIPTION
<code>acceptingRequests</code>	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, <code>Flags</code> value of <code>RDM_SVC_STF_HAS_ACCEPTING_REQS</code> should be specified.
<code>Action</code>	Required . Indicates how a cache of the service state should apply this information. For details on <code>Rss1FilterEntry</code> actions, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
<code>Flags</code>	Required . Indicates the presence of optional service state members. For details refer to Section 8.4.9.2.
<code>RssiRDMServiceState</code>	Required . Indicates whether the original provider of the data can respond to new requests. Requests can still be made if so indicated by <code>acceptingRequests</code> . 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.
<code>Status</code>	This status should be applied to all open items associated with this service. If present, <code>Flags</code> value of <code>RDM_SVC_STF_HAS_STATUS</code> should be specified.

Table 153: `Rss1RDMServiceState` 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 154: **Rss1RDMServiceState** Flags

8.4.9.3 RSSL Service State Utility Functions

FUNCTION NAME	DESCRIPTION
rssIClearRDMServiceState	Clears an Rss1RDMServiceState structure. Useful for structure reuse.

Table 155: **Rss1RDMServiceState** Utility Functions

8.4.10 Source Directory RDM Service Group State

An **Rss1RDMServiceGroupState** 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 *Refinitiv 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 Rss1FilterEntry 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_GRF_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_GRF_HAS_STATUS should be specified.

Table 156: **Rss1RDMServiceGroupState** 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 157: **Rss1RDMServiceGroupState** Flags

8.4.10.3 RSSL Service Group State Utility Functions

FUNCTION NAME	DESCRIPTION
rssIClearRDMServiceGroupState	Clears an Rss1RDMServiceGroupState structure. Useful for structure reuse.

Table 158: **Rss1RDMServiceGroupState** Utility Functions

8.4.11 Source Directory RDM Service Load

An **Rss1RDMServiceLoad** 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 *Refinitiv Domain Model Usage Guide*.

8.4.11.1 RSSL Service Load Members

STRUCTURE MEMBER	DESCRIPTION
Action	Required . Indicates how a cache of the service load should apply this information. For information on Rss1FilterEntry actions, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
Flags	Required . Indicates presence of optional service load members. For details, refer to Section 8.4.11.2.
loadFactor	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>Refinitiv Domain Model Usage Guide</i> .
openLimit	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.
openWindow	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 159: **Rss1RDMServiceLoad** 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 160: **Rss1RDMServiceLoad** Flags

8.4.11.3 RSSL Service Load Utility Functions

FUNCTION NAME	DESCRIPTION
rssIClearRDMServiceLoad	Clears a Rss1RDMServiceLoad structure. Useful for structure reuse.

Table 161: **Rss1RDMServiceLoad** Utility Functions

8.4.12 Source Directory RDM Service Data

An **Rss1RDMServiceData** 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 *Refinitiv 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 Rss1FilterEntry actions, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .
Data	Optional . Contains the encoded Rss1Buffer 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 Rss1DataType of the data. For information on Rss1DataTypes , 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 162: **Rss1RDMServiceData** 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 163: **Rss1RDMServiceData** Flags

8.4.12.3 RSSL Service Data Utility Functions

FUNCTION NAME	DESCRIPTION
rssiClearRDMServiceData	Clears an Rss1RDMServiceData structure. Useful for structure reuse.

Table 164: **Rss1RDMServiceData** Utility Functions

8.4.13 Source Directory RDM Service Link Information

An **Rss1RDMServiceLinkInfo** 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 *Refinitiv Domain Model Usage Guide*.

The **Rss1RDMServiceLinkInfo** structure contains an array of **Rss1RDMServiceLink** 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 Rss1FilterEntry 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 Rss1RDMServiceLink structures, each representing a source.

Table 165: **Rss1RDMServiceLinkInfo** Structure Members

8.4.13.2 RSSL Service Link Info Utility Functions

FUNCTION NAME	DESCRIPTION
rssiClearRDMServiceLinkInfo	Clears an Rss1RDMServiceLinkInfo structure. Useful for structure reuse.

Table 166: **Rss1RDMServiceLinkInfo** 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 *Refinitiv 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 167: **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 168: **RsslRDMServiceLink** Flags

8.4.14.3 RSSL Service Link Utility Functions

FUNCTION NAME	DESCRIPTION
rssIClearRDMServiceLink	Clears an Rss1RDMServiceLink structure. Useful for structure reuse.

