

Enterprise Transport API C Edition 3.5.1.L1

VALUE ADDED COMPONENTS

Document Version: 3.5.1
Date of issue: September 2020
Document ID: ETAC351UMVAC.200



© **Refinitiv 2015 - 2020**. 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 | 3 |
| 1.7 | Document Conventions..... | 3 |
| 1.7.1 | <i>Optional, Conditional, and Required</i> | 3 |
| 1.7.2 | <i>Typographic</i> | 3 |
| 1.7.3 | <i>Document Structure</i> | 3 |
| 1.7.4 | <i>Diagrams</i> | 4 |
| 2 | Product Description and Overview | 5 |
| 2.1 | What is the Enterprise Transport API? | 5 |
| 2.2 | What are Enterprise Transport API Value Added Components? | 5 |
| 2.3 | Enterprise Transport API Reactor | 6 |
| 2.4 | Open Message Model Consumer Watchlist..... | 7 |
| 2.4.1 | <i>Data Stream Aggregation and Recovery</i> | 7 |
| 2.4.2 | <i>Additional Features</i> | 7 |
| 2.4.3 | <i>Usage Notes</i> | 7 |
| 2.5 | Administration Domain Model Representations | 8 |
| 2.6 | Value Added Utilities | 8 |
| 2.7 | Value Added Cache | 8 |
| 3 | Building an Open Message Model Consumer | 9 |
| 3.1 | Overview | 9 |
| 3.2 | Leverage Existing or Create New RsslReactor | 9 |
| 3.3 | Implement Callbacks and Populate Role | 9 |
| 3.4 | Establish Connection using rsslReactorConnect | 10 |
| 3.5 | Issue Requests and/or Post Information | 10 |
| 3.6 | Log Out and Shut Down..... | 10 |
| 3.7 | Additional Consumer Details | 11 |
| 4 | Building an Open Message Model Interactive Provider | 12 |
| 4.1 | Overview | 12 |
| 4.2 | Leverage Existing or Create New RsslReactor | 12 |
| 4.3 | Create an RsslServer | 12 |
| 4.4 | Implement Callbacks and Populate Role | 13 |
| 4.5 | Associate Incoming Connections Using rsslReactorAccept..... | 13 |
| 4.6 | Perform Login Process..... | 13 |
| 4.7 | Provide Source Directory Information | 14 |
| 4.8 | Provide or Download Necessary Dictionaries | 15 |
| 4.9 | Handle Requests and Post Messages | 15 |
| 4.10 | Dispatch Round Trip Time Messages | 16 |
| 4.11 | Disconnect Consumers and Shut Down | 16 |
| 4.12 | Additional Interactive Provider Details | 16 |
| 5 | Building an Open Message Model Non-Interactive Provider | 17 |
| 5.1 | Building an Open Message Model Non-Interactive Provider Overview | 17 |

| | | |
|----------|--|-----------|
| 5.2 | Leverage Existing or Create New RsslReactor | 17 |
| 5.3 | Implement Callbacks and Populate Role | 17 |
| 5.4 | Establish Connection using rsslReactorConnect | 18 |
| 5.5 | Perform Dictionary Download | 18 |
| 5.6 | Provide Content | 18 |
| 5.7 | Log Out and Shut Down | 19 |
| 5.8 | Additional Non-Interactive Provider Details..... | 19 |
| 6 | Reactor Detailed View | 20 |
| 6.1 | Concepts | 20 |
| 6.1.1 | <i>Functionality: Enterprise Transport API Versus Enterprise Transport API Reactor</i> | 21 |
| 6.1.2 | <i>Reactor Error Handling</i> | 22 |
| 6.1.3 | <i>Reactor Error Info Codes</i> | 22 |
| 6.1.4 | <i>Enterprise Transport API Reactor Application Lifecycle</i> | 23 |
| 6.2 | Reactor Use | 24 |
| 6.2.1 | <i>Creating a Reactor</i> | 24 |
| 6.2.2 | <i>Destroying a Reactor</i> | 26 |
| 6.3 | Reactor Channel Use | 27 |
| 6.3.1 | <i>Reactor Channel Roles</i> | 28 |
| 6.3.2 | <i>Reactor Channel Role: Open Message Model Consumer</i> | 30 |
| 6.3.3 | <i>Reactor Channel Role: Open Message Model Provider</i> | 32 |
| 6.3.4 | <i>Reactor Channel Role: Open Message Model Non-Interactive Provider</i> | 33 |
| 6.3.5 | <i>Reactor Channel: Role Union</i> | 34 |
| 6.4 | Managing Reactor Channels | 35 |
| 6.4.1 | <i>Adding Reactor Channels</i> | 35 |
| 6.4.2 | <i>Removing Reactor Channels</i> | 41 |
| 6.5 | Reporting on Channel Statistics | 41 |
| 6.6 | Dispatching Data | 41 |
| 6.6.1 | <i>rsslReactorDispatch Function</i> | 42 |
| 6.6.2 | <i>Reactor Callback Functions</i> | 43 |
| 6.6.3 | <i>Reactor Callback: Channel Event</i> | 44 |
| 6.6.4 | <i>Reactor Callback: Default Message</i> | 47 |
| 6.6.5 | <i>Reactor Callback: Refinitiv Domain Model Login Message</i> | 48 |
| 6.6.6 | <i>Reactor Callback: Refinitiv Domain Model Directory Message</i> | 50 |
| 6.6.7 | <i>Reactor Callback: Refinitiv Domain Model Dictionary Message</i> | 51 |
| 6.7 | Writing Data | 53 |
| 6.7.1 | <i>Writing Data using rsslReactorSubmitMsg()</i> | 53 |
| 6.7.2 | <i>Writing data using rsslReactorSubmit()</i> | 56 |
| 6.8 | Creating and Using Tunnel Streams | 63 |
| 6.8.1 | <i>Authenticating a Tunnel Stream</i> | 64 |
| 6.8.2 | <i>Opening a Tunnel Stream</i> | 64 |
| 6.8.3 | <i>Negotiating Stream Behaviors: Class of Service</i> | 66 |
| 6.8.4 | <i>Tunnel Stream Callback Functions and Event Types</i> | 69 |
| 6.8.5 | <i>Opening a Tunnel Stream Code Sample</i> | 72 |
| 6.8.6 | <i>Accepting Tunnel Streams</i> | 73 |
| 6.8.7 | <i>Receiving Content on a TunnelStream</i> | 78 |
| 6.8.8 | <i>Sending Content on a TunnelStream</i> | 78 |
| 6.9 | Cloud Connectivity | 82 |
| 6.9.1 | <i>rsslReactorQueryServiceDiscovery</i> | 82 |
| 6.9.2 | <i>OAuth Credential Management</i> | 84 |
| 6.10 | JSON to Refinitiv Wire Format Protocol Conversion for WebSocket Support | 87 |
| 6.10.1 | <i>Consumer and WebSocket Support</i> | 87 |
| 6.10.2 | <i>Interactive Provider and WebSocket Support</i> | 89 |
| 6.10.3 | <i>RsslReactorJsonConverterOptions Structure</i> | 91 |
| 6.10.4 | <i>RsslReactorJsonConversionEventCallback Function</i> | 92 |

| | | |
|----------|---|------------|
| 6.10.5 | <i>RsslReactorServiceNameToldCallback Function</i> | 92 |
| 6.10.6 | <i>RsslReactorServiceNameToldEvent Structure</i> | 93 |
| 6.10.7 | <i>RsslReactorJsonConversionEvent Structure</i> | 93 |
| 6.11 | Reactor Utility Functions | 94 |
| 6.11.1 | <i>General Reactor Utility Functions</i> | 94 |
| 6.11.2 | <i>RsslReactorChannelInfo Structure Members</i> | 94 |
| 6.11.3 | <i>rsslReactorIoctl Option Values</i> | 94 |
| 7 | Consuming Data from the Cloud | 95 |
| 7.1 | Overview | 95 |
| 7.2 | Encrypted Connections | 95 |
| 7.3 | Authentication Token Management | 96 |
| 7.3.1 | <i>Client_ID (AppKey)</i> | 96 |
| 7.3.2 | <i>Obtaining Initial Access and Refresh Tokens</i> | 96 |
| 7.3.3 | <i>Refreshing the Access Token and Sending a Login Reissue</i> | 97 |
| 7.3.4 | <i>Managing the Password and Client Secret</i> | 98 |
| 7.3.5 | <i>Session Management per User Credential</i> | 98 |
| 7.4 | Service Discovery | 99 |
| 7.5 | Consuming Market Data | 100 |
| 7.6 | HTTP Error Handling for Reactor Token Reissues | 100 |
| 7.7 | Cloud Connection Use Cases | 101 |
| 7.7.1 | <i>Session Management Use Case</i> | 101 |
| 7.7.2 | <i>Disabling the Watchlist</i> | 101 |
| 7.7.3 | <i>Query Service Discovery</i> | 101 |
| 7.8 | Logging of Authentication and Service Discovery Interaction | 102 |
| 7.8.1 | <i>Logged Request Information</i> | 102 |
| 7.8.2 | <i>Logged Response Information</i> | 102 |
| 8 | Administration Domain Models Detailed View | 103 |
| 8.1 | Concepts | 103 |
| 8.2 | Refinitiv Domain Model Message Base | 104 |
| 8.2.1 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Message Base Structure Members</i> | 104 |
| 8.2.2 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Message Types</i> | 104 |
| 8.2.3 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Encoding and Decoding Functions</i> | 105 |
| 8.3 | Refinitiv Domain Model Login Domain | 105 |
| 8.3.1 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Login Request</i> | 105 |
| 8.3.2 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Login Refresh</i> | 110 |
| 8.3.3 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Login Status</i> | 116 |
| 8.3.4 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Login Close</i> | 118 |
| 8.3.5 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Consumer Connection Status</i> | 118 |
| 8.3.6 | <i>Login Round Trip Time Message Use</i> | 121 |
| 8.3.7 | <i>Login Post Message Use</i> | 122 |
| 8.3.8 | <i>Login Ack Message Use</i> | 122 |
| 8.3.9 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Login Message Union</i> | 122 |
| 8.3.10 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Login Encoding and Decoding</i> | 123 |
| 8.4 | Refinitiv Source Sink Library Refinitiv Domain Model Source Directory Domain | 128 |
| 8.4.1 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Directory Request</i> | 128 |
| 8.4.2 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Directory Refresh</i> | 129 |
| 8.4.3 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Directory Update</i> | 131 |
| 8.4.4 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Directory Status</i> | 132 |
| 8.4.5 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Directory Close</i> | 133 |
| 8.4.6 | <i>Refinitiv Source Sink Library Refinitiv Domain Model Consumer Status</i> | 134 |
| 8.4.7 | <i>Source Directory Refinitiv Domain Model Service</i> | 135 |
| 8.4.8 | <i>Source Directory Refinitiv Domain Model Service Info</i> | 136 |
| 8.4.9 | <i>Source Directory Refinitiv Domain Model Service State</i> | 138 |

| | | |
|-----------|--|------------|
| 8.4.10 | Source Directory Refinitiv Domain Model Service Group State..... | 139 |
| 8.4.11 | Source Directory Refinitiv Domain Model Service Load..... | 140 |
| 8.4.12 | Source Directory Refinitiv Domain Model Service Data | 141 |
| 8.4.13 | Source Directory Refinitiv Domain Model Service Link Information | 142 |
| 8.4.14 | Source Directory Refinitiv Domain Model Service Link | 143 |
| 8.4.15 | Source Directory Refinitiv Domain Model Sequenced Multicast Information..... | 144 |
| 8.4.16 | Refinitiv Source Sink Library Refinitiv Domain Model Directory Message Union | 145 |
| 8.4.17 | Source Directory Encoding and Decoding..... | 146 |
| 8.5 | Dictionary Domain..... | 153 |
| 8.5.1 | Refinitiv Source Sink Library Refinitiv Domain Model Dictionary Request | 153 |
| 8.5.2 | Refinitiv Source Sink Library Refinitiv Domain Model Dictionary Refresh | 154 |
| 8.5.3 | Refinitiv Source Sink Library Refinitiv Domain Model Dictionary Status | 156 |
| 8.5.4 | Refinitiv Source Sink Library Refinitiv Domain Model Dictionary Close | 158 |
| 8.5.5 | Refinitiv Source Sink Library Refinitiv Domain Model Dictionary Message Union..... | 158 |
| 8.5.6 | Dictionary Encoding and Decoding..... | 159 |
| 8.6 | Refinitiv Domain Model Queue Messages | 164 |
| 8.6.1 | Queue Data Message Persistence | 164 |
| 8.6.2 | Queue Request..... | 164 |
| 8.6.3 | Queue Refresh | 165 |
| 8.6.4 | Queue Status..... | 165 |
| 8.6.5 | Queue Close..... | 165 |
| 8.6.6 | Queue Data | 166 |
| 8.6.7 | QueueDataExpired | 169 |
| 8.6.8 | Queue Ack..... | 170 |
| 9 | Value Added Utilities | 171 |
| 9.1 | Utility Overview | 171 |
| 9.2 | Memory Buffer..... | 171 |
| 9.3 | Using the Memory Buffer | 172 |
| 10 | Payload Cache Detailed View | 173 |
| 10.1 | Concepts | 173 |
| 10.2 | Payload Cache..... | 174 |
| 10.2.1 | Managing the Payload Cache..... | 174 |
| 10.2.2 | Cache Error Handling | 174 |
| 10.2.3 | Payload Cache Instances | 175 |
| 10.2.4 | Managing Refinitiv Domain Model Field Dictionaries for Payload Cache..... | 176 |
| 10.2.5 | Payload Cache Utilities..... | 178 |
| 10.3 | Payload Cache Entries..... | 179 |
| 10.3.1 | Managing Payload Cache Entries | 179 |
| 10.3.2 | Applying Data | 180 |
| 10.3.3 | Retrieving Data | 181 |

List of Figures

| | | |
|------------|---|-----|
| Figure 1. | Network Diagram Notation | 4 |
| Figure 2. | UML Diagram Notation..... | 4 |
| Figure 3. | Open Message Model APIs with Value Added Components | 5 |
| Figure 4. | Enterprise Transport API Value Added Components..... | 6 |
| Figure 5. | Enterprise Transport API Reactor Thread Model..... | 20 |
| Figure 6. | Enterprise Transport API Reactor Application Lifecycle | 23 |
| Figure 7. | Flow Chart for writing data via rssIReactorSubmit | 56 |
| Figure 8. | Tunnel Stream Illustration | 63 |
| Figure 9. | Obtaining an Authentication Token | 96 |
| Figure 10. | Login Reissue | 97 |
| Figure 11. | Service Discovery | 99 |
| Figure 12. | Consumer Application using Cache to Store Payload Data for Item Streams | 173 |