Table 169: **Rss1RDMServiceLink** Utility Functions

8.4.15 Source Directory RDM Sequenced Multicast Information

An **Rss1RDMServiceSeqMcastInfo** structure is included in the services advertised by the Reference Data Server component of an Refinitiv 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 Refinitiv Real Time Direct, refer to the *Refinitiv Real Time Direct Developers Guide*.

8.4.15.1 RSSL Service Sequenced Multicast Information Structure

STRUCTURE MEMBER	DESCRIPTION
flags	Required . Sets any optional members. For details, refer to Section 8.4.15.2.
action	Required . Sets how a cache should apply this information. For details, refer to the <i>Transport API C Edition Developers Guide</i> .
snapshotServer	Sets the network address/port information for the Snapshot Server, if one is present.
gapRecoveryServer	Sets the network address/port information for the Gap Recovery Server, if one is present.
refDataServer	Sets the network address/port information for the Reference Data Server, if one is present.
StreamingMCastChanServerCount	The number of real time stream components in StreamingMCastChanServerList .
StreamingMCastChanServerList	Sets the network address/port information for real time stream components.
GapMCastChanServerCount	Number of Gap Fill Server components in GapMCastServerList .
GapMCastChanServerList	Sets the network address/port information for Gap Fill Server components.

Table 170: **Rss1RDMServiceSeqMcastInfo** Structure Members

8.4.15.2 RSSL Service Sequenced Multicast Info Enumeration Values

FLAG ENUMERATION	DESCRIPTION
RDM_SVC_SMF_HAS_SNAPSHOT_SERV	Indicates the presence of snapshotServer .
RDM_SVC_SMF_HAS_GAP_REC_SERV	Indicates the presence of gapRecoveryServer .
RDM_SVC_SMF_HAS_REF_DATA_SERV	Indicates the presence of refDataServer .
RDM_SVC_SMF_HAS_SMC_SERV	Indicates the presence of StreamingMCastChanServerList .
RDM_SVC_SMF_HAS_GMC_SERV	Indicates the presence of GapMCastChanServerList .

Table 171: 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 172: RSSL RDM Address/Port Information Structure Members

8.4.15.4 RSSL Sequenced Multicast Info Utility Functions

UTILITY	DESCRIPTION
rssIClearRDMMCAddressPortInfo	Clears an Rss1RDMAddressPortInfo structure.
rssIClearRDMServiceSeqMCastInfo	Clears an Rss1RDMServiceSeqMCastInfo structure.

Table 173: **Rss1RDMServiceSeqMcastInfo** 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 Rss1RDMDirectoryRequest as described in Section 8.4.1.
close	The Rss1RDMDirectoryClose as described in Section 8.4.5.
refresh	The Rss1RDMDirectoryRefresh as described in Section 8.4.2.
status	The Rss1RDMDirectoryStatus as described in Section 8.4.4.
update	The Rss1RDMDirectoryUpdate as described in Section 8.4.3.
consumerStatus	The Rss1RDMDirectoryConsumerStatus as described in Section 8.4.6.

Table 174: **Rss1RDMDirectoryMsg** Union Members

8.4.16.2 Directory Message Utility Functions

FUNCTION NAME	DESCRIPTION
rssIClearRDMDirectoryMsg	Clears an Rss1RDMDirectoryMsg union. Useful for reuse.
rssICopyRDMDirectoryMsgopy	Performs a deep copy of an Rss1RDMDirectoryMsg structure.

Table 175: **Rss1RDMDirectoryMsg** Utility Functions

8.4.17 Source Directory Encoding and Decoding

8.4.17.1 RSSL RDM Directory Encoding and Decoding Functions

FUNCTION NAME	DESCRIPTION
rsslEncodeRDMDirectoryMsg	Used to encode an RDM Directory message. This function takes the RsslRDMDirectoryMsg as a parameter. Alternately, rsslEncodeRDMMMsg can be used if encoding from an RsslRDMMMsg .
rsslDecodeRDMDirectoryMsg	Used to decode an RDM Directory message. This function populates the RsslRDMDirectoryMsg and leverages the Value Added Utility message buffer (refer to Section 10.2). Alternately, rsslDecodeRDMMMsg can be used to decode into an RsslRDMMMsg .

Table 176: 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 = rsslEncodeRDMMMsg(&encodeIter, (RsslRDMMMsg*)&directoryRequest, &msgBuffer.length,
    &rsslErrorInfo);
```

Code Example 24: 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;
RsslRDMMMsg 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 RsslRDMMMsg structure. */
ret = rsslDecodeRDMMMsg(&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 25: 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 = "Refinitiv";
serviceList[0].info.vendor.length = 9;

/* 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 = "RWFEnum";
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 26: 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 27: 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 **Rss1FieldEntry** 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 *Refinitiv 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 *Refinitiv 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 **Rss1RDMDictionaryRequest** 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 *Refinitiv 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 177: **Rss1RDMDictionaryRequest** 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>Refinitiv Domain Model Usage Guide</i> .

Table 178: **Rss1RDMDictionaryRequest** Flag

8.5.1.3 Dictionary Request Utility Functions

FUNCTION NAME	DESCRIPTION
rssiClearRDMDictionaryRequest	Clears an Rss1RDMDictionaryRequest structure. Useful for structure reuse.
rssiCopyRDMDictionaryRequest	Performs a deep copy of an Rss1RDMDictionaryRequest structure.

Table 179: **Rss1RDMDictionaryRequest** 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 **Rss1RDMDictionaryRefresh** 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 *Refinitiv Domain Model Usage Guide*.

8.5.2.1 Dictionary Refresh Structure Members

STRUCTURE MEMBER	DESCRIPTION
dataBody	When decoding, this points to the encoded data buffer with dictionary content. This buffer should be set on an Rss1DecodeIterator and passed to the appropriate decode function according to the type . Not used when encoding. The dictionary is retrieved from the Rss1DataDictionary structure.
pDictionary	Conditional (required when encoding). Points to an Rss1DataDictionary object that contains content to encode. For more information on the Rss1DataDictionary structure, refer to the Enterprise Transport API C Edition <i>Refinitiv Domain Model Usage Guide</i> . Not used when decoding.
dictionaryId	When decoding, this will be populated with the dictionary's ID. The Dictionary is retrieved from the Rss1DataDictionary structure. This structure's presence is indicated by the RDM_DC_RFF_HAS_INFO flag. Not used when encoding.
dictionaryName	Required . Indicates the name of the dictionary being provided.
flags	Required . Indicates the presence of optional dictionary refresh members. For details, refer to Section 8.5.2.2.
rdmMsgBase	Required . Contains general message information like streamId and domainType .
sequenceNumber	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.
serviceId	Required . Indicates the service ID of the service from which the dictionary is provided.
startFid	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 180: **Rss1RDMDictionaryRefresh** 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 Rss1State , 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 == 0xF: 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 Rss1DataDictionary structure.

Table 180: Rss1RMDictionaryRefresh 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 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 181: Rss1RMDictionaryRefreshFlags

8.5.2.3 Dictionary Refresh Utility Functions

FUNCTION NAME	DESCRIPTION
rssiClearRDMDictionaryRefresh	Clears an Rss1RDMDictionaryRefresh structure. Useful for structure reuse.
rssiCopyRDMDictionaryRefresh	Performs a deep copy of an Rss1RDMDictionaryRefresh structure.

Table 182: **Rss1RDMDictionaryRefresh** 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 *Refinitiv Domain Model Usage Guide*.

The **Rss1RDMDictionaryStatus** 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 *Refinitiv Domain Model Usage Guide*.

8.5.3.1 Dictionary Status Structure Members

STRUCTURE MEMBER	DESCRIPTION
flags	Required . Indicate the presence of optional dictionary status members. For details, refer to Section 8.5.3.2.
rdmMsgBase	Required . Contains general message information like streamId and domainType .
state	Optional . Indicates the state of the dictionary stream. For more information on Rss1State , 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 183: **Rss1RDMDictionaryStatus** Structure Members

8.5.3.2 Dictionary Status Flag Enumeration Value

FLAG ENUMERATION	DESCRIPTION
RDM_DC_STF_CLEAR_CACHE	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.
RDM_DC_STF_HAS_STATE	Indicates the presence of state . If absent, any previously conveyed state continues to apply.

Table 184: **Rss1RDMDictionaryStatus** Flags

8.5.3.3 Dictionary Status Utility Functions

FUNCTION NAME	DESCRIPTION
rssiClearRDMLoginStatus	Clears an RsslRDMLoginStatus structure. Useful for structure reuse.
rssiCopyRDMLoginStatus	Performs a deep copy of an RsslRDMLoginStatus structure.

Table 185: **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
rdmMsgBase	Required. Contains general message information like <code>streamId</code> and <code>domainType</code> .

Table 186: `Rss1RDMDictionaryClose` Structure Members

8.5.4.2 Dictionary Close Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rssiClearRDMDictionaryClose</code>	Clears an <code>Rss1RDMDictionaryClose</code> structure. Useful for structure reuse.
<code>rssiCopyRDMDictionaryClose</code>	Performs a deep copy of an <code>Rss1RDMDictionaryClose</code> structure.

Table 187: `Rss1RDMDictionaryClose` 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 <code>Rss1RDMDictionaryRequest</code> as described in Section 8.5.1.
<code>close</code>	The <code>Rss1RDMDictionaryClose</code> as described in Section 8.5.4.
<code>refresh</code>	The <code>Rss1RDMDictionaryRefresh</code> as described in Section 8.5.2.
<code>status</code>	The <code>Rss1RDMDictionaryStatus</code> as described in Section 8.5.3.

Table 188: `Rss1RDMDictionaryMsg` Union Members

8.5.5.2 Dictionary Message Utility Functions

FUNCTION NAME	DESCRIPTION
<code>rssiClearRDMDictionaryMsg</code>	Clears an <code>Rss1RDMDictionaryMsg</code> union. Useful for reuse.
<code>rssiCopyRDMDictionaryMsg</code>	Performs a deep copy of an <code>Rss1RDMDictionaryMsg</code> structure.

Table 189: `Rss1RDMDictionaryMsg` Utility Functions

8.5.6 Dictionary Encoding and Decoding

8.5.6.1 RSSL RDM Dictionary Encoding and Decoding Functions

FUNCTION NAME	DESCRIPTION
rsslEncodeRDMDictionaryMsg	Used to encode an RDM Dictionary message. This function takes the RsslRDMDictionaryMsg as a parameter. Alternately, rsslEncodeRDMMMsg can be used if encoding from an RsslRDMMMsg .
rsslDecodeRDMDictionaryMsg	Used to decode an RDM Directory message. This function populates the RsslRDMDictionaryMsg and leverages the Value Added Utility message buffer (refer to Section 10.2). Alternately, rsslDecodeRDMMMsg can be used to decode into an RsslRDMMMsg .

Table 190: 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 = rsslEncodeRDMMMsg(&encodeIter, (RsslRDMMMsg*)&dictionaryRequest, &msgBuffer.length,
    &rsslErrorInfo);
```