List of Tables

| | | |
|-----------|--|----|
| Table 1: | Acronyms and Abbreviations | 1 |
| Table 2: | Enterprise Transport API Functionality and Enterprise Transport API Reactor Comparison | 21 |
| Table 3: | RsslErrorInfo Structure Members | 22 |
| Table 4: | Reactor Error Info Codes | 22 |
| Table 5: | RsslReactor Structure Members | 24 |
| Table 6: | RsslReactor Creation Function | 24 |
| Table 7: | RsslCreateReactorOptions Structure Members | 25 |
| Table 8: | RsslCreateReactorOptions Utility Function | 25 |
| Table 9: | RsslReactor Destruction Function | 26 |
| Table 10: | RsslReactorChannel Structure Members | 27 |
| Table 11: | RsslReactorChannelRoleBase Structure Members | 28 |
| Table 12: | RsslReactorChannelRoleBase.role Enumerated Values | 29 |
| Table 13: | RsslReactorOMMConsumerRole Structure Members | 30 |
| Table 14: | RsslReactorOMMConsumerRole.dictionaryDownloadMode Enumerated Values | 31 |
| Table 15: | Open Message Model Consumer Role Watchlist Options | 31 |
| Table 16: | RsslReactorOMMConsumerRole Utility Function | 32 |
| Table 17: | RsslReactorOMMProviderRole Structure Members | 32 |
| Table 18: | RsslReactorOMMProviderRole Utility Function | 32 |
| Table 19: | RsslReactorOMNIProviderRole Structure Members | 33 |
| Table 20: | RsslReactorOMNIProviderRole Utility Function | 33 |
| Table 21: | RsslReactorChannelRole Union Members | 34 |
| Table 22: | RsslReactorChannelRole Utility Function | 34 |
| Table 23: | rsslReactorConnect Function | 35 |
| Table 24: | RsslReactorConnectOptions Structure Members | 36 |
| Table 25: | RsslReactorConnectInfo Structure Members | 37 |
| Table 26: | RsslReactorConnectOptions Utility Function | 37 |
| Table 27: | rsslReactorAccept Function | 39 |
| Table 28: | RsslReactorAcceptOptions Structure Members | 39 |
| Table 29: | RsslReactorAcceptOptions Utility Function | 40 |
| Table 30: | rsslReactorCloseChannel Function | 41 |
| Table 31: | rsslReactorDispatch Function | 42 |
| Table 32: | RsslReactorDispatchOptions Structure Members | 42 |
| Table 33: | RsslReactorDispatchOptions Utility Function | 42 |
| Table 34: | RsslReactorCallbackRet Callback Return Codes | 43 |
| Table 35: | RsslReactorChannelEvent Structure Members | 44 |
| Table 36: | RsslReactorChannelEventType Enumeration Values | 44 |
| Table 37: | RsslReactorChannelEvent Utility Functions | 45 |
| Table 38: | RsslMsgEvent Structure Members | 47 |
| Table 39: | RsslMsgEvent Utility Function | 47 |
| Table 40: | RsslRDMLoginMsgEvent Structure Members | 48 |
| Table 41: | RsslRDMLoginMsgEvent Utility Function | 48 |
| Table 42: | RsslRDMDirectoryMsgEvent Structure Members | 50 |
| Table 43: | RsslRDMDirectoryMsgEvent Utility Function | 50 |
| Table 44: | RsslRDMDictionaryMsgEvent Structure Members | 51 |
| Table 45: | RsslRDMDictionaryMsgEvent Utility Function | 52 |
| Table 46: | rsslReactorSubmitMsg Function | 53 |
| Table 47: | RsslReactorSubmitMsgOptions Structure Members | 53 |
| Table 48: | RsslReactorRequestMsgOptions Structure Members | 54 |
| Table 49: | RsslReactorSubmitMsgOptions Utility Function | 54 |
| Table 50: | rsslReactorSubmitMsg Return Codes | 54 |
| Table 51: | Reactor Buffer Management Functions | 57 |

| | | |
|------------|--|-----|
| Table 52: | rsslReactorGetBuffer Return Values | 58 |
| Table 53: | rsslReactorSubmit Function | 58 |
| Table 54: | RsslReactorSubmitOptions Structure Members | 59 |
| Table 55: | rsslReactorSubmit Return Codes | 59 |
| Table 56: | RsslReactorSubmitOptions Utility Function | 60 |
| Table 57: | rsslReactorPackBuffer Function | 61 |
| Table 58: | rsslReactorPackBuffer Return Values | 61 |
| Table 59: | RsslTunnelStreamAuthInfo Structure Members | 64 |
| Table 60: | rsslReactorOpenTunnelStream Method | 64 |
| Table 61: | RsslTunnelStreamOpenOptions | 65 |
| Table 62: | RsslClassOfService.common Structure Members | 66 |
| Table 63: | RsslClassOfService.authentication Structure Members | 67 |
| Table 64: | RsslClassOfService.flowControl Structure Members | 67 |
| Table 65: | RsslClassOfService.dataIntegrity Structure Members | 68 |
| Table 66: | RsslClassOfService.guarantee Structure Members | 68 |
| Table 67: | Tunnel Stream Callback Event Types | 69 |
| Table 68: | Tunnel Stream Callback Functions | 70 |
| Table 69: | Tunnel Stream Callback Event Types | 71 |
| Table 70: | RsslTunnelStreamRequestEvent Structure Members | 74 |
| Table 71: | rsslReactorAcceptTunnelStream Function | 74 |
| Table 72: | RsslReactorAcceptTunnelStreamOptions Options | 75 |
| Table 73: | rsslReactorRejectTunnelStream Function | 75 |
| Table 74: | RsslReactorRejectTunnelStreamOptions Options | 75 |
| Table 75: | Tunnel Stream Buffer Methods | 78 |
| Table 76: | Tunnel Stream Submit Method | 78 |
| Table 77: | RsslTunnelStreamSubmitOptions Structure Members | 79 |
| Table 78: | RsslTunnelStreamSubmitMsgOptions Structure Members | 79 |
| Table 79: | RsslTunnelStreamInfo Structure Members | 79 |
| Table 80: | rsslReactorCloseTunnelStream Method | 80 |
| Table 81: | RsslTunnelStreamCloseOptions Structure Members | 81 |
| Table 82: | rsslReactorQueryServiceDiscovery Method | 82 |
| Table 83: | RsslReactorServiceDiscoveryOptions Structure Members | 82 |
| Table 84: | RsslReactorDiscoveryTransportProtocol Enumerations | 83 |
| Table 85: | RsslReactorDiscoveryDataFormatProtocol Enumerations | 83 |
| Table 86: | RsslReactorServiceEndpointEvent Structure Members | 83 |
| Table 87: | RsslReactorServiceEndpointEvent Structure Members | 84 |
| Table 88: | RsslReactorOAuthCredential Structure Members | 84 |
| Table 89: | RsslReactorOAuthCredentialEvent Structure Members | 85 |
| Table 90: | RsslReactorOAuthCredentialRenewal Members | 85 |
| Table 91: | rsslReactorSubmitOAuthCredentialRenewal | 86 |
| Table 92: | rsslReactorSubmitOAuthCredentialRenewal Options | 86 |
| Table 93: | RsslReactorOAuthCredentialRenewalMode Enums | 86 |
| Table 94: | RsslReactorJsonConverterOptions Structure Members | 91 |
| Table 95: | RsslReactorServiceNameToldCallback Parameters | 92 |
| Table 96: | RsslReactorServiceNameToIdEvent Structure Members | 93 |
| Table 97: | RsslReactorJsonConversionEvent Structure Members | 93 |
| Table 98: | Reactor Utility Functions | 94 |
| Table 99: | RsslReactorChannelInfo Structure Members | 94 |
| Table 100: | Domains Representations in the Administration Domain Model Value Added Component | 103 |
| Table 101: | RsslRDMMsgBase Structure Members | 104 |
| Table 102: | RsslRDMMsg | 104 |
| Table 103: | Refinitiv Domain Model Encoding and Decoding Functions | 105 |
| Table 104: | RsslRDMLoginRequest Structure Members | 106 |
| Table 105: | RsslRDMLoginRequest Flags | 108 |
| Table 106: | RsslRDMLoginRequest Utility Functions | 109 |

| | | |
|------------|--|-----|
| Table 107: | RsslRDMLLoginRefresh Structure Members | 110 |
| Table 108: | RsslRDMLLoginRefresh Flags | 113 |
| Table 109: | RsslRDMLLoginRefresh Utility Functions..... | 115 |
| Table 110: | RsslRDMServerInfo Structure Members..... | 115 |
| Table 111: | RsslRDMServerInfo Flags | 115 |
| Table 112: | RsslRDMServerInfo Utility Functions..... | 115 |
| Table 113: | RsslRDMLLoginStatus Structure Members..... | 116 |
| Table 114: | RsslRDMLLoginStatus Flags | 117 |
| Table 115: | RsslRDMLLoginStatus Utility Functions..... | 118 |
| Table 116: | RsslRDMLLoginClose Structure Member..... | 118 |
| Table 117: | RsslRDMLLoginClose Utility Functions..... | 118 |
| Table 118: | RsslRDMLLoginConsumerConnectionStatus Structure Members | 119 |
| Table 119: | RsslRDMLLoginConsumerConnectionStatus Flags | 119 |
| Table 120: | RsslRDMLLoginWarmStandbyInfo Structure Members | 119 |
| Table 121: | RDMLLoginServerTypes Enumeration Values..... | 119 |
| Table 122: | RsslRDMLLoginConsumerConnectionStatus Utility Functions..... | 120 |
| Table 123: | Login Round Trip Time Members..... | 121 |
| Table 124: | Login Round Trip Time Flag Enumeration Values | 121 |
| Table 125: | Login Round Trip Time Utility Functions | 121 |
| Table 126: | RsslRDMLLoginMsg Union Members | 122 |
| Table 127: | RsslRDMLLoginMsg Utility Functions | 122 |
| Table 128: | Refinitiv Domain Model Login Encoding and Decoding Functions..... | 123 |
| Table 129: | RsslRDMDirectoryRequest Structure Members..... | 128 |
| Table 130: | RsslRDMDirectoryRequest Flags | 129 |
| Table 131: | RsslRDMDirectoryRequest Utility Functions..... | 129 |
| Table 132: | RsslRDMDirectoryRefresh Structure Members..... | 129 |
| Table 133: | RsslRDMDirectoryRefresh Flags | 130 |
| Table 134: | RsslRDMDirectoryRefresh Utility Functions..... | 130 |
| Table 135: | RsslRDMDirectoryUpdate Structure Members..... | 131 |
| Table 136: | RsslRDMDirectoryUpdate Flags..... | 131 |
| Table 137: | RsslRDMDirectoryUpdate Utility Functions | 132 |
| Table 138: | RsslRDMDirectoryStatus Structure Members..... | 132 |
| Table 139: | RsslRDMDirectoryStatus Flags..... | 133 |
| Table 140: | RsslRDMDirectoryStatus Utility Functions | 133 |
| Table 141: | RsslRDMDirectoryClose Structure Member..... | 133 |
| Table 142: | RsslRDMDirectoryClose Utility Functions | 133 |
| Table 143: | RsslRDMDirectoryConsumerStatus Structure Members | 134 |
| Table 144: | RsslRDMDirectoryConsumerStatusService Structure Members..... | 134 |
| Table 145: | RsslRDMDirectoryConsumerStatus Utility Functions..... | 134 |
| Table 146: | RsslRDMService Structure Members | 135 |
| Table 147: | RsslRDMService Flags | 135 |
| Table 148: | RsslRDMService Utility Function | 136 |
| Table 149: | RsslRDMServiceInfo Structure Members..... | 136 |
| Table 150: | RsslRDMServiceInfo Flags | 137 |
| Table 151: | RsslRDMServiceInfo Utility Functions..... | 138 |
| Table 152: | RsslRDMServiceState Structure Members | 138 |
| Table 153: | RsslRDMServiceState Flags | 139 |
| Table 154: | RsslRDMServiceState Utility Functions..... | 139 |
| Table 155: | RsslRDMServiceGroupState Structure Members | 139 |
| Table 156: | RsslRDMServiceGroupState Flags | 140 |
| Table 157: | RsslRDMServiceGroupState Utility Functions..... | 140 |
| Table 158: | RsslRDMServiceLoad Structure Members..... | 140 |
| Table 159: | RsslRDMServiceLoad Flags | 141 |
| Table 160: | RsslRDMServiceLoad Utility Functions..... | 141 |
| Table 161: | RsslRDMServiceData Structure Members..... | 141 |

| | | |
|------------|---|-----|
| Table 162: | RsslRDMServiceData Flags | 142 |
| Table 163: | RsslRDMServiceData Utility Functions..... | 142 |
| Table 164: | RsslRDMServiceLinkInfo Structure Members..... | 142 |
| Table 165: | RsslRDMServiceLinkInfo Utility Functions | 142 |
| Table 166: | RsslRDMServiceLink Structure Members..... | 143 |
| Table 167: | RsslRDMServiceLink Flags | 143 |
| Table 168: | RsslRDMServiceLink Utility Functions..... | 144 |
| Table 169: | RsslRDMServiceSeqMcastInfo Structure Members | 144 |
| Table 170: | Refinitiv Source Sink Library Refinitiv Domain Model Service Sequenced Multicast Info Enumeration Values 144 | |
| Table 171: | Refinitiv Source Sink Library Refinitiv Domain Model Address/Port Information Structure Members..... | 145 |
| Table 172: | RsslRDMServiceSeqMcastInfo Utility Functions | 145 |
| Table 173: | RsslRDMDirectoryMsg Union Members..... | 145 |
| Table 174: | RsslRDMDirectoryMsg Utility Functions..... | 145 |
| Table 175: | Refinitiv Domain Model Directory Encoding and Decoding Functions | 146 |
| Table 176: | RsslRDMDictionaryRequest Structure Members | 153 |
| Table 177: | RsslRDMDictionaryRequest Flag | 153 |
| Table 178: | RsslRDMDictionaryRequest Utility Functions..... | 154 |
| Table 179: | RsslRDMDictionaryRefresh Structure Members | 154 |
| Table 180: | RsslRDMDictionaryRefreshFlags | 155 |
| Table 181: | RsslRDMDictionaryRefresh Utility Functions..... | 156 |
| Table 182: | RsslRDMDictionaryStatus Structure Members..... | 156 |
| Table 183: | RsslRDMDictionaryStatus Flags | 156 |
| Table 184: | RsslRDMDictionaryStatus Utility Functions..... | 157 |
| Table 185: | RsslRDMDictionaryClose Structure Members..... | 158 |
| Table 186: | RsslRDMDictionaryClose Utility Functions | 158 |
| Table 187: | RsslRDMDictionaryMsg Union Members | 158 |
| Table 188: | RsslRDMDictionaryMsg Utility Functions | 158 |
| Table 189: | Refinitiv Domain Model Dictionary Encoding and Decoding Functions | 159 |
| Table 190: | RsslRDMQueueRequest Members | 164 |
| Table 191: | RsslRDMQueueRefresh Members | 165 |
| Table 192: | RsslRDMQueueStatus Members | 165 |
| Table 193: | RsslRDMQueueClose Members | 165 |
| Table 194: | RsslRDMQueueData Members | 166 |
| Table 195: | Queue Data Flag..... | 166 |
| Table 196: | RsslRDMQueueTimeoutCodes | 167 |
| Table 197: | Queue Data Message Encoding Methods | 167 |
| Table 198: | RsslRDMQueueDataExpired Structure Members..... | 169 |
| Table 199: | RsslRDMQueueDataUndeliverableCodes | 169 |
| Table 200: | RsslRDMQueueAck | 170 |
| Table 201: | Memory Buffer Functions | 171 |
| Table 202: | Payload Cache Management Functions | 174 |
| Table 203: | RsslCacheError Structure Members | 174 |
| Table 204: | Function for Cache Error Handling | 175 |
| Table 205: | Functions for Managing Cache Instances..... | 175 |
| Table 206: | RsslPayloadCacheConfigOptions Structure Members..... | 175 |
| Table 207: | Functions for Setting Dictionary to Cache..... | 176 |
| Table 208: | Payload Cache Utility Functions | 178 |
| Table 209: | Payload Cache Entry Management Functions | 179 |
| Table 210: | Functions for Applying and Retrieving Cache Entry Data | 181 |
| Table 211: | Functions for Using the Payload Cursor | 182 |

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 and interact with these other devices.

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 Value Added Components are written to both the C and Java 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 |
| Enterprise Message API | Enterprise Message API is part of the Refinitiv Real-Time SDK. |

Table 1: Acronyms and Abbreviations

| ACRONYM / TERM | MEANING |
|---|---|
| Enterprise Transport API | 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) |
| OMM | Open Message Model |
| QoS | Quality of Service |
| 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. |
| Refinitiv Wire Format | A Refinitiv proprietary 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](#)

1.6 Documentation Feedback

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

- Send us your comments via email at apidocumentation@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 apidocumentation@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:

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





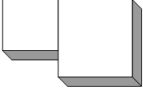




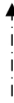
| | | | |
|---|---|---|--|
|  | Feed Handler, Refinitiv Real-Time server, or other application |  | Network of multiple servers |
|  | Enterprise Transport API application |  | Point-to-point connection showing direction of primary data flow |
|  | Application with local daemon |  | Point-to-point connection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

Figure 1. Network Diagram Notation




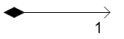
| | |
|---|---|
|  | Object |
|  | Inheritance: object on left is like object on right |
|  | Composition: object on left is made up of some number of objects on right |
|  | Composition: object on left is made up of one object on right |

Figure 2. UML Diagram Notation

2 Product Description and Overview

2.1 What is the Enterprise Transport API?

The Enterprise Transport API is a low-level Enterprise Transport API that provides the most flexible development environment to the application developer. It is the foundation on which all Refinitiv Open Message Model-based components are built. The Enterprise Transport API allows applications to achieve the highest throughput and lowest latency available with any Open Message Model 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 Open Message Model API offerings.

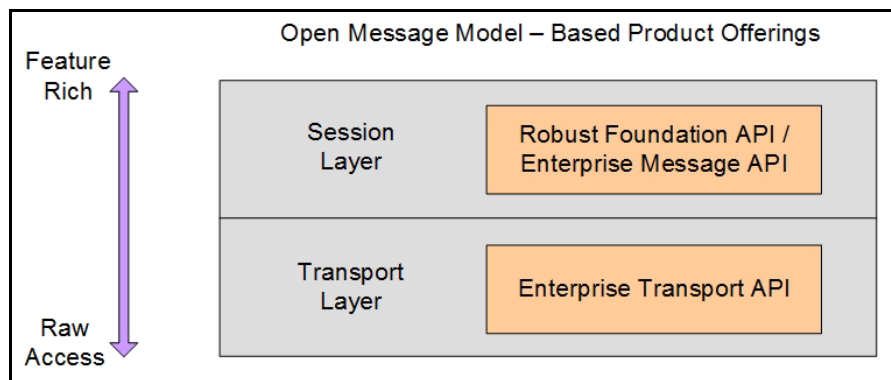


Figure 3. Open Message Model APIs with Value Added Components

The Enterprise Transport API Value Added Components provide alternate entry points for applications to leverage Open Message Model-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 Open Message Model 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.