Code Example 28: 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;
RsslRDMMMsg 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 RsslRDMMMsg structure. */
ret = rsslDecodeRDMMMsg(&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 29: 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. */
rsslClearRMDDictionaryRefresh(&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 30: 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 31: 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: Refinitiv 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 191: **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 Rss1State , 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 192: Rss1RDMQueueRefresh 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 Rss1State , refer to the Enterprise Transport API C Edition <i>Developers Guide</i> .

Table 193: Rss1RDMQueueStatus 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 194: Rss1RDMQueueClose 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 195: **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 196: Queue Data Flag

8.6.6.3 Queue Message Timeout Codes

Queue message timeout codes are special codes that can be set on the `Rss1RDMQueueData.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 197: Rss1RDMQueueTimeoutCodes

8.6.6.4 Queue Data Encoding

The `Rss1RDMQueueData` 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>rssiEncodeRDMQueueMsg</code>	When sending no payload or payload content is preencoded and specified on the <code>Rss1RDMQueueData.encDataBody</code> buffer, this method encodes the <code>Rss1RDMQueueData</code> message in a single call.
<code>rssiEncodeRDMQueueMsgComplete</code>	Completes the content encoding into this <code>Rss1RDMQueueData</code> message.
<code>rssiEncodeRDMQueueMsgInit</code>	Begins the process of encoding content into this <code>Rss1RDMQueueData</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. 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.

Table 198: 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 32: 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 **Rss1RDMQueueDataExpired** messages on the queue stream to OMM consumer applications.

OMM consumer applications do not send this message.

8.6.7.1 Rss1RDMQueueDataExpired 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 Rss1RDMQueueData 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 Rss1RDMQueueData 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 Rss1RDMQueueData 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 199: Rss1RDMQueueDataExpired 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 200: Rss1RDMQueueDataUndeliverableCodes

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 200: Rss1RDMQueueDataUndeliverableCodes (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 Rss1RDMQueueData.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 201: Rss1RDMQueueAck

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:

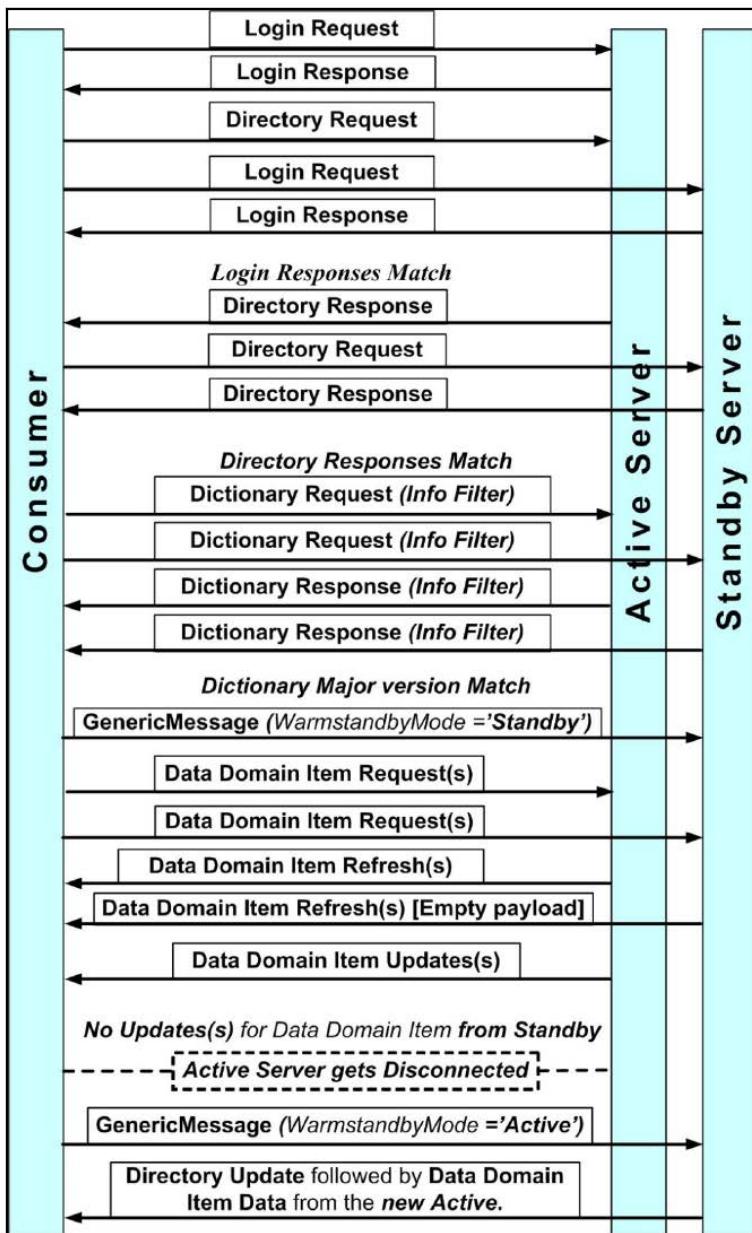


Figure 12. Login Based Warm Standby Order of Events in a Cutover from Active to Standby

The following figure illustrates the sequence of events when using the Service Based Warm Standby feature:

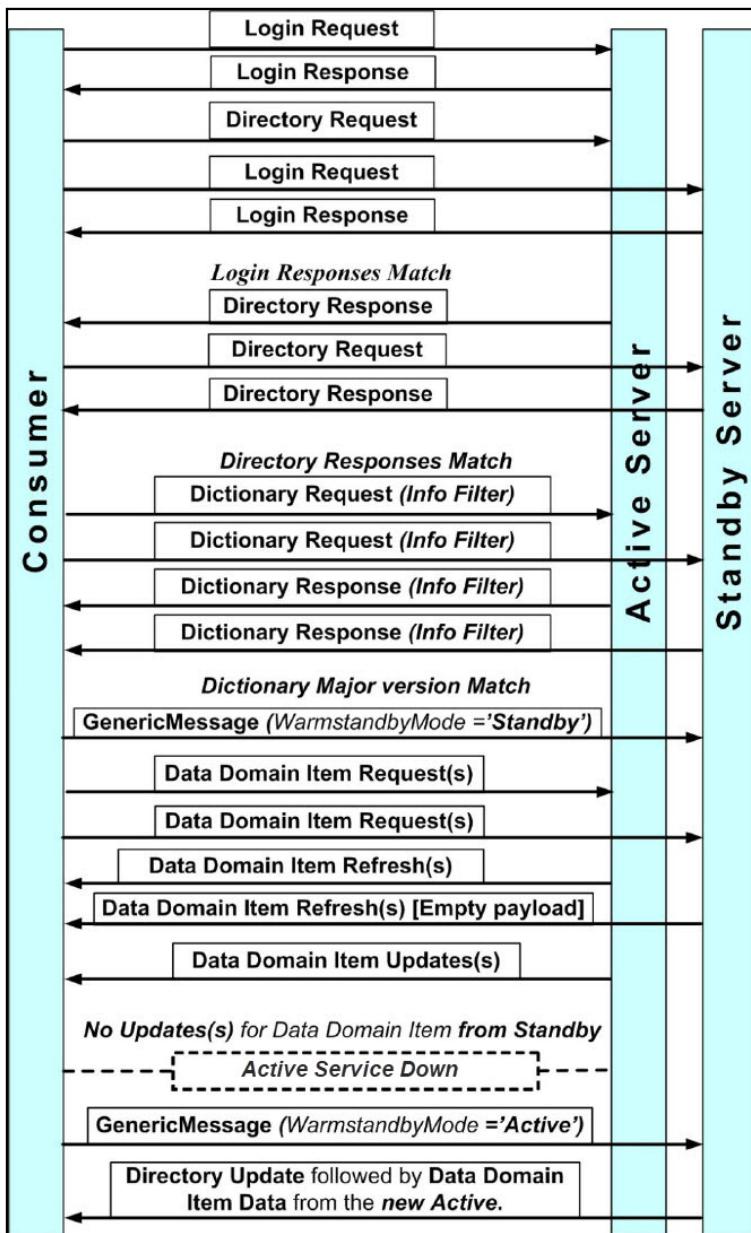


Figure 13. Service Based Warm Standby Order of Events in a Cutover from Active to Standby

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.uni-
    fied.address = "dataserver1";
reactorWarmStandbyGroup.startingActiveServer.reactorConnectInfo.rsslConnectOptions.connectionInfo.uni-
    fied.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 33: Warm Standby Configuration Example

9.4

10 Value Added Utilities

10.1 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.

10.2 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
rsslReserveBufferMemory	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.
rsslReserveAlignedBufferMemory	Reserves memory from an RsslBuffer . Similar to rsslReserveBufferMemory , but will ensure that the memory is aligned on an appropriate word boundary. Subsequent calls to the non-aligned rsslReserveBufferMemory 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.
rsslCopyBufferMemory	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 202: Memory Buffer Functions

10.3 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 34: Memory Buffer Example

11 Payload Cache Detailed View

11.1 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.

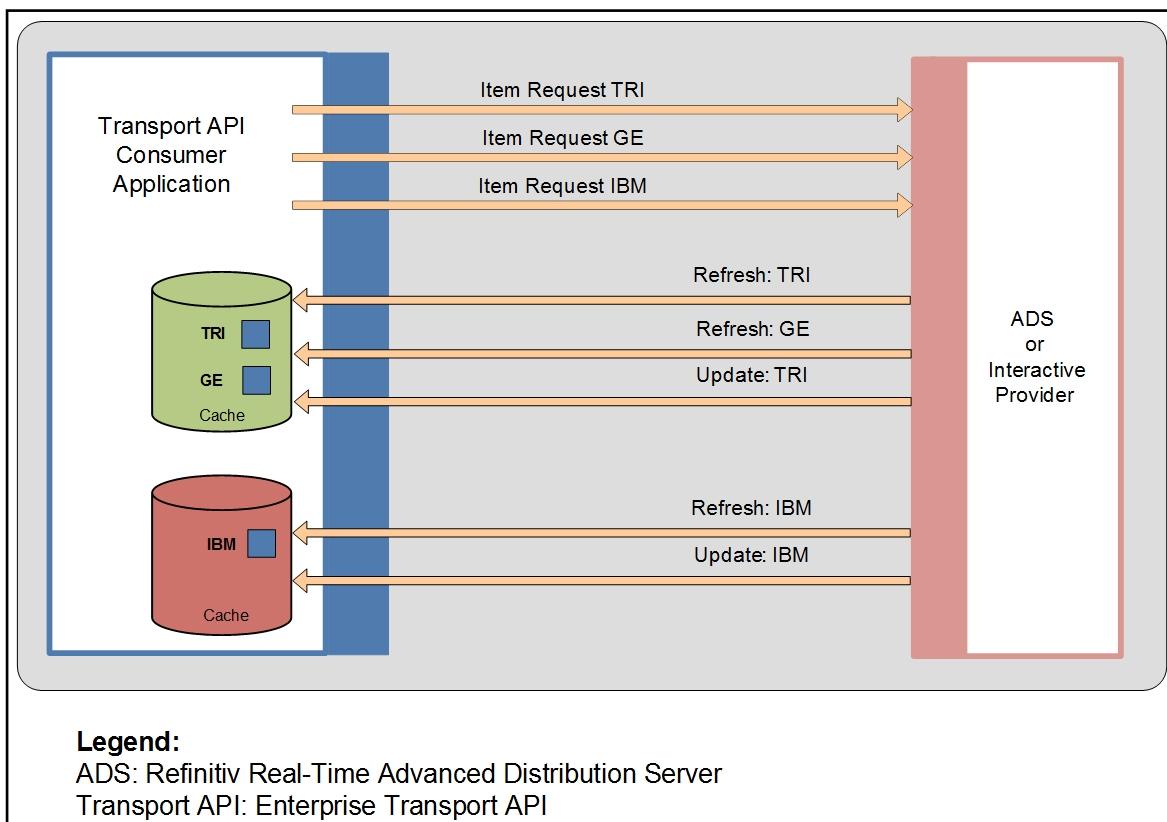


Figure 14. Consumer Application using Cache to Store Payload Data for Item Streams