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.

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.

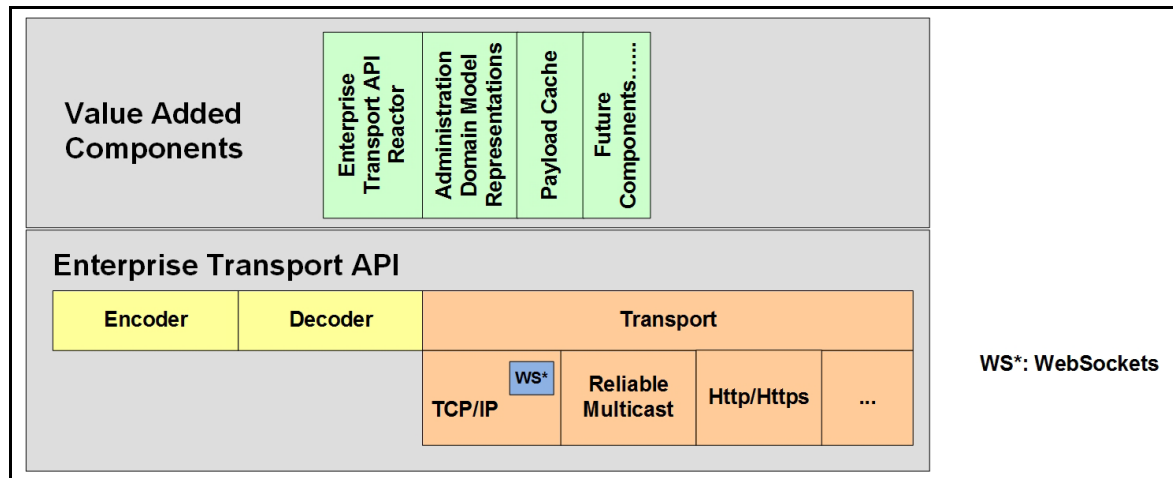


Figure 4. Enterprise Transport API Value Added Components

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 Open Message Model in its own functions and to connect to other Open Message Model-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 **rsslReactor.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 Open Message Model 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.

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 Refinitiv Domain Model-specific representations of the Open Message Model 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 Open Message Model devices (i.e., Refinitiv Real-Time Distribution System infrastructure), but also ensures that encoding/decoding for these domain models follow Open Message Model-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 Data Package.

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

2.6 Value Added Utilities

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

2.7 Value Added Cache

Applications can leverage the Open Message Model payload cache feature. Using the payload cache, an application can maintain a local store of the Open Message Model container data it consumes, publishes, or transforms. The cache maintains the latest values of the Open Message Model 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 Open Message Model 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 Open Message Model Consumer

3.1 Overview

This chapter provides an overview of how to create an Open Message Model Consumer application using the Enterprise Transport API Reactor and Administration Domain Model Representation Value Added Components. The Value Added Components simplify the work done by an Open Message Model consumer application when establishing a connection to other Open Message Model 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 Open Message Model 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 **rsslIVACConsumer** example application, included with the Enterprise Transport API product, provides one implementation of an Open Message Model consumer application that uses the Enterprise Transport API Value Added Components. The application is written with simplicity in mind and demonstrates usage of the Enterprise Transport API and Enterprise Transport API Value Added Components. Portions of functionality have been abstracted and can easily be reused, though you might need to modify it to achieve your own unique performance and functionality goals.

3.2 Leverage Existing or Create New RsslReactor

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

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

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

3.3 Implement Callbacks and Populate Role

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

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

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

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

- **RsslRDMLLoginMsgCallback**, which processes all data for the Refinitiv Domain Model Login domain.

- **RsslRDMDirectoryMsgCallback**, which processes all data for the Refinitiv Domain Model Source Directory domain.
- **RsslRDMDictionaryMsgCallback**, which processes all data for the Refinitiv Domain Model Dictionary domain.

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

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

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

3.4 Establish Connection using **rsslReactorConnect**

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

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

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

The **rsslReactorConnect** function is described in Section 6.4.1.1. Dispatching is described in Section 6.6.

3.5 Issue Requests and/or Post Information

After the **RsslReactorChannel** is established, the channel can be used to request additional content. When issuing the request, the consuming application can use the **serviceId** of the desired service, along with the stream's identifying information. Requests can be sent for any domain using the formats defined in that domain model specification. Domains provided by 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 **RsslDefaultMsgCallback**.

At this point, an Open Message Model 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 Enterprise Transport API Message Package and the Enterprise Transport API Data Package described in the Enterprise Transport API C Edition *Developers Guide*.

3.6 Log Out and Shut Down

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

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

- Closing an **RsslReactorChannel** is described in Section 6.4.2.
- Shutting down an **RsslReactor** is described in Section 6.2.2.

3.7 Additional Consumer Details

The following locations provide specific details about using Open Message Model consumers, the Enterprise Transport API, and Enterprise Transport API Value Added Components:

- The **rssIVAConsumer** application demonstrates one way of implementing of an Open Message Model consumer application that uses Enterprise Transport API 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 Enterprise Transport API Reactor.
- Chapter 8 provides more information about the Administration Domain Model Representations.
- The Enterprise Transport API C Edition *Developers Guide* provides specific Enterprise Transport API 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 Open Message Model Interactive Provider

4.1 Overview

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

The following steps summarize this process:

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

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

4.2 Leverage Existing or Create New RsslReactor

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

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

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

4.3 Create an RsslServer

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

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

4.4 Implement Callbacks and Populate Role

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

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

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

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

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

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

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

4.5 Associate Incoming Connections Using **rsslReactorAccept**

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

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

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

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

4.6 Perform Login Process

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

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

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

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

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

Information about the Login domain and expected content formatting is available in the Enterprise Transport API C Edition *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 Open Message Model 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 Enterprise Transport API Message Package and the Enterprise Transport API Data Package described in the Enterprise Transport API C Edition *Developers Guide*.

Information about the Source Directory domain and expected content formatting is available in the Enterprise Transport API C Edition *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 **RsslFieldList** 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 Enterprise Transport API 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 Enterprise Transport API 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 **RsslRDMDictionaryMsg**. More details and code examples are in Section 8.5. Dictionary requests will be provided via the **RsslRDMDictionaryMsgCallback**, when specified on the **RsslReactorOMMProviderRole**.

Whether loading a dictionary from file or requesting it from a Refinitiv Real-Time Advanced Data Hub, the Enterprise Transport API offers several utility functions for loading, downloading, and managing a properly-formatted field dictionary. The Enterprise Transport API 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 Enterprise Transport API Message Package and the Enterprise Transport API 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 **RsslStatusMsg** to indicate the reason and close the stream. All requests and responses should follow specific formatting as defined in the domain model specification. The Enterprise Transport API C Edition *Refinitiv Domain Model Usage Guide* defines all domains provided by Refinitiv. This content will be returned to the application via the **RsslDefaultMsgCallback**.

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

All content requested, received, or posted is encoded and decoded using the Enterprise Transport API Message Package and the Enterprise Transport API 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 Enterprise Transport API does not regulate rules for implementing the Round Trip Time message, the Enterprise Transport API provides several examples for applying this feature in provider applications. Generally, if the provider wants to support the RTT feature, the provider must provide methods for sending generic RTT messages to a consumer and extend callback methods for RTT calculation.

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

4.11 Disconnect Consumers and Shut Down

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

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

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

- Closing an **RsslReactorChannel** is described in Section 6.4.2.
- Shutting down an **RsslReactor** is described in Section 6.2.2.

4.12 Additional Interactive Provider Details

For specific details about Open Message Model interactive providers, the Enterprise Transport API, and Enterprise Transport API Value Added Component use, refer to the following locations:

- The **rsslVAPProvider** application demonstrates one implementation of an Open Message Model interactive provider application that uses Enterprise Transport API 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 Enterprise Transport API Reactor.
- Chapter 8 provides more information about the Administration Domain Model Representations.
- The Enterprise Transport API C Edition *Developers Guide* provides specific Enterprise Transport API 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 Open Message Model Non-Interactive Provider

5.1 Building an Open Message Model Non-Interactive Provider Overview

This chapter provides an overview of how to create an Open Message Model non-interactive provider application using the Enterprise Transport API Reactor and Administration Domain Model Representation Value Added Components. The Value Added Components simplify the work done by an Open Message Model 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 Open Message Model 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 Open Message Model consumer applications that indicate interest.

The general process can be summarized by the following steps.

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

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

5.2 Leverage Existing or Create New RsslReactor

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

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

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

5.3 Implement Callbacks and Populate Role

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

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

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

Additionally, an Open Message Model non-interactive provider can specify the administrative domain-specific callback function **RsslRDMLLoginMsgCallback**, which processes all data for the Refinitiv Domain Model Login domain.

The **RsslReactorOMNIPProviderRole** structure should be populated with all callback information for the **RsslReactorChannel**. **RsslReactorOMNIPProviderRole** 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 **RsslReactorOMNIPProviderRole** structure or with specific information according to the application and user. The **RsslReactor** will use this information when starting up the **RsslReactorChannel**.

- For detailed information on the **RsslReactorOMNIPProviderRole**, refer to Section 6.3.1.
- For information on the various callback functions and their specifications, refer to Section 6.6.2.

5.4 Establish Connection using **rsslReactorConnect**

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

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

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

- For further details on the **rsslReactorConnect** function, refer to Section 6.4.1.1.
- For further details on dispatching, refer to Section 6.6.

5.5 Perform Dictionary Download

If connected to a supporting Refinitiv Real-Time Advanced Data Hub, an Open Message Model non-interactive provider can download the RWFFld and RWFEnum dictionaries to retrieve the appropriate dictionary information for providing field list content. An Open Message Model 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 Enterprise Transport API 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 **RsslReactorChannel** is established, it can begin pushing content to the Refinitiv Real-Time Advanced Data Hub. Each unique information stream should begin with an **RsslRefreshMsg**, conveying all necessary identification information for the content. Because the provider instantiates this information, a negative value **streamId** should be used for all streams. The initial identifying refresh can be followed by other status or update messages.

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

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

5.7 Log Out and Shut Down

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

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

- For details on closing an **RsslReactorChannel**, refer to Section 6.4.2.
- Shutting down an **RsslReactor** is described in Section 6.2.2.

5.8 Additional Non-Interactive Provider Details

The following locations discuss specific details about using Open Message Model non-interactive providers and the Enterprise Transport API:

- The **rsslIVANIPProvider** application demonstrates one implementation of an Open Message Model non-interactive provider application that uses Enterprise Transport API 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 Enterprise Transport API Reactor.
- Chapter 8 provides more information about Administration Domain Model Representations.
- The Enterprise Transport API C Edition *Developers Guide* provides specific Enterprise Transport API 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 **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 Open Message Model. This component helps simplify many aspects of a typical Enterprise Transport API application, regardless of whether the application is an Open Message Model consumer, Open Message Model interactive provider, or Open Message Model non-interactive provider. The Enterprise Transport API 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 Enterprise Transport API and Enterprise Transport API Reactor functionalities, refer to Section 6.1.1.

The Enterprise Transport API Reactor internally depends on the Administration Domain Model Representation component. This allows the user to provide and consume the administrative Refinitiv Domain Model 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 Enterprise Transport API Reactor also leverages several utility components, contained in the Value Added Utilities. This includes constructs like mutex locks, a simple queue, and memory buffers.

The Enterprise Transport API 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 Enterprise Transport API 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.

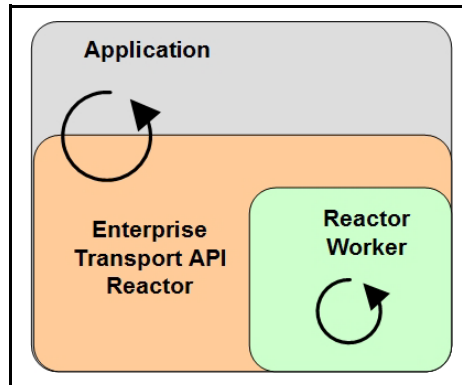


Figure 5. Enterprise Transport API Reactor Thread Model

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

Table 2: Enterprise Transport API Functionality and Enterprise Transport API Reactor Comparison

6.1.2 Reactor Error Handling

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

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

RsslErrorInfo members are as follows:

| STRUCTURE MEMBER | DESCRIPTION |
|--------------------------------|--|
| <code>rsslErrorInfoCode</code> | An informational code about this error. Indicates whether it reports a failure condition or is intended to provide non-failure-related information to the user. For details on available codes, refer to Table 20. |
| <code>rsslError</code> | Returns an rsslError structure (i.e., the underlying error information from the Enterprise Transport API). rsslError includes a pointer to the RsslChannel on which the error occurred, both a Enterprise Transport API and a system error number, and more descriptive error text. The rsslError and its values are described in the Enterprise Transport API C Edition <i>Developers Guide</i> . |
| <code>errorLocation</code> | Provides information about the file and line on which the error occurred. Detailed error text is provided via the rsslError portion of this structure. RsslErrorInfo.errorLocation length is limited to 1,024 bytes. |

Table 3: RsslErrorInfo Structure Members

6.1.3 Reactor Error Info Codes

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

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

Table 4: Reactor Error Info Codes

6.1.4 Enterprise Transport API Reactor Application Lifecycle

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

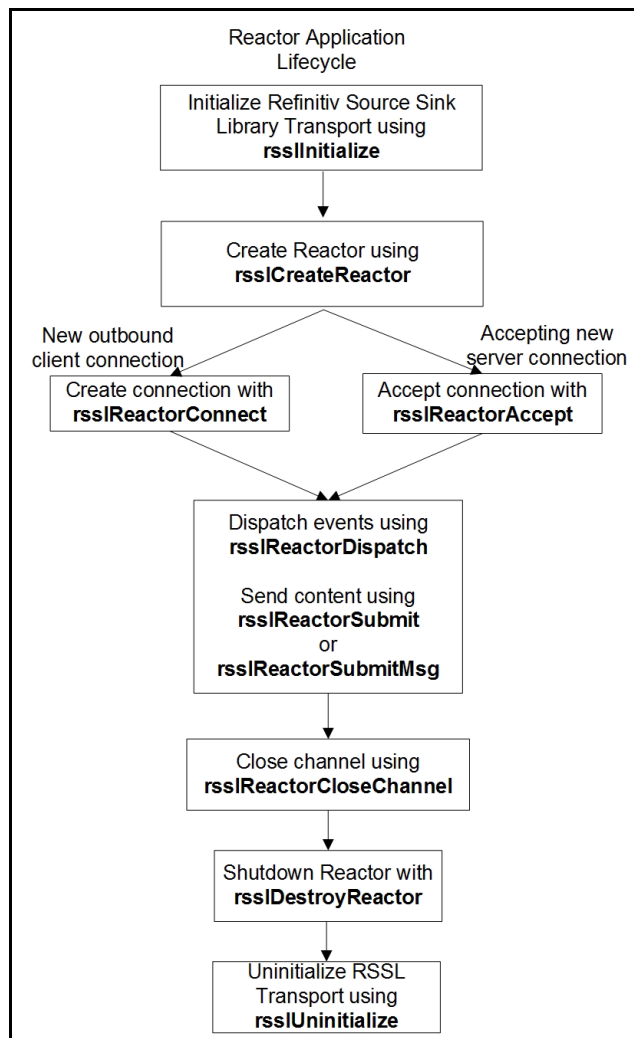


Figure 6. Enterprise Transport API Reactor Application Lifecycle

6.2 Reactor Use

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

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

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

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

Table 5: RsslReactor Structure Members

6.2.1 Creating a Reactor

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

6.2.1.1 Reactor Creation

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

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

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

Table 6: RsslReactor Creation Function

6.2.1.2 RsslCreateReactorOptions Structure Members

| STRUCTURE MEMBER | DESCRIPTION |
|--------------------------------|---|
| dispatchDecodeMemoryBufferSize | The size, in bytes, of an internally created memory buffer. The memory buffer is used by the RsslReactor when performing any necessary message decoding required for callbacks. When cleared, defaults to 65,536 bytes. |
| port | Deprecated. RsslReactor now chooses an ephemeral port upon creation. Any values specified in this parameter are ignored. |
| maxEventsInPool | Specifies the maximum number of event objects in the event's pool. |
| reissueTokenAttemptInterval | The time (in milliseconds) that the RsslReactor waits before attempting to reissue the token. The minimum interval is 1000 milliseconds, while the default setting is 5000. |
| reissueTokenAttemptLimit | The maximum number of times the RsslReactor attempts to reissue the token. If set to default (i.e., -1), there is no maximum limit. |
| restEnableLog | Enables Rest request/response logging. You can specify where to output the logs using the restLogOutputStream member. |
| 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 | Specifies the timeout (in seconds) for token service and service discovery request. If the request times out, the Enterprise Transport API Reactor resends the token reissue and the timeout restarts. When using the rsslReactorConnect() method, if the request times out, the Reactor does not retry. If set to 0 , there is no timeout. By default, the Enterprise Transport API behaves as if set to 90 seconds. |
| 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 Optimized solution/offering. |
| tokenReissueRatio | Specifies a ratio to multiply the access token's expiration time (in seconds) to determine the length of time the RsslReactor waits before retrieving a new access token and refreshing its connection to 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 . |
| tokenServiceURL | Specifies the URL of Refinitiv Data Platform that the RTSDK API uses to obtain an authentication token. |
| userSpecPtr | A pointer that can be set by the application. This value is preserved and stored in the userSpecPtr of the RsslReactor returned from rsslCreateReactor . This information can be useful for coupling this RsslChannel with other user-created information, such as a watch list associated with this connection. |

Table 7: RsslCreateReactorOptions Structure Members

6.2.1.3 RsslCreateReactorOptions Utility Function

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

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

Table 8: RsslCreateReactorOptions Utility Function

6.2.2 Destroying a Reactor

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

6.2.2.1 Reactor Destruction

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

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

Table 9: RsslReactor Destruction Function

6.2.2.2 Reactor Creation and Destruction Example

```
RsslCreateReactorOptions reactorCreateOptions;

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

rsslClearCreateReactorOptions(&reactorCreateOptions);

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

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

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

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

Code Example 1: Reactor Creation and Destruction Example

6.3 Reactor Channel Use

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

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

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

| STRUCTURE MEMBER | DESCRIPTION |
|------------------|--|
| hostname | Provides the name of the host to which a consumer or NIP application connects. |
| majorVersion | When an RsslReactorChannel is up (RSSL_RC_CET_CHANNEL_UP), this is populated with the major version number associated with the content sent on this connection. Typically only minor version increases are associated with a fully backward compatible change or extension. The Enterprise Transport API Reactor will leverage the versioning information for any content it is encoding or decoding. Proper use of versioning should be handled by the application for any other application encoded or decoded content. For more information on versioning, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> . |
| minorVersion | When an RsslReactorChannel is up (RSSL_RC_CET_CHANNEL_UP), this is populated with the minor version number associated with the content sent on this connection. Typically, a minor version increase is associated with a fully backward compatible change or extension. The Enterprise Transport API Reactor will leverage the versioning information for any content it is encoding or decoding. Proper use of versioning should be handled by the application for any other application encoded or decoded content. For more information on versioning, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> . |
| oldSocketId | It is possible for a file descriptor to change over time, typically due to some kind of connection keep-alive mechanism. If this occurs, this is typically communicated via a callback indicating RSSL_RC_CET_FD_CHANGE . The previous RsslReactorChannel is stored in oldSocketId so the application can properly unregister and then register the new socketId with their I/O notification mechanism. |
| port | Provides the server port number to which the consumer or NIP application connects. |
| protocolType | When an RsslReactorChannel is up (RSSL_RC_CET_CHANNEL_UP), this is populated with the protocolType associated with the content being sent on this connection. If the server indicates a protocolType that does not match the protocolType specified by the client, the connection is rejected. 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> . |
| pRsslChannel | A pointer to the underlying RsslChannel structure, as defined in the Enterprise Transport API C Edition <i>Developers Guide</i> , mainly for reference purposes. All operations should be performed using the Enterprise Transport API Reactor functionality; the application should not use this RsslChannel directly with any Refinitiv Source Sink Library Transport functionality. |
| pRsslServer | A pointer to the underlying RsslServer structure, as defined in the Enterprise Transport API C Edition <i>Developers Guide</i> , mainly for reference purposes. This is populated only if the channel was created via the rsslReactorAccept function, as described in Section 6.4.1.7. |

Table 10: RsslReactorChannel Structure Members

| STRUCTURE MEMBER | DESCRIPTION |
|------------------|--|
| socketId | Represents a file descriptor that can be used in some kind of I/O notification mechanism (e.g. select, poll) to alert users when dispatch is required on a specific RsslReactorChannel . This is the file descriptor associated with this end of the network connection; the file descriptor value may be different from the other end of the connection. |
| userSpecPtr | A pointer that can be set by the user of the RsslChannel . This value can be set directly or via the RsslReactorConnectOptions and RsslReactorAcceptOptions . This information can be useful for coupling this RsslReactorChannel with other user-created information, such as a watch list associated with this connection. |

Table 10: RsslReactorChannel Structure Members (Continued)

6.3.1 Reactor Channel Roles

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

- **RsslReactorOMMConsumerRole** for Open Message Model Consumer applications
- **RsslReactorOMMProviderRole** for Open Message Model Interactive Provider applications
- **RsslReactorOMNIProviderRole** for Open Message Model Non-Interactive Provider applications

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

6.3.1.1 RsslReactorChannelRoleBase Structure

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

| STRUCTURE MEMBER | DESCRIPTION |
|----------------------|--|
| channelEventCallback | This RsslReactorChannel 's user-defined callback function to handle all RsslReactorChannel specific events, like RSSL_RC_CET_CHANNEL_UP or RSSL_RC_CET_CHANNEL_DOWN . This callback function is required for all role types. This callback is defined in more detail in Section 6.6.2. |
| defaultMsgCallback | This RsslReactorChannel 's user-defined callback function to handle RsslMsg content not handled by another domain-specific callback function. This callback function is required for all role types and is defined in more detail in Section 6.6.2. |
| roleType | The role type enumeration value, as defined in Section 6.3.1.2. |

Table 11: RsslReactorChannelRoleBase Structure Members

6.3.1.2 roleType Enumerations

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

Table 12: RsslReactorChannelRoleBase.role Enumerated Values

6.3.2 Reactor Channel Role: Open Message Model Consumer

When an **RsslReactorChannel** is acting as an Open Message Model consumer application, it connects to an Open Message Model Interactive Provider. As part of this process it is expected to perform a login to the system. Once 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 Refinitiv Domain Models and are described in the Enterprise Transport API C Edition *Refinitiv Domain Model Usage Guide*.

An **RsslReactorChannel** in a consumer role helps to simplify this connection process by exchanging these messages on the user's behalf. The user can choose to provide specific information or leverage a default populated message, which uses the information of the user currently logged into the machine running the application. In addition, the Enterprise Transport API 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 Open Message Model Consumer Role

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

| STRUCTURE MEMBER | DESCRIPTION |
|------------------------|---|
| base | The role base structure, as defined in Section 6.3.1.1. |
| clientId | Deprecated. You must instead specify clientId in the RsslReactorOAuthCredential structure. For details, refer to Section 6.9.2.1. |
| dictionaryDownloadMode | Informs the RsslReactorChannel of the method to use when requesting dictionaries. Allowable modes are defined in Section 6.3.2.2. |
| dictionaryMsgCallback | This RsslReactorChannel 's user-defined callback function to handle dictionary message content. If not specified, all received dictionary messages will be passed to the defaultMsgCallback . <ul style="list-style-type: none"> For more details on this callback, refer to Section 6.6.2. Dictionary messages are described in Section 8.5. |
| directoryMsgCallback | This RsslReactorChannel 's user-defined callback function to handle directory message content. If not specified, all received directory messages will be passed to the defaultMsgCallback . <ul style="list-style-type: none"> For more details on this callback, refer to Section 6.6.2. Directory messages are described in Section 8.4. |
| loginMsgCallback | This RsslReactorChannel 's user-defined callback function to handle login message content. If not specified, all received login messages will be passed to the defaultMsgCallback . <ul style="list-style-type: none"> For more details on this callback, refer to Section 6.6.2. Login messages are described in Section 8.3. |
| pOAuthCredential | A pointer to the RsslReactorOAuthCredential structure which specifies the user's OAuth credentials. For details on the RsslReactorOAuthCredential structure, refer to Section 6.9.2.1. Use the RsslReactorOAuthCredential structure if the application must create and send the login message; in this case the application must also manage the login life cycle. |
| pDirectoryRequest | The RsslRDMDirectoryRequest (defined in Section 8.4.1) sent during the connection establishment process. This can be populated with specific source directory request information or invoke the rsslInitDefaultRDMDirectoryRequest function to populate with default information. <ul style="list-style-type: none"> If this parameter is specified, a pDirectoryRequest is required. If this parameter is empty, a directory request is not sent to the system. |

Table 13: RsslReactorOMMConsumerRole Structure Members

| STRUCTURE MEMBER | DESCRIPTION |
|------------------|--|
| pLoginRequest | The RsslRDMLLoginRequest (defined in Section 8.3.1) sent during the connection establishment process. This can be populated with a user's specific information or invoke the rsslInitDefaultRDMLLoginRequest function to populate with default information. If this parameter is empty, a login is not sent to the system; useful for systems that do not require a login. Use pLoginRequest when the application needs the Enterprise Transport API Reactor to send the initial login request. |
| watchlistOptions | Configurable options for the consumer watchlist. Options are described in more detail in Section 6.3.2.3. |

Table 13: RsslReactorOMMConsumerRole Structure Members (Continued)

6.3.2.2 Open Message Model Consumer Role Dictionary Download Modes

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

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

Table 14: RsslReactorOMMConsumerRole.dictionaryDownloadMode Enumerated Values

6.3.2.3 Open Message Model 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: Open Message Model Consumer Role Watchlist Options

6.3.2.4 Open Message Model Consumer Role Utility Method

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

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

Table 16: RsslReactorOMMConsumerRole Utility Function

6.3.3 Reactor Channel Role: Open Message Model Provider

When an **RsslReactorChannel** is acting as an Open Message Model provider application, it allows connections from Open Message Model 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 Refinitiv Domain Models and are described in the Enterprise Transport API C Edition *Refinitiv Domain Model Usage Guide*.

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

6.3.3.1 Open Message Model Provider Role Members

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

| STRUCTURE MEMBER | DESCRIPTION |
|---|--|
| <code>base</code> | The role base structure, as defined in Section 6.3.1.1. |
| <code>dictionaryMsgCallback</code> | This RsslReactorChannel 's user-defined callback function to handle dictionary message content. If unspecified, all received dictionary messages will be passed to the defaultMsgCallback . <ul style="list-style-type: none"> For further details on this callback, refer to Section 6.6.2. Dictionary messages are described in Section 8.5. |
| <code>directoryMsgCallback</code> | This RsslReactorChannel 's user-defined callback function to handle directory message content. If unspecified, all received directory messages will be passed to the defaultMsgCallback . <ul style="list-style-type: none"> For further details on this callback, refer to Section 6.6.2. Directory messages are described in Section 8.4. |
| <code>loginMsgCallback</code> | This RsslReactorChannel 's user-defined callback function to handle login message content. If unspecified, all received login messages are passed to the defaultMsgCallback . <ul style="list-style-type: none"> For further details on this callback, refer to Section 6.6.2. Login messages are described in Section 8.3. |
| <code>tunnelStreamListenerCallback</code> | This RsslReactorChannel 's user-defined callback for accepting or rejecting tunnel streams. For further details on this callback, refer to Section 6.8.6. |

Table 17: RsslReactorOMMProviderRole Structure Members

6.3.3.2 Open Message Model Provider Role Utility Function

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

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

Table 18: RsslReactorOMMProviderRole Utility Function

6.3.4 Reactor Channel Role: Open Message Model Non-Interactive Provider

When an **RsslReactorChannel** acts as an Open Message Model 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 Refinitiv Domain Models and are described in the Enterprise Transport API C Edition *Refinitiv Domain Model Usage Guide*.

An **RsslReactorChannel** in a non-interactive provider role helps to simplify this connection process by exchanging these messages on the user's behalf. The user can choose to provide specific information or leverage a default populated message, which uses the information of the user currently logged into the machine running the application. In addition, the Enterprise Transport API 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 Open Message Model Non-Interactive Role Members

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

| STRUCTURE MEMBER | DESCRIPTION |
|-------------------|--|
| base | The role base structure, as defined in Section 6.3.1.1. |
| pLoginRequest | The RsslRDMLLoginRequest , defined in Section 8.3.1, sent when establishing a connection. You can populate this with a user's specific information or invoke the rsslInitDefaultRDMLLoginRequest function to populate with a default set of information. If empty, a login is not sent to the system; useful for systems that do not require a login. |
| pDirectoryRefresh | The RsslRDMDirectoryRefresh , defined in Section 8.4.2, sent when establishing a connection. You can populate this with specific source directory refresh information. <ul style="list-style-type: none"> If this parameter is specified, a pDirectoryRefresh is required. If this parameter is left empty, a directory request is not sent to the system. |
| loginMsgCallback | The RsslReactorChannel 's user-defined callback function that handles login message content. If unspecified, all received login messages are passed to the defaultMsgCallback . For further details on this callback, refer to Section 6.6.2. |

Table 19: **RsslReactorOMMNIPProviderRole** Structure Members

6.3.4.2 Open Message Model Non-Interactive Provider Role Utility Function

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

| FUNCTION NAME | DESCRIPTION |
|-----------------------------|--|
| rsslClearOMMNIPProviderRole | Clears the RsslReactorOMMNIPProviderRole structure. Useful for structure reuse. |

Table 20: **RsslReactorOMMNIPProviderRole** Utility Function

6.3.5 Reactor Channel: Role Union

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

6.3.5.1 Union Members

| UNION MEMBER | DESCRIPTION |
|-------------------|--|
| base | The role's base structure, as defined in Table 11. |
| ommConsumerRole | The RsslReactorOMMConsumerRole , as defined in Section 6.3.2. |
| ommProviderRole | The RsslReactorOMMProviderRole , as defined in Section Section 6.3.3. |
| ommNIProviderRole | The RsslReactorOMMINIProviderRole , 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 **rsslReactorConnect** function will create a new **RsslReactorChannel** and associate it with an **RsslReactor**. This function creates a new outbound connection. The **RsslReactorChannel** is returned to the application via a callback, as described in Section 6.6.2, at which point it begins dispatching.

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

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

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

Table 23: rsslReactorConnect Function

6.4.1.2 RsslReactorConnectOptions Structure Members

| STRUCTURE MEMBER | DESCRIPTION |
|-----------------------|--|
| connectionCount | Specifies the number of connections listed in reactorConnectionList . If set to 0 , rsslConnectOptions is used. |
| connectionDebugFlags | Specifies a set of RsslDebugFlags for use when calling user-set debug callbacks as set by rsslSetDebugFunctions . If set to 0 , debug callbacks are not used |
| initializationTimeout | Specifies the amount of time (in seconds) to wait to successfully establish an RsslReactorChannel . If a RsslReactorChannel is not established in this timeframe, an event is dispatched to the application to indicate that the RsslReactorChannel is down. |
| reactorConnectionList | Specifies an array of connection information. When used with reconnectAttemptLimit , the RsslReactor attempts to connect to each host in the list with each reconnection attempt. |
| reconnectAttemptLimit | The maximum number of times the RsslReactor attempts to reconnect a channel when it fails. If set to -1 , there is no limit. |
| reconnectMinDelay | Specifies the minimum length of time the RsslReactor waits (in milliseconds) before attempting to reconnect a failed channel. The time increases with each reconnection attempt, from reconnectMinDelay to reconnectMaxDelay . |
| reconnectMaxDelay | Specifies the maximum length of time the RsslReactor waits (in milliseconds) before attempting to reconnect a failed channel. The time increases with each reconnection attempt, from reconnectMinDelay to reconnectMaxDelay . |
| rsslConnectOptions | Specifies information (rsslConnectOptions) about the host or network to which to connect, the type of connection to use, and other transport-specific configuration information associated with the underlying rsslConnect function. This is described in more detail in the Enterprise Transport API C Edition <i>Developers Guide</i> . |
| statisticFlags | Specifies RsslReactorChannelStatisticFlags which set the type of statistics reporting (if any) to perform on the RsslReactor channel. RsslReactorChannelStatisticFlags uses the following enums: <ul style="list-style-type: none"> • RSSL_RC_ST_NONE (or 0x0000): Turns off statistics reporting. • RSSL_RC_ST_READ (or 0x0001): Turns on statistics reporting for the number of bytes read and the number of uncompressed bytes read. • RSSL_RC_ST_WRITE (or 0x0002): Turns on statistics reporting for the number of bytes written and uncompressed bytes written. • RSSL_RC_ST_PING (or 0x0004): Turns on statistics reporting for the number of pings received and the number of pings sent. |

Table 24: RsslReactorConnectOptions Structure Members

6.4.1.3 RsslReactorConnectInfo Structure Members

| CLASS MEMBER | DESCRIPTION |
|-------------------------|---|
| rsslConnectOptions | Specifies information (rsslConnectOptions) about the host or network to which to connect, the type of connection to use, and other transport-specific configuration information associated with the underlying rsslConnect function. 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 true , the channel obtains the authentication token and refreshes it on behalf of user to keep session active. The default setting is false . |
| initializationTimeout | Specifies the amount of time (in seconds) to wait to successfully establish an RsslReactorChannel . If a RsslReactorChannel is not established in this timeframe, an event is dispatched to the application to indicate that the RsslReactorChannel is down. |
| location | Specifies the cloud location (e.g., us-east) of the service provider endpoint to which the RTSDK API establishes a connection. If location is not specified, the default setting is us-east . 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 RsslReactorAuthTokenEvents . The Reactor requests a token for the Consumer (i.e., disabling watchlist) and NiProvider applications to send login requests and reissues with the token. |

Table 25: RsslReactorConnectInfo Structure Members

6.4.1.4 RsslReactorConnectOptions Utility Function

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

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

Table 26: RsslReactorConnectOptions Utility Function

6.4.1.5 rsslReactorConnect Example

```
RsslReactorConnectOptions reactorConnectOpts;
RsslReactorOMMConsumerRole consumerRole;

RsslRDMLLoginRequest loginRequest;
RsslRDMDirectoryRequest directoryRequest;

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

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



```

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

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

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

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

```

Code Example 2: rsslReactorConnect Example

6.4.1.6 rsslReactorConnect Segmented Multicast Consumer Example

```

RsslReactorConnectOptions reactorConnectOpts;
RsslReactorOMMConsumerRole consumerRole;

RsslRDMLLoginRequest 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.segmented.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. */
rsslInitDefaultRDMLLoginRequest(&loginRequest, 1);
consumerRole.pLoginRequest = &loginRequest;

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

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

```

Code Example 3: rsslReactorConnect Segmented Multicast Consumer Example

6.4.1.7 Reactor Accept

The **rsslReactorAccept** function creates a new **RsslReactorChannel** and associates it with an **RsslReactor**. This function accepts the connection from an already running **RsslServer**. The **RsslReactorChannel** will be returned to the application via a callback, as described in Section 6.6.2, at which point it can begin dispatching on the channel.