11.2 Payload Cache

This section describes how the payload cache is managed (initialization and uninitalization), and how instances of cache (collections of payload entries) are created and destroyed.

11.2.1 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
<code>rsslPayloadCacheInitialize</code>	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 <code>rsslPayloadCacheUninitialize</code> must be made before the component is uninitialized.
<code>rsslPayloadCacheUninitialize</code>	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.
<code>rsslPayloadCacheIsInitialized</code>	This function can be used by an application to determine if the payload cache component has already been initialized (by a call to <code>rsslPayloadCacheInitialize</code>).

Table 203: Payload Cache Management Functions

11.2.2 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.

11.2.2.1 Cache Error Structure Members

The `RsslCacheError` has the following structure members:

STRUCTURE MEMBER	DESCRIPTION
<code>rsslErrorId</code>	Specifies an error ID. The range of values is defined by the set of Enterprise Transport API return codes (from the <code>RsslReturnCodes</code> enumeration).
<code>text</code>	This <code>char []</code> will contain text with additional information when a function call returns a failed result. The size of the buffer is fixed to <code>MAX_OMM_CACHE_ERROR_TEXT</code> as defined on the cache interface.

Table 204: RsslCacheError Structure Members

11.2.2.2 Clearing a Cache Error

The following function clears the **Rss1CacheError**.

STRUCTURE MEMBER	DESCRIPTION
rss1CacheErrorClear	Clears the Rss1CacheError structure. Use this function prior to passing the structure to a cache interface function.

Table 205: Function for Cache Error Handling

11.2.3 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 11.2.3.1. Before using payload caches, you must first have initialized this function using **rss1PayloadCacheInitialize** as described in Section 11.2.1.

11.2.3.1 Managing Payload Instances

FUNCTION NAME	DESCRIPTION
rss1PayloadCacheCreate	Creates a payload cache instance, and returns the Rss1PayloadCacheHandle . All operations on the cache require this handle. Options are passed in via the Rss1PayloadCacheConfigOptions defined in Section 11.2.3.2.
rss1PayloadCacheDestroy	Destroys a payload cache instance. Any entries remaining in the cache are also destroyed at this time.

Table 206: Functions for Managing Cache Instances

11.2.3.2 Payload Cache Structure Member

STRUCTURE MEMBER	DESCRIPTION
maxItems	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 rss1PayloadEntryCreate function will return a null Rss1PayloadEntryHandle when the maximum number of items is reached. When set to zero, the cache allows an unlimited number of items. Refer to Section 11.3.1.

Table 207: **Rss1PayloadCacheConfigOptions** Structure Members

11.2.4 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 **Rss1DataDictionary** structure loaded with the field dictionary. The dictionary structure can be loaded from a file (using the **rss1LoadFieldDictionary** function) or from an encoded dictionary message from a provider (using the **rss1DecodeFieldDictionary** function). The cache does not use the enumerated dictionary content, so loading the enumeration dictionary is not required. For more information on using **Rss1DataDictionary**, refer to the *Enterprise Transport API Reference Manual*.

After the **Rss1DataDictionary** 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 11.2.4.1.

The cache builds its own field definition database from the **Rss1DataDictionary** definitions. After binding, the application does not need to retain the dictionary structure, because the cache does not refer to the **Rss1DataDictionary** 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 11.2.4.3.

11.2.4.1 Setting Functions

FUNCTION NAME	DESCRIPTION
<code>rss1PayloadCacheBindDictionary</code>	Deprecated. For details on using this function, refer to the <i>Enterprise Transport API C Edition Developers Guide</i> . Otherwise, for equivalent functionality, refer to <code>rss1PayloadCacheSetDictionary</code> in this table.
<code>rss1PayloadCacheBindSharedDictionaryKey</code>	Deprecated. For details on using this function, refer to the <i>Enterprise Transport API C Edition Developers Guide</i> . Otherwise, for equivalent functionality, refer to <code>rss1PayloadCacheSetSharedDictionaryKey</code> in this table.
<code>rss1PayloadCacheSetDictionary</code>	This function sets an Rss1DataDictionary to a cache instance (identified by Rss1PayloadCacheHandle). 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. 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 Rss1DataDictionary structure, enabling the dictionary to be extended. Refer to Section 11.2.4.3.
<code>rss1PayloadCacheSetSharedDictionaryKey</code>	Use this function when sharing a dictionary among multiple caches. This function sets a cache (identified by the Rss1PayloadCacheHandle) 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 <code>rss1PayloadCacheSetDictionary</code> function. This function does not require the Rss1DataDictionary structure, since that was already loaded during the initial setting with this dictionary key.