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

Table 27: rsslReactorAccept Function

6.4.1.8 RsslReactorAcceptOptions Structure Members

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

Table 28: RsslReactorAcceptOptions Structure Members

6.4.1.9 RsslReactorAcceptOptions Utility Function

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

| FUNCTION NAME | DESCRIPTION |
|--|---|
| <code>rsslClearReactorAcceptOptions</code> | Clears the RsslReactorAcceptOptions structure. Useful for structure reuse. |

Table 29: RsslReactorAcceptOptions Utility Function

6.4.1.10 rsslReactorAccept Example

```
RsslReactorAcceptOptions reactorAcceptOpts;
RsslReactorOMMProviderRole providerRole;

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

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

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

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

Table 30: `rsslReactorCloseChannel` 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 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.

6.6 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 6.6.2.

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

Typically, an application registers both the **RsslReactor.eventFd** and each **RsslReactorChannel**'s `socketId` with an I/O notifier (e.g., `select`, `poll`). The I/O notifier can help inform the application when data is available on particular **RsslReactorChannels** or when channel information is available from the **RsslReactor**. An application can also forgo the use of notifiers and instead periodically call the dispatch function to process data as described in Section 6.6.1.

6.6.1 rsslReactorDispatch Function

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

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

| FUNCTION NAME | DESCRIPTION |
|---------------------|--|
| rsslReactorDispatch | This function processes events and messages across the provided RsslReactor and all of its associated RsslReactorChannels . When channel information or data is available for an RsslReactorChannel , the channel's user-defined callback function is invoked. The application can dispatch on a specified channel or over all channels associated with the RsslReactor . The application can also control the maximum number of messages dispatched with a single call to rsslReactorDispatch . This can be controlled through passed-in RsslReactorDispatchOptions , as described in Section 6.6.1.1. |

Table 31: rsslReactorDispatch Function

6.6.1.1 Reactor Dispatch Options

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

| STRUCTURE MEMBER | 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 . |
| maxMessages | Controls the maximum number of events or messages processed in this call. If this is larger than the number of available messages, rsslReactorDispatch will return when there is no more data to process. This value is initialized to allow up to 100 messages to be returned with a single call to rsslReactorDispatch . |

Table 32: RsslReactorDispatchOptions Structure Members

6.6.1.2 RsslReactorDispatchOptions Utility Function

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

| FUNCTION NAME | DESCRIPTION |
|---------------------------------|---|
| rsslClearReactorDispatchOptions | Clears the RsslReactorDispatchOptions structure. Useful for structure reuse. |

Table 33: RsslReactorDispatchOptions Utility Function

6.6.1.3 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

6.6.2 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., RsslRDMLLoginMsgCallback). This will cause the RsslReactor to invoke the RsslDefaultMsgCallback for this message upon the domain-specific callbacks return. |

Table 34: RsslReactorCallbackRet Callback Return Codes

6.6.3 Reactor Callback: Channel Event

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

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

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

6.6.3.1 Reactor Channel Event

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

| STRUCTURE MEMBER | DESCRIPTION |
|------------------|--|
| channelEventType | The type of event that has occurred on the RsslReactorChannel . For a list of enumeration values, refer to Section 6.6.3.2. |
| 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 35: RsslReactorChannelEvent Structure Members

6.6.3.2 Reactor Channel Event Type Enumeration Values

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

Table 36: RsslReactorChannelEventType Enumeration Values

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

Table 36: RsslReactorChannelEventType Enumeration Values (Continued)

6.6.3.3 Reactor Channel Event Utility Functions

| FUNCTION NAME | DESCRIPTION |
|------------------------------|---|
| rsslClearReactorChannelEvent | Clears an RsslReactorChannelEvent structure. |

Table 37: RsslReactorChannelEvent Utility Functions

6.6.3.4 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

6.6.4 Reactor Callback: Default Message

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

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

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

6.6.4.1 Reactor Message Event

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

| STRUCTURE MEMBER | DESCRIPTION |
|------------------|---|
| pRsslMsgBuffer | An RsslBuffer containing the raw, undecoded message that was read and processed by the callback. NOTE: When the consumer watchlist is enabled, an RsslBuffer is not provided, because the message might not match this buffer, or the message might be internally generated. |
| pRsslMsg | An RsslMsg structure populated with message content by calling 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 Refinitiv Domain Model 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 38: RsslMsgEvent Structure Members

6.6.4.2 Reactor Message Event Utility Functions

| FUNCTION NAME | DESCRIPTION |
|-------------------|--|
| rsslClearMsgEvent | Clears an RsslMsgEvent structure. |

Table 39: RsslMsgEvent Utility Function

6.6.4.3 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

6.6.5 Reactor Callback: Refinitiv Domain Model Login Message

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

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

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

6.6.5.1 Reactor Refinitiv Domain Model Login Message Event

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

| STRUCTURE MEMBER | DESCRIPTION |
|------------------|--|
| baseMsgEvent | An RsslMsgEvent populated with the raw buffer (RsslMsg) and any error information. This structure is defined in Section 6.6.4.1. |
| pRDMLLoginMsg | The Refinitiv Domain Model representation of the decoded Login message. If not present, an error was encountered while processing the information. This message is presented as the RsslRDMLLoginMsg , described in Section 8.3. |

Table 40: RsslRDMLLoginMsgEvent Structure Members

6.6.5.2 Reactor Refinitiv Domain Model Login Message Event Utility Function

| FUNCTION NAME | DESCRIPTION |
|----------------------------|---|
| rsslClearRDMLLoginMsgEvent | Clears an RsslRDMLLoginMsgEvent structure. |

Table 41: RsslRDMLLoginMsgEvent Utility Function

6.6.5.3 Reactor Refinitiv Domain Model Login Message Event Callback Example

```

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

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

```

Code Example 9: Reactor Refinitiv Domain Model Login Message Event Callback Example

6.6.6 Reactor Callback: Refinitiv Domain Model Directory Message

The **RsslReactor** Refinitiv Domain Model Directory Message callback is used to communicate all received Refinitiv Domain Model 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.

6.6.6.1 Reactor Refinitiv Domain Model 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 6.6.4.1. |
| pRDMDirectoryMsg | The Refinitiv Domain Model 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 42: RsslRDMDirectoryMsgEvent Structure Members

6.6.6.2 Reactor Refinitiv Domain Model Directory Message Event Utility Function

| FUNCTION NAME | DESCRIPTION |
|-------------------------------|--|
| rsslClearRDMDirectoryMsgEvent | Clears an RsslRDMDirectoryMsgEvent structure. |

Table 43: RsslRDMDirectoryMsgEvent Utility Function

6.6.6.3 Reactor Refinitiv Domain Model Directory Message Event Callback Example

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

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

```

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

```

Code Example 10: Reactor Refinitiv Domain Model Directory Message Event Callback Example

6.6.7 Reactor Callback: Refinitiv Domain Model Dictionary Message

The **RsslReactor** Refinitiv Domain Model Dictionary Message callback is used to communicate all received Refinitiv Domain Model 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.

6.6.7.1 Reactor Refinitiv Domain Model 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 6.6.4.1. |
| pRDMDictionaryMsg | The Refinitiv Domain Model 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 44: RsslRDMDictionaryMsgEvent Structure Members

6.6.7.2 Reactor Refinitiv Domain Model Dictionary Message Event Utility Function

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

Table 45: RsslRDMDictionaryMsgEvent Utility Function

6.6.7.3 Reactor Refinitiv Domain Model 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 Refinitiv Domain Model Dictionary Message Event Callback Example

6.7 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.

6.7.1 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).

6.7.1.1 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 46: rsslReactorSubmitMsg Function

6.7.1.2 Reactor Submit Message Options

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

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

Table 47: 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 47: **RsslReactorSubmitMsgOptions** Structure Members (Continued)

6.7.1.3 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 48: **RsslReactorRequestMsgOptions** Structure Members

6.7.1.4 RsslReactorSubmitMsgOptions Utility Function

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

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

Table 49: **RsslReactorSubmitMsgOptions** Utility Function

6.7.1.5 rsslReactorSubmitMsg Return Codes

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

| RETURN CODE | DESCRIPTION |
|----------------------------|--|
| RSSL_RET_SUCCESS | Indicates that the rsslReactorSubmitMsg function has succeeded. |
| RSSL_RET_BUFFER_NO_BUFFERS | <p>Indicates that not enough pool buffers are available to write the message.</p> <p>The application can try to submit the message later, or it can use rsslReactorChannelIoctl to increase the number of available pool buffers and try again.</p> |
| 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 50: **rsslReactorSubmitMsg** Return Codes

6.7.1.6 rsslReactorSubmitMsg Example

The following example shows typical use of `rsslReactorSubmitMsg`.

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

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

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

Code Example 12: rsslReactorSubmitMsg Example

6.7.2 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.

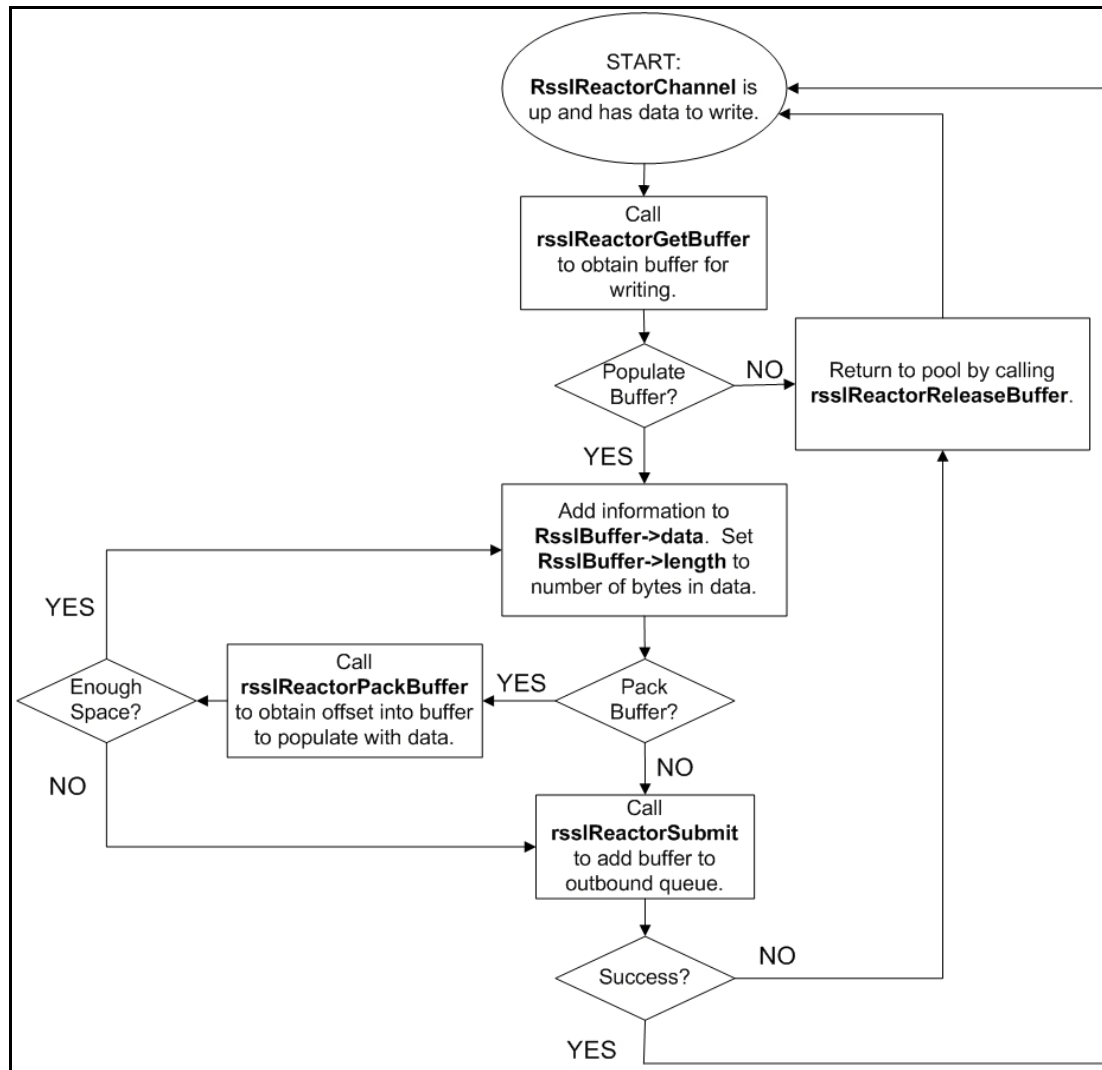


Figure 7. Flow Chart for writing data via `rsslReactorSubmit`

6.7.2.1 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.

6.7.2.2 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 6.7.2.4).</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 6.7.2.8.</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 51: Reactor Buffer Management Functions

6.7.2.3 Obtaining a Buffer: `rsslReactorGetBuffer` Return Values

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

| RETURN CODE | DESCRIPTION |
|--|---|
| Valid buffer returned Success Case | An RsslBuffer is returned to the user. The RsslBuffer.length indicates the number of bytes available to populate and the RsslBuffer.data provides a starting location for population. |
| NULL buffer returned Error Code: <code>RSSL_RET_BUFFER_NO_BUFFERS</code> | NULL is returned to the user. This value indicates that there are no buffers available to the user. See RsslErrorInfo content for more details. This typically occurs because all available buffers are queued and pending flushing to the connection. The <code>rsslReactorChannelIoctl</code> function can be used to increase the number of guaranteedOutputBuffers (for details, refer to Section 6.11). |
| NULL buffer returned Error Code: <code>RSSL_RET_FAILURE</code> | NULL is returned to the user. This value indicates that some type of general failure has occurred. The RsslReactorChannel should be closed. |
| NULL buffer returned Error Code: <code>RSSL_RET_INIT_NOT_INITIALIZED</code> | Indicates that the underlying RSSL Transport has not been initialized. See the RsslErrorInfo content for more details. |

Table 52: `rsslReactorGetBuffer` Return Values

6.7.2.4 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 6.7.2.5 describes the `rsslReactorSubmit` function and its associated parameters.

6.7.2.5 Writing Data: `rsslReactorSubmit` Function

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

| FUNCTION NAME | DESCRIPTION |
|--------------------------------|--|
| <code>rsslReactorSubmit</code> | Writes data. This function expects the buffer to be properly populated, where length reflects the actual number of bytes used. This function calls the Enterprise Transport API rsslWrite function and also triggers the rsslFlush function (described in the Enterprise Transport API C Edition <i>Developers Guide</i>). This function allows for several modifications and additional parameters to be specified via the RsslReactorSubmitOptions structure, defined in Section 6.7.2.6. For a list of return codes, refer to Section 6.7.2.7. |

Table 53: `rsslReactorSubmit` Function

6.7.2.6 Writing Data: Reactor Submit Options

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

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

Table 54: RsslReactorSubmitOptions Structure Members

6.7.2.7 Writing Data: rsslReactorSubmit Return Codes

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

| RETURN CODE | DESCRIPTION |
|---------------------------|---|
| RSSL_RET_SUCCESS | Indicates that the rsslReactorSubmit function has succeeded. The RsslBuffer will be released by the Enterprise Transport API Reactor. |
| RSSL_RET_WRITE_CALL_AGAIN | Indicates that a large buffer could not be fully written with this rsslReactorSubmit 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 rsslReactorChannelIoctl to increase the number of available pool buffers. After pool buffers become available again, the same buffer should be used to call rsslReactorSubmit 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. If the application does not subsequently pass the buffer to rsslReactorSubmit , the application should release it by calling rsslReactorReleaseBuffer . |
| RSSL_RET_FAILURE | Indicates that a general write failure has occurred. The RsslReactorChannel should be closed. The application should release the RsslBuffer by calling rsslReactorReleaseBuffer . |

Table 55: rsslReactorSubmit Return Codes

6.7.2.8 Writing Data: RsslReactorSubmitOptions Utility Function

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

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

Table 56: RsslReactorSubmitOptions Utility Function

6.7.2.9 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

6.7.2.10 Packing Additional Data into a Buffer

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

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

| FUNCTION NAME | DESCRIPTION |
|------------------------------------|--|
| <code>rsslReactorPackBuffer</code> | <p>Packs the contents of a passed-in RsslBuffer and returns a new RsslBuffer to continue packing new data into. For a buffer to allow packing, it must be requested from rsslReactorGetBuffer as 'packable' and cannot exceed the maxFragmentSize. The returned buffer provides a data pointer for populating and the length conveys number of bytes available in the buffer.</p> <p>An application can use the RsslBuffer.length to determine the amount of space available to continue packing buffers into. After each buffer is populated, the length should be set to reflect the actual number of bytes contained in the buffer. This will ensure that only the necessary space is reserved while packing.</p> <p>rsslReactorPackBuffer return values are defined in Section 6.7.2.11.</p> <p>This function calls the rsslPackBuffer function as described in the Enterprise Transport API C Edition <i>Developers Guide</i>.</p> |

Table 57: **rsslReactorPackBuffer** Function

6.7.2.11 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 Refinitiv Source Sink Library Transport has not been initialized. See the RsslErrorInfo content for more details. |

Table 58: **rsslReactorPackBuffer** Return Values

6.7.2.12 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`

6.8 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 end point. The provider end-point acknowledges creation of the stream as well as the behaviors that it will enforce on the stream. After the stream is established, the consumer can exchange any content it wants, though the tunnel stream will enforce behaviors on the transmitted content as negotiated with the provider.

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

In the following diagram, the orange cylinder represents a tunnel stream that connects the consumer application to the provider application. Notice that the tunnel stream passes directly through intermediate components: the tunnel stream has end-to-end line of sight so that the provider and consumer effectively talk to one another directly, though they traverse multiple devices in the system. Each black line flowing through the cylinder represents a different substream, where each substream transmits its own independent stream of information. Each substream could communicate different market content; for example one could be a Time Series request while another could be a request for Market Price content. A substream can also connect to a special provider application called a Queue Provider. A Queue Provider allows for persistence of content exchanged over the tunnel stream and substream, and helps provide content beyond the end-point visible to the consumer. To interact with a Queue Provider, additional addressing information is required, described in more detail in Section 8.6.

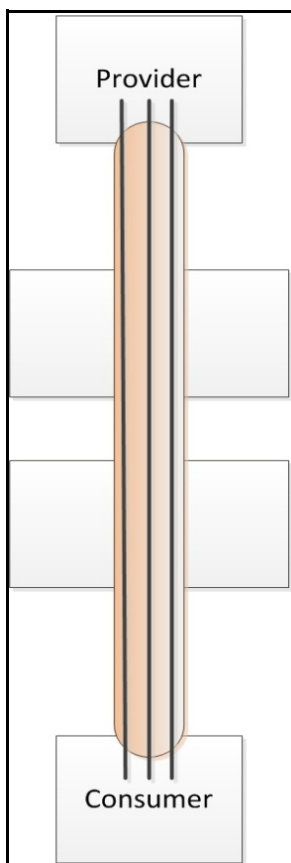


Figure 8. Tunnel Stream Illustration

6.8.1 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 6.8.3.

The **RsslClassOfService.authentication.type** may be set to **RDM_COS_AU_OMM_LOGIN**. When an Open Message Model consumer expects this type of authentication, it should set an **RsslRDMLoginRequest** message on the **RsslTunnelStreamOpenOptions.pAuthLoginRequest** member. If the Open Message Model 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. Open Message Model 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 59: **RsslTunnelStreamAuthInfo** Structure Members

6.8.2 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 6.8.4.

6.8.2.1 rsslReactorOpenTunnelStream Method

| METHOD NAME | DESCRIPTION |
|-----------------------------|--|
| rsslReactorOpenTunnelStream | Begins the establishment of a tunnel stream. The RsslTunnelStream is returned via the RsslTunnelStreamStatusEventCallback as specified on the RsslTunnelStreamOpenOptions . For more details, refer to Section 6.8.2.2. |

Table 60: **rsslReactorOpenTunnelStream** Method

6.8.2.2 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 6.8.4. |
| 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 6.8.4. 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 6.8.4. |
| name | Specifies the tunnel stream name, which is provided to the remote application. name cannot be longer than 255 characters. |
| responseTimeout | Sets the duration (in seconds) to wait for a provider to respond to a tunnel stream open request. If the provider does not respond in time, an RsslTunnelStreamStatusEvent is sent to the application to indicate that the tunnel stream was not opened. |
| guaranteedOutputBuffers | Sets the number of guaranteed output buffers available for the tunnel stream. |
| pAuthLoginRequest | Specifies the RsslRDMLLoginRequest to send if RsslClassOfService.authentication.type is set to RDM_COS_AU_OMM_LOGIN . If absent, the API uses the login request provided on the RsslReactorOMMConsumerRole.pLoginRequest . |
| classOfService | The class of service of the tunnel stream to be opened. For further details on RsslClassOfService , refer to Section 6.8.3. |

Table 61: RsslTunnelStreamOpenOptions

6.8.3 Negotiating Stream Behaviors: Class of Service

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

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

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

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

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

6.8.3.1 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 62: RsslClassOfService.common Structure Members

6.8.3.2 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 6.8.1. |

Table 63: RsslClassOfService.authentication Structure Members

6.8.3.3 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 64: RsslClassOfService.flowControl Structure Members

6.8.3.4 ClassOfService Data Integrity Member

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

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

Table 65: `RsslClassOfService.dataIntegrity` Structure Members

6.8.3.5 ClassOfService Guarantee Members

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

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

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

Table 66: `RsslClassOfService.guarantee` Structure Members

6.8.4 Tunnel Stream Callback Functions and Event Types

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

| EVENT | EVENT DESCRIPTION | CLASS MEMBER | CLASS MEMBER DESCRIPTION |
|-------------------------------|--|-----------------|--|
| RsslTunnelStreamStatusEvent | This event presents the tunnel stream and its status. | pReactorChannel | A pointer to the RsslReactorChannelTunnelStream with which this tunnel stream is associated. |
| | | pState | Indicates status information associated with the RsslTunnelStream . For example: <ul style="list-style-type: none"> A state of OPEN and OK indicates that the tunnel stream is established and content should be flowing as expected. A state of CLOSED_RECOVER or SUSPECT indicates that the connection or tunnel stream might be lost. However, if performing guaranteed messaging, content might be persisted by the reactor and communicated upon recovery of the tunnel stream. |
| | | pRsslMsg | A pointer to an RsslMsg structure. |
| | | pAuthInfo | If the event was produced by an authentication message, pAuthInfo is populated by an RsslTunnelStreamAuthInfo structure. For more information, refer to Section 6.8.1. |
| RsslTunnelStreamMsgEvent | This event presents content received on the RsslTunnelStream . If a more specific handler (i.e., RsslTunnelStreamQueueMsgCallback) is also configured, messages of that type will go to their specific handler. | pReactorChannel | A pointer to the RsslReactorChannelTunnelStream with which this tunnel stream is associated. |
| | | pRsslMsg | A pointer to an RsslMsg structure, used to deliver any Open Message Model 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 6.8.4.2. |
| | | pQueueMsg | A pointer to a Queue Message containing Open Message Model content or opaque content exchanged with a Queue Provider. Refer to subsequent chapters for information about Queue Messages. |

Table 67: Tunnel Stream Callback Event Types

6.8.4.1 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 6.8.4.2. 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 6.8.4.2. |
| 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 6.8.4.2. |
| 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 6.8.4.2. |

Table 68: Tunnel Stream Callback Functions

6.8.4.2 Tunnel Stream Callback Event Types

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

| EVENT | EVENT DESCRIPTION | CLASS MEMBER | CLASS MEMBER DESCRIPTION |
|-------------------------------|--|-----------------|--|
| RsslTunnelStreamStatusEvent | This event presents the tunnel stream and its status. | pReactorChannel | A pointer to the RsslReactorChannelTunnelStream with which this tunnel stream is associated. |
| | | pState | Indicates status information associated with the RsslTunnelStream . For example: <ul style="list-style-type: none"> A state of OPEN and OK indicates that the tunnel stream is established and content should be flowing as expected. A state of CLOSED_RECOVER or SUSPECT indicates that the connection or tunnel stream might be lost. However, if performing guaranteed messaging, content might be persisted by the reactor and communicated upon recovery of the tunnel stream. |
| | | pRsslMsg | A pointer to an RsslMsg structure. |
| | | pAuthInfo | If the event was produced by an authentication message, pAuthInfo is populated by an RsslTunnelStreamAuthInfo structure. For more information, refer to Section 6.8.1. |
| RsslTunnelStreamMsgEvent | This event presents content received on the RsslTunnelStream . If a more specific handler (i.e., RsslTunnelStreamQueueMsgCallback) is also configured, messages of that type will go to their specific handler. | pReactorChannel | A pointer to the RsslReactorChannelTunnelStream with which this tunnel stream is associated. |
| | | pRsslMsg | A pointer to an RsslMsg structure, used to deliver any Open Message Model 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 6.8.4.2. |
| | | pQueueMsg | A pointer to a Queue Message containing Open Message Model content or opaque content exchanged with a Queue Provider. Refer to subsequent chapters for information about Queue Messages. |

Table 69: Tunnel Stream Callback Event Types

6.8.5 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

6.8.6 Accepting Tunnel Streams

Open Message Model 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 6.8.6.1.

- 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.

6.8.6.1 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 70: RsslTunnelStreamRequestEvent Structure Members

6.8.6.2 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 6.8.6.3. |

Table 71: rsslReactorAcceptTunnelStream Function

6.8.6.3 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 6.8.4.1. |
| defaultMsgCallback | Specifies the instance of the callback used to handle all other content received on this RsslTunnelStream . For details on RsslTunnelStreamDefaultMsgCallback , refer to Section 6.8.4.1. |
| 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 6.8.3. |
| guaranteedOutputBuffers | Sets the number of pooled buffers available to the application when writing content to RsslTunnelStream . |

Table 72: RsslReactorAcceptTunnelStreamOptions Options

6.8.6.4 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 6.8.6.5. |

Table 73: rsslReactorRejectTunnelStream Function

6.8.6.5 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 74: RsslReactorRejectTunnelStreamOptions Options

6.8.6.6 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

6.8.6.7 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

6.8.7 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 6.8.4.
- Tunnel stream callback methods deliver the events described in Section 6.8.4.2.

6.8.8 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.

6.8.8.1 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 6.8.8.5. |
| <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 75: Tunnel Stream Buffer Methods

6.8.8.2 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 Refinitiv Domain Model 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 Refinitiv Domain Model 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 76: Tunnel Stream Submit Method

6.8.8.3 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.</p> <p>For example:</p> <ul style="list-style-type: none"> If the submitted buffer contains an <code>RsslMsg</code>, set <code>containerType</code> <code>RSSL_DT_MSG</code>. If sending non-Refinitiv Wire Format data, set <code>containerType</code> to a non-Refinitiv Wire Format 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 77: RsslTunnelStreamSubmitOptions Structure Members

6.8.8.4 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 78: RsslTunnelStreamSubmitMsgOptions Structure Members

6.8.8.5 RsslTunnelStreamInfo Structure Member

The following table describes values available when using the `rsslTunnelStreamGetInfo` method (for details, refer to Section 6.8.8.1). 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 79: RsslTunnelStreamInfo Structure Members

6.8.8.6 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

6.8.8.7 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 80: `rsslReactorCloseTunnelStream` Method

6.8.8.8 RsslTunnelStreamCloseOptions

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

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

Table 81: `RsslTunnelStreamCloseOptions` Structure Members

6.8.8.9 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

6.9 Cloud Connectivity

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

6.9.1 `rsslReactorQueryServiceDiscovery`

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

6.9.1.1 `rsslReactorQueryServiceDiscovery` Method

| METHOD | DESCRIPTION |
|---|---|
| <code>rsslReactorQueryServiceDiscovery</code> | <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 6.9.1.2).</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 7.3.5) 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 82: `rsslReactorQueryServiceDiscovery` Method

6.9.1.2 `RsslReactorServiceDiscoveryOptions`

| MEMBER | DESCRIPTION |
|--|--|
| <code>clientId</code> | Required. An <code>RsslBuffer</code> that specifies a unique ID defined for an application making a request to the token service. |
| <code>clientSecret</code> | An <code>RsslBuffer</code> that specifies the client secret (if one exists) used by the OAuth client to authenticate to the authorization Server. |
| <code>dataFormat</code> | 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 6.9.1.4. |
| <code>password</code> | Required. An <code>RsslBuffer</code> that specifies a password for authorization with the token service. |
| <code>proxyHostName</code> | Optional. An <code>RsslBuffer</code> that specifies a proxy server hostname. |
| <code>proxyPort</code> | Optional. An <code>RsslBuffer</code> that specifies a proxy server port. |
| <code>proxyUserName</code> | Optional. An <code>RsslBuffer</code> that specifies a username to perform authorization with a proxy server. |
| <code>proxyPasswd</code> | Optional. An <code>RsslBuffer</code> that specifies a password to perform authorization with a proxy server. |
| <code>proxyDomain</code> | Optional. An <code>RsslBuffer</code> that specifies the proxy domain of the user to authenticate. Required for NTLM or for Negotiate/Kerberos or for Kerberos authentication protocols. |
| <code>pServiceEndpointEventCallback</code> | A callback function that receives <code>RsslReactorServiceEndpointEvents</code> . Applications can take service endpoint information from the callback to get an endpoint and establish a connection to the service. |
| <code>tokenScope</code> | An <code>RsslBuffer</code> that specifies an optional token scope to limit the scope of the generated token from the token service. |

Table 83: `RsslReactorServiceDiscoveryOptions` Structure Members

| MEMBER | DESCRIPTION |
|-------------|--|
| transport | Optional. An enumeration that specifies the desired transport protocol to retrieve service endpoints from the service discovery. For available values, refer to Section 6.9.1.3. |
| 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 . |

Table 83: RsslReactorServiceDiscoveryOptions Structure Members (Continued)

6.9.1.3 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 84: RsslReactorDiscoveryTransportProtocol Enumerations

6.9.1.4 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 Refinitiv Wire Format data format. |
| RSSL_RD_DP_JSON2 = 2 | Specifies that the service discovery should use the tr_json2 data format |

Table 85: RsslReactorDiscoveryDataFormatProtocol Enumerations

6.9.1.5 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. |
| userSpecPtr | Optional. A user-specified pointer associated with this RsslReactorServiceEndpointEvent . |

Table 86: RsslReactorServiceEndpointEvent Structure Members

6.9.1.6 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 87: RsslReactorServiceEndpointEvent Structure Members

6.9.2 OAuth Credential Management

6.9.2.1 RsslReactorOAuthCredential Structure

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

RsslReactorOAuthCredential includes the following members:

| MEMBER | DESCRIPTION |
|-------------------------------|--|
| clientId | Required. An RsslBuffer that specifies a unique ID defined for the application that makes the request. For further details on the Client ID, refer to Section 7.3.1. |
| clientSecret | An RsslBuffer that specifies a the Client ID 'secret' that OAuth clients can use to authenticate. 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/ . |
| password | Required. A RsslBuffer that specifies the password used in tandem with the userName to obtain the access token. |
| 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 7.3.2. |
| takeExclusiveSignOnControl | Specifies the exclusive sign-on control that forces other applications to sign-out using the same credentials. |

Table 88: RsslReactorOAuthCredential Structure Members

| MEMBER | DESCRIPTION |
|------------|---|
| 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 Authorization - All about tokens in the Developer Community Portal. By default, the Enterprise Transport API uses the scope: trapi.streaming.pricing.read . |
| userName | Required. An RsslBuffer that specifies the user name used to obtain the access token from the Refinitiv Data Platform. |

Table 88: **RsslReactorOAuthCredential** Structure Members (Continued)

6.9.2.2 RsslReactorOAuthCredentialEvent

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 89: **RsslReactorOAuthCredentialEvent** Structure Members

6.9.2.3 RsslReactorOAuthCredentialRenewal

| MEMBER | DESCRIPTION |
|--------------|---|
| 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 except when specifying sensitive information in the RsslReactorOAuthCredentialEventCallback . |
| password | Required. An RsslBuffer that specifies the password, which is sent with the userName to get an access token and a refresh token. |
| 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 . |
| clientId | An RsslBuffer that specifies the unique Client ID for the application that makes the request. |
| clientSecret | An RsslBuffer that specifies the client secret (if one exists) used by the OAuth client to authenticate to the authorization Server. |
| tokenScope | An RsslBuffer that specifies the scope of the generated token. |

Table 90: **RsslReactorOAuthCredentialRenewal** Members

6.9.2.4 rsslReactorSubmitOAuthCredentialRenewal Method

| MEMBERS | DESCRIPTION |
|---|--|
| rsslReactorSubmitOAuthCredentialRenewal | <p>Uses the passed-in RsslReactor and RsslReactorOAuthCredentialRenewal to submit the application's password (and client secret if available) to the Refinitiv Data Platform token service. An application can also use this method to change its password.</p> <p>For a list of options you can use with rsslReactorSubmitOAuthCredentialRenewal, refer to Section 6.9.2.5.</p> <p>If you call this method outside of the RsslReactorOAuthCredentialEventCallback, you should also include pAuthTokenEventCallback to receive a result response.</p> <p>Error handling is managed by the RsslErrorInfo structure.</p> |

Table 91: rsslReactorSubmitOAuthCredentialRenewal

6.9.2.5 rsslReactorSubmitOAuthCredentialRenewal Options

| OPTION | DESCRIPTION |
|-------------------------|---|
| pAuthTokenEventCallback | <p>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.</p> <p>pAuthTokenEventCallback is needed only when changing a password without a channel in order to get a response from the request. The application does not have to send a login reissue in this case.</p> |
| proxyDomain | An RsslBuffer that specifies the domain for authenticated proxies. |
| proxyHostName | An RsslBuffer that specifies the proxy's host name. |
| proxyPasswd | An RsslBuffer that specifies the password for authenticated proxies. |
| proxyPort | An RsslBuffer that specifies the proxy's port. |
| proxyUserName | An RsslBuffer that specifies the username for authenticated proxies. |
| renewalMode | A RsslReactorOAuthCredentialRenewalMode that specifies the mode in which the Enterprise Transport API submits OAuth credential renewals. For available ENUMs and their descriptions, refer to Section 6.9.2.6. |

Table 92: rsslReactorSubmitOAuthCredentialRenewal Options

6.9.2.6 RsslReactorOAuthCredentialRenewalMode Enums

| MODE | DESCRIPTION |
|--|--|
| RSSL_ROC_RT_RENEW_TOKEN_WITH_PASSWORD | Use this renewal mode when normally submitting a password to obtain an access and refresh token. |
| RSSL_ROC_RT_RENEW_TOKEN_WITH_PASSWORD_CHANGE | Use this renewal mode only when changing the application's password. |

Table 93: RsslReactorOAuthCredentialRenewalMode Enums

6.10 JSON to Refinitiv Wire Format Protocol Conversion for WebSocket Support

6.10.1 Consumer and 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 Refinitiv Wire Format sub-protocol, use `RsslConnectOptions.wsOpts.protocols = "rssl.rwf"`.
 - This is the only requirement. For an example of setting `wsOpts`, refer to Section 6.10.1.1.
 - For remaining tasks for managing the Consumer, refer to Chapter 3, Building an Open Message Model Consumer.
- To support WebSocket connection requests for JSON2 and Refinitiv Wire Format sub-protocols (either alone, or in tandem with Refinitiv Wire Format), 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 Refinitiv Wire Format), after initializing the Interactive Provider, you must then perform the following:

- Connect the Consumer over an encrypted WebSocket. For an example, refer to Section 6.10.1.2.
- Define the Reactor JSON Converter event callback by using the `jsonConversionEventCallback` function. For an example, refer to Section 6.10.1.3.
- Define the `ServiceName` to `ServiceId` callback (using the `serviceNameToIdCallback` function). For an example, refer to Section 6.10.1.3.
- Clear and populate a JSON Converter options structure using the callbacks from the preceding tasks. For an example, refer to Section 6.10.1.4.
- Initialize the Reactor JSON Converter using the `rsslReactorInitJsonConverter` function passing in the JSON converter options structure. For an example, refer to Section 6.10.2.4.
- For remaining tasks on managing the Consumer, refer to Chapter 3, Building an Open Message Model Consumer.

6.10.1.1 Example: Initializing the Consumer to Support WebSocket Connections for Refinitiv Wire Format 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
}
```