Table 208: Functions for Setting Dictionary to Cache

11.2.4.2 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 35: Creating Cache and Setting to Dictionary

11.2.4.3 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.

11.2.5 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 it's data dictionary.

Table 209: Payload Cache Utility Functions

11.3 Payload Cache Entries

A payload cache entry stores a single OMM container (whose data types are defined by **Rss1ContainerType**). 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.

11.3.1 Managing Payload Cache Entries

Payload cache entries are created within a cache instance. A cache entry is defined by the **Rss1PayloadEntryHandle** 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 **Rss1PayloadEntryHandle** 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 11.2.5.

Use the following functions to manage cache entries:

FUNCTION NAME	DESCRIPTION
rsslPayloadEntryCreate	This method returns a Rss1PayloadEntryHandle to the newly created entry in the cache defined by the given Rss1PayloadCacheHandle . The Rss1PayloadEntryHandle 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 Rss1PayloadCacheConfigOptions would be exceeded).
rsslPayloadEntryDestroy	This method destroys the cache entry defined by Rss1PayloadEntryHandle and removes it from its cache.
rsslPayloadEntryClear	This method deletes any data in the cache entry Rss1PayloadEntryHandle and returns the entry to its initial state. The entry itself remains in the cache and can be re-used.

Table 210: Payload Cache Entry Management Functions

11.3.2 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 < RSSI_RET_SUCCESS)
{
    printf("Error (%d) applying data to cache entry: %s\n", cacheError.rsslErrorId,
           cacheError.text);
    return ret;
}

return ret;
}

```

Code Example 36: Applying Data to a Payload Cache Entry

11.3.3 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
<code>rsslPayloadEntryGetDataType</code>	Returns the <code>RsslContainerType</code> stored in the cache entry (<code>RsslPayloadEntryHandle</code>). When initially created (or after the entry is cleared), the data type will be <code>RSSL_DT_UNKNOWN</code> . The data type is defined by the container type of the first refresh message applied to the entry.
<code>rsslPayloadEntryApply</code>	Applies the OMM data in the payload of the <code>RsslMsg</code> to the cache entry (<code>RsslPayloadEntryHandle</code>). The first message applied must be a refresh message (class <code>RSSL_MC_REFRESH</code>).
<code>rsslPayloadEntryRetrieve</code>	Retrieves data from the cache entry by encoding the OMM container into the buffer provided with the <code>RsslEncodeIterator</code> given by the application. For single-part retrieval, the <code>RsslPayloadCursorHandle</code> parameter is optional. For details on multi-part retrieval, refer to Section 11.3.3.1.

Table 211: Functions for Applying and Retrieving Cache Entry Data

11.3.3.1 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
<code>rsslPayloadCursorCreate</code>	Creates a cursor for optional use in the <code>rsslPayloadEntryRetrieve</code> function (required for multi-part retrieval). Returns the <code>RsslPayloadCursorHandle</code> .
<code>rsslPayloadCursorDestroy</code>	Destroys the cursor referenced by the <code>RsslPayloadCursorHandle</code> .
<code>rsslPayloadCursorClear</code>	Clears the state of the cursor for the given <code>RsslPayloadCursorHandle</code> . Whenever retrieving data from a cache entry, the cursor must be cleared prior to the first call to <code>rsslPayloadEntryRetrieve</code> . Clearing the cursor also allows it to be reused with a retrieval on a different container.
<code>rsslPayloadCursorIsComplete</code>	Returns the completion state of a retrieval where the <code>RsslPayloadCursorHandle</code> was used. The state must be checked after each call to <code>rsslPayloadEntryRetrieve</code> 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 212: Functions for Using the Payload Cursor

11.3.3.2 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.

11.3.3.3 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 used 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 37: Cache Retrieval with Multi-Part Support

© 2015 - 2023 Refinitiv. All rights reserved.

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

Any third party names or marks are the trademarks or registered trademarks of the relevant third party.

Document ID: ETAC371UMVAC.230

Date of issue: June 2023