6.10.1.2 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;
}
```

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

```

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

    return RSSL_RC_CRET_SUCCESS;
}

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

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

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

    return RSSL_RET_FAILURE;
}

```

6.10.1.4 Example: Clear and Populate a JSON Converter Options Structure

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

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

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

6.10.1.5 Example: Initialize the Reactor JSON Converter

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

6.10.2 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 Refinitiv Wire Format sub-protocol, use **RsslBindOptions.wsOpts.protocols = "rssl.rwf"**.
 - This is the only requirement. For an example of setting **wsOpts**, refer to Section 6.10.2.1.
 - For remaining tasks for managing the Interactive Provider, refer to Chapter 4, Building an Open Message Model Interactive Provider.
- To support WebSocket connection requests for both JSON2 and Refinitiv Wire Format 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 Refinitiv Wire Format), 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 6.10.2.2.
- Define the **ServiceName** to **ServiceId** callback (using the **serviceNameToIdCallback** function). For an example, refer to Section 6.10.2.2.
- Clear and populate a JSON converter options structure using the callbacks from the preceding tasks. For an example, refer to Section 6.10.2.3.
- Initialize the Reactor JSON Converter using the **rsslReactorInitJsonConverter** function passing in the JSON converter options structure. For an example, refer to Section 6.10.2.4.
- For remaining tasks on managing the Interactive Provider, refer to Chapter 4, Building an Open Message Model Interactive Provider.

6.10.2.1 Example: Initializing the Publisher to Support WebSocket Connections for Refinitiv Wire Format Only

To support WebSocket connections for only the Refinitiv Wire Format 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;
```

6.10.2.2 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
}
```

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

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

rsslClearReactorJsonConverterOptions(&jsonConverterOptions); // Clear allocated structure

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

6.10.2.4 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();
}
```

6.10.3 RsslReactorJsonConverterOptions Structure

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

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

Table 94: RsslReactorJsonConverterOptions Structure Members


| MEMBER | DESCRIPTION |
|------------------------------|--|
| outputBufferSize | <p>Sets the size (an RsslUInt32) that the converter allocates for its output buffer. Defaults to 65535.</p> <p> WARNING! If the output buffer is not large enough, JSON/Refinitiv Wire Format conversion will fail.</p> |
| pDictionary | <p>Sets the data dictionary (RsslDataDictionary) that Enterprise Transport API uses to initialize the Refinitiv Wire Format/JSON converter.</p> <p>pDictionary defaults to 0.</p> |
| pJsonConversionEventCallback | <p>Specifies the callback function (RsslReactorJsonConversionEventCallback) that receives the RsslReactorJsonConversionEvent if the JSON converter fails to convert a message.</p> <p>pJsonConversionEventCallback defaults to 0.</p> <ul style="list-style-type: none"> For further details on RsslReactorJsonConversionEventCallback, refer to Section 6.10.4. For further details on RsslReactorJsonConversionEvent, refer to Section 6.10.7. |
| pServiceNameToIdCallback | <p>Specifies the callback function (RsslReactorServiceNameToIdCallback) that handles conversion of the ServiceName to ServiceId.</p> <p>pServiceNameToIdCallback defaults to 0.</p> <p>For further details on RsslReactorServiceNameToIdCallback, refer to Section 6.10.5.</p> |
| userSpecPtr | <p>Specifies a user-defined pointer which is retrieved in the callback function.</p> <p>pUserSpec is a void* type and defaults to 0.</p> |

Table 94: RsslReactorJsonConverterOptions Structure Members (Continued)

6.10.4 RsslReactorJsonConversionEventCallback Function

The **RsslReactorJsonConversionEventCallback** function communicates conversion information when the JSON converter fails to convert JSON to Refinitiv Wire Format messages by providing an **RsslReactorJsonConversionEvent** structure to the application. For further details on **RsslReactorJsonConversionEvent**, refer to Section 6.10.7.

6.10.5 RsslReactorServiceNameToIdCallback 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 6.10.6.

- 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 95: RsslReactorServiceNameToIdCallback Parameters

6.10.6 RsslReactorServiceNameToIdEvent Structure

The **RsslReactorServiceNameToIdEvent** 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. |

Table 96: RsslReactorServiceNameToIdEvent Structure Members

6.10.7 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 97: RsslReactorJsonConversionEvent Structure Members

6.11 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 6.11.1 - Section 6.11.3.

6.11.1 General Reactor Utility Functions

| FUNCTION NAME | DESCRIPTION |
|--|---|
| <code>rsslReactorGetChannelInfo</code> | Allows the application to query RsslReactorChannel negotiated parameters and settings and retrieve all current settings. This includes maxFragmentSize and negotiated compression information as well as many other values. For a full list of available settings, refer to the RsslReactorChannelInfo structure defined in Section 6.11.2. This function calls the Enterprise Transport API rsslGetChannelInfo function which has its use and return values described in the Enterprise Transport API C Edition <i>Developers Guide</i> . |
| <code>rsslReactorIoctl</code> | Allows the application to change various settings associated with the RsslReactorChannel . The available options are defined in Section 6.11.3. This function calls the Enterprise Transport API rsslIoctl function which has its use and return values described in the Enterprise Transport API C Edition <i>Developers Guide</i> . |

Table 98: Reactor Utility Functions

6.11.2 RsslReactorChannelInfo Structure Members

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

| STRUCTURE MEMBER | DESCRIPTION |
|------------------------------|--|
| <code>rsslChannelInfo</code> | Returns the underlying RsslChannel information. This includes maxFragmentSize , number of output buffers, compression information, and more. The RsslChannelInfo function structure is fully described in the Enterprise Transport API C Edition <i>Developers Guide</i> . |

Table 99: RsslReactorChannelInfo Structure Members

6.11.3 rsslReactorIoctl Option Values

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

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:

- Authentication Token Management (for details, refer to Section 7.3)
- Service Discovery (for details, refer to Section 7.4)
- Consuming Market Data (for details, refer to Section 7.5)
- Login Reissue (for details, refer to Section 7.3.3)

By default, for cloud connections the Enterprise Transport API connects to a server in the **us-east** 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 Authentication Token Management

7.3.1 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/>.

7.3.2 Obtaining Initial Access and Refresh Tokens

To obtain an access token, the RTSDK API sends its username, **Client_ID** (from **RsslReactorOAuthCredential** as described in Section 6.9.2.1), 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 7.3.3). 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:

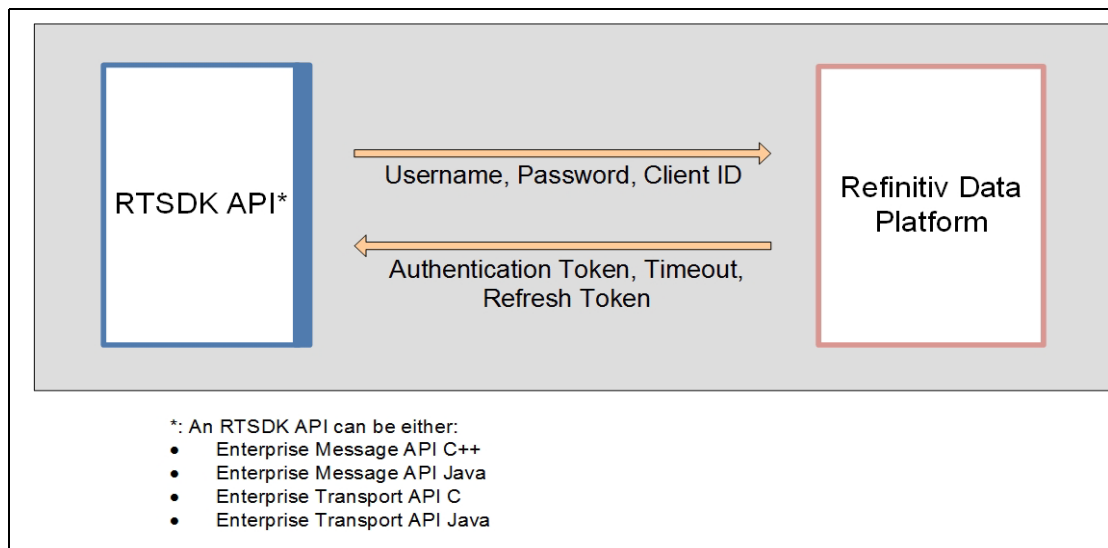


Figure 9. Obtaining an Authentication Token

7.3.3 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 7.3.2.

The login reissue process is illustrated in the following diagram:

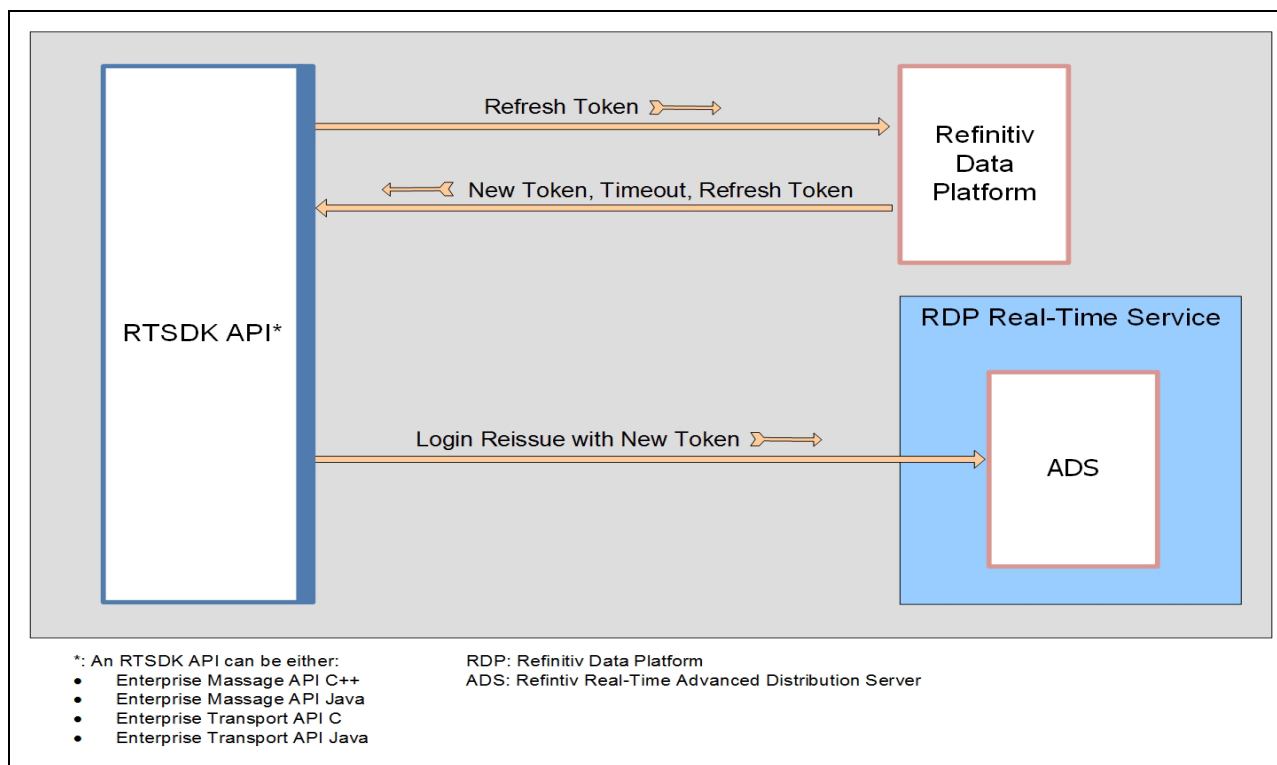


Figure 10. Login Reissue

7.3.4 Managing the Password and Client Secret

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

If you configure the Enterprise Transport API reactor to not store the password and client secret, whenever the Enterprise Transport API needs these credentials (i.e., when obtaining an initial access token or new refresh token), the API sends the **RsslReactorOAuthCredentialEvent** callback to the application. For details on the **RsslReactorOAuthCredentialEvent** callback, refer to Section 6.9.2.2.

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

- For details on **RsslReactorOAuthCredentialRenewal**, refer to Section 6.9.2.3.
- For details on the **rsslReactorSubmitOAuthCredentialRenewal** method, refer to Section 6.9.2.4.



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

7.3.5 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.

7.4 Service Discovery

After obtaining a token (for details, refer to Section 7.3.2), the Enterprise Transport API can perform a service discovery against the Refinitiv Data Platform to obtain connection details for the Refinitiv Real-Time Advanced Distribution Server in the cloud. 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). 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 a Refinitiv Real-Time Advanced Distribution Server (i.e., an endpoint) set up for failover and 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`** cloud location. An endpoint setup for failover lists multiple locations in its location field (e.g., **`location: [us-east-1a, us-east-1b]`**). If multiple endpoints are set up for failover, the Enterprise Transport API chooses to connect to the first endpoint listed.

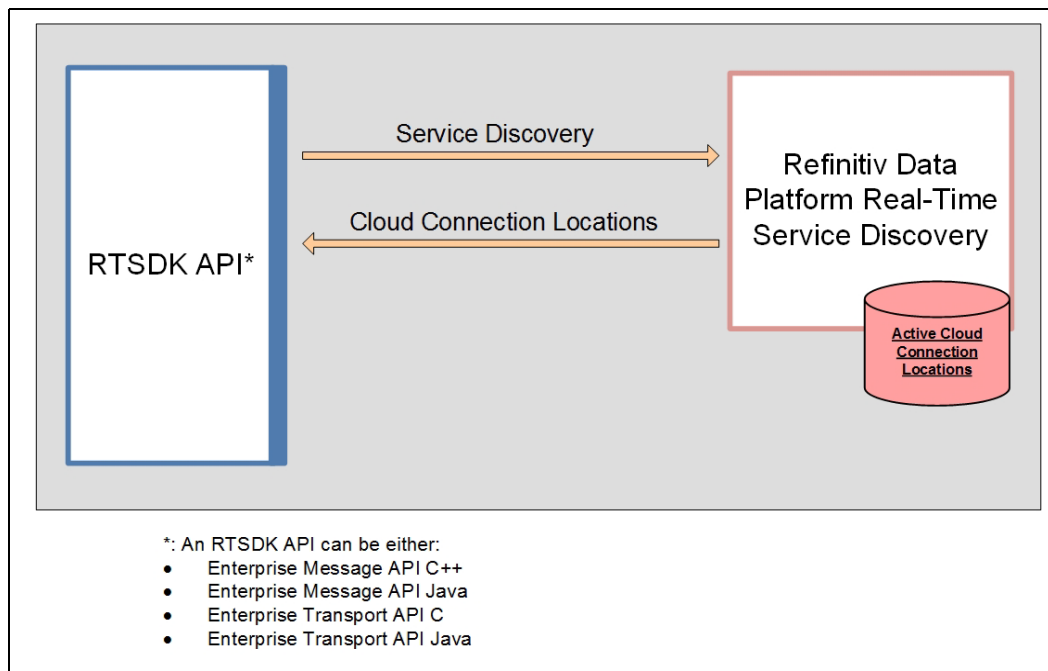
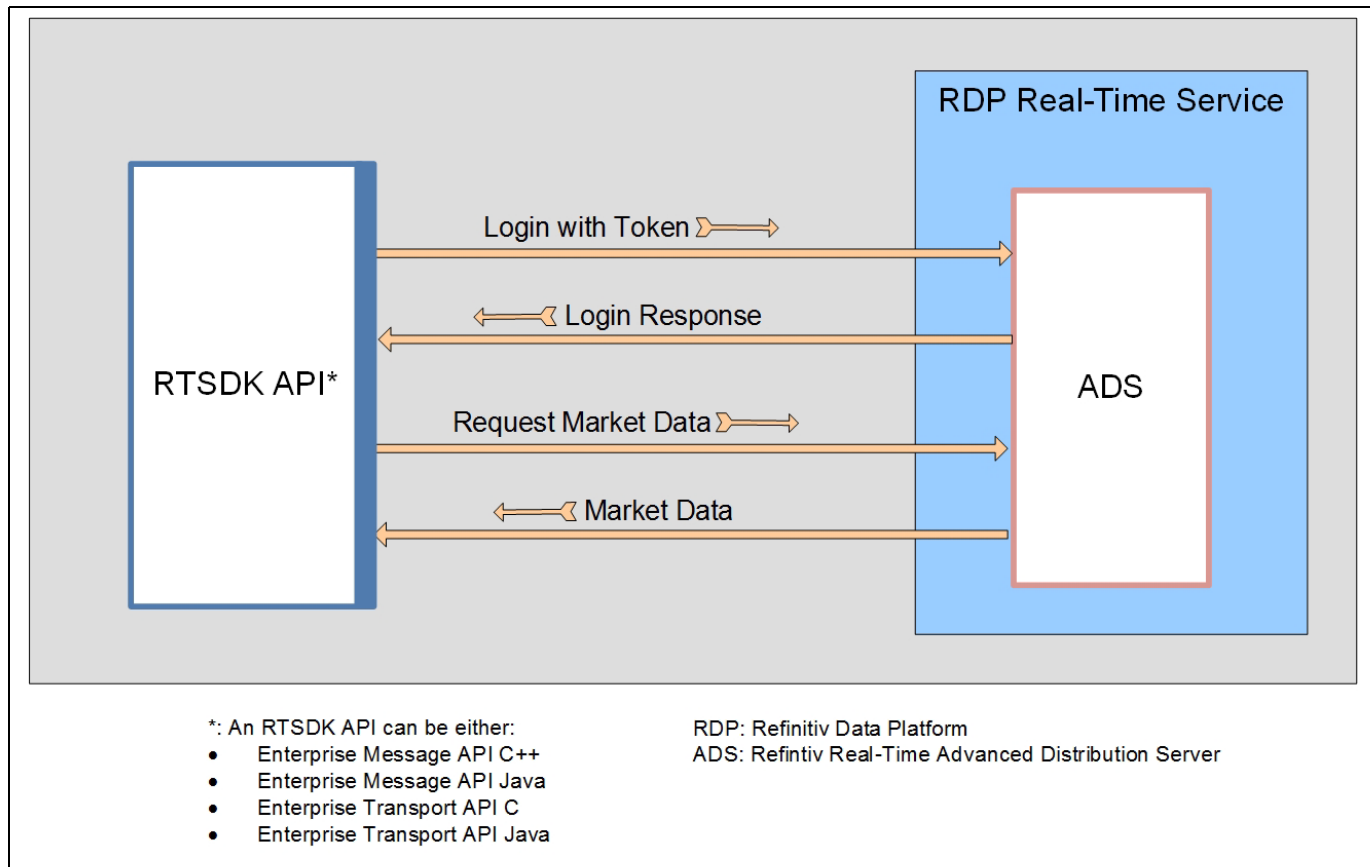


Figure 11. Service Discovery

7.5 Consuming Market Data

After obtaining its login token (for details, refer to Section 7.3.2) and running a service discovery (for details, refer to Section 7.4), 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 7.3.3).



7.6 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:
 - Retry with username and password for error codes 400 and 401
 - Stop retry the request for error codes 403 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

7.7 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 7.7.1)
- Disabling the watchlist (for details, refer to Section 7.7.2)
- Query service discovery (for details, refer to Section 7.7.3)

7.7.1 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 7.3.2)
- Queries service discovery (according to the details in Section 7.4)
- Consumes market data (according to the details in Section 7.5)
- Manages login reissues when needed on a cyclical basis (according to the details in Section 7.3.3)

A special use case exists for connecting to a specific (i.e., non-default) host. As described in Section 7.4, 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.

7.7.2 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 7.3.2)
- If needed, queries service discovery (according to the details in Section 7.4)

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**).

7.7.3 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 7.3.2)
- Queries service discovery (according to the details in Section 7.4)

7.8 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.

7.8.1 Logged Request Information

With logging turned on in the fashion mentioned in Section 7.8, 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
```

7.8.2 Logged Response Information

With logging turned on in the fashion mentioned in Section 7.8, 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 Refinitiv Domain Model-specific representations of Open Message Model administrative domain models. This Value Added Component contains structures that represent messages within the Login, Source Directory, and Dictionary domains (discussed in Table 100). 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 Open Message Model devices (i.e., Refinitiv Real-Time Distribution System infrastructure), and also ensures that encoding/decoding for these domain models follow Open Message Model-specified formatting rules. Applications can use this Value Added Component directly to help with encoding, decoding, and representation of these domain models. When using the Enterprise Transport API Reactor, this component is embedded to manage and present callbacks with a domain-specific representation of content.

Where possible, the members of an Administration Domain Model Representation structure are represented in the structure with the same **RsslDataType** that is specified for the element by the domain model. In cases where multiple elements are part of a more complex container such as an **RsslMap** or **RsslElementList**, 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 | 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. The Dictionary domain is considered an administrative domain. Many Refinitiv components require this content and expect it to follow the domain model definition. For further details refer to Section 8.5. |
| Login | Authenticates users and advertises/requests features that are not specific to a particular domain. Use of and support for this domain is required for all Open Message Model applications. Login is considered an administrative domain. Many Refinitiv components require this content and expect it to conform to the domain model definition. For further details refer to Section 8.3. |
| Source Directory | Advertises information about available services and their state, quality of service, and capabilities. This domain also conveys any group status and group merge information. Interactive and non-interactive provider applications require support for this domain. Refinitiv strongly recommends that consumers request this domain. Source Directory is considered an administrative domain, and many Refinitiv components expect this content and require it to conform to the domain model definition. For further details, refer to Section 8.4. |

Table 100: Domains Representations in the Administration Domain Model Value Added Component

8.2 Refinitiv Domain Model 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 Refinitiv Source Sink Library Refinitiv Domain Model Message Base Structure Members

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

| STRUCTURE MEMBER | DESCRIPTION |
|------------------|---|
| streamId | Required. A unique signed-integer identifier associated with all messages flowing in the stream. <ul style="list-style-type: none"> Positive values indicate a consumer-instantiated stream, typically via a request message. Negative values indicate a provider-instantiated stream, often associated with Non-Interactive Providers. |
| domainType | Required. Identifies the specific domain message model type. If value is less than 128 , domain is a 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 101: RsslRDMMsgBase Structure Members

8.2.2 Refinitiv Source Sink Library Refinitiv Domain Model Message Types

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

| DOMAIN TYPE | REFINITIV DOMAIN MODEL: MESSAGE TYPE | REFINITIV DOMAIN MODEL: MESSAGE STRUCTURE |
|---|--------------------------------------|---|
| RSSL_DMT_LOGIN (RsslRDMLLoginMsg) Refer to Section 8.3 | RDM_LG_MT_REQUEST | RsslRDMLLoginRequest |
| | RDM_LG_MT_REFRESH | RsslRDMLLoginRefresh |
| | RDM_LG_MT_STATUS | RsslRDMLLoginStatus |
| | RDM_LG_MT_CLOSE | RsslRDMLLoginClose |
| | RDM_LG_MT_CONSUMER_CONNECTION_STATUS | RsslRDMLLoginConsumerConnectionStatus |
| RSSL_DMT_SOURCE (RsslRDMDirectoryMsg) Refer to Section 8.4 | RDM_DR_MT_REQUEST | RsslRDMDirectoryRequest |
| | RDM_DR_MT_REFRESH | RsslRDMDirectoryRefresh |
| | RDM_DR_MT_UPDATE | RsslRDMDirectoryUpdate |
| | RDM_DR_MT_STATUS | RsslRDMDirectoryStatus |
| | RDM_DR_MT_CLOSE | RsslRDMDirectoryClose |
| | RDM_DR_MT_CONSUMER_STATUS | RsslRDMDirectoryConsumerStatus |

Table 102: RsslRDMMsg

| DOMAIN TYPE | REFINITIV DOMAIN MODEL: MESSAGE TYPE | REFINITIV DOMAIN MODEL: MESSAGE STRUCTURE |
|--|---|--|
| RSSL_DMT_DICTIONARY (RsslRDMDictionaryMsg) Refer to Section 8.5 | RDM_DC_MT_REQUEST | RsslRDMDictionaryRequest |
| | RDM_DC_MT_REFRESH | RsslRDMDictionaryRefresh |
| | RDM_DC_MT_STATUS | RsslRDMDictionaryStatus |
| | RDM_DC_MT_CLOSE | RsslRDMDictionaryClose |

Table 102: RsslRDMMsg (Continued)

8.2.3 Refinitiv Source Sink Library Refinitiv Domain Model Encoding and Decoding Functions

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

| FUNCTION NAME | DESCRIPTION |
|------------------|--|
| rsslEncodeRDMMsg | Used to encode any message that the RsslRDMMsg can represent. This function takes the RsslRDMMsg as a parameter. |
| rsslDecodeRDMMsg | Used to decode any message that the RsslRDMMsg can represent. This function populates the RsslRDMMsg and leverages the Value Added Utility message buffer (refer to Section 9.2). NOTE: The decoded message may refer to encoded data from the original RsslMsg . If the message is to be stored, the appropriate copy function for the decoded RsslRDMMsg should be used to create a full copy. |

Table 103: Refinitiv Domain Model Encoding and Decoding Functions

8.3 Refinitiv Domain Model Login Domain

The Login domain registers a user with the system, after which the user can request¹, post², or provide³ Open Message Model 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 Refinitiv Source Sink Library Refinitiv Domain Model Login Request

A **Login Request** message is encoded and sent by Open Message Model 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 Open Message Model provider can use the login request information to authenticate users with the Data Access Control System.

The **RsslRDMLLoginRequest** represents all members of a login request message and allows for simplified use in Open Message Model applications that leverage Refinitiv Domain Models. 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 Refinitiv Source Sink Library Refinitiv Domain Model Login Request Structure Members

| STRUCTURE MEMBER | DESCRIPTION |
|--------------------------|---|
| allowSuspectData | <p>Optional. If present, a flags value of RDM_LG_RQF_HAS_ALLOW_SUSPECT_DATA should be specified. If absent, a default value of 1 is assumed.</p> <ul style="list-style-type: none"> 1: Indicates that the consumer application allows for suspect streamState information. 0: Indicates that the consumer application prefers any suspect data to result in the stream being closed with an RSSL_STREAM_CLOSED_RECOVER state. |
| applicationId | <p>Optional. If present, a flags value of RDM_LG_RQF_HAS_APPLICATION_ID should be specified.</p> <p>When populated, should contain the Data Access Control System applicationId. If the server authenticates with the Data Access Control System, the consumer application may be required to pass in a valid application id.</p> <p>If initializing RsslRDMLLoginRequest using rsslInitDefaultRDMLLoginRequest, an applicationId of 256 will be used.</p> |
| applicationName | <p>Optional. If present, a flags value of RDM_LG_RQF_HAS_APPLICATION_NAME should be specified.</p> <p>When present, the applicationName in the login request identifies the Open Message Model consumer or non-interactive provider.</p> <p>If initializing RsslRDMLLoginRequest using rsslInitDefaultRDMLLoginRequest, applicationName is set to upa.</p> |
| authenticationExtended | <p>Optional. If present, a flags value of RDM_LG_RQF_HAS_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. Indicate 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 104: RsslRDMLLoginRequest 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 RsslRDMLLoginRequest using rsslInitDefaultRDMLLoginRequest, 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 104: RsslRDMLLoginRequest 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 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 104: 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 Refinitiv Source Sink Library Refinitiv Domain Model 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 105: 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 105: RsslRDMLoginRequest Flags (Continued)

8.3.1.3 Refinitiv Source Sink Library Refinitiv Domain Model Login Request Utility Functions

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

Table 106: RsslRDMLoginRequest Utility Functions

8.3.2 Refinitiv Source Sink Library Refinitiv Domain Model Login Refresh

A **Login Refresh** message is encoded and sent by an Open Message Model 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 **RsslRDMLLoginRefresh** represents all members of a login refresh message and allows for simplified use in Open Message Model applications that leverage Refinitiv Domain Models. 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 107: RsslRDMLLoginRefresh Structure Members

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

Table 107: Rss1RDMLoginRefresh 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 Open Message Model Posting:</p> <ul style="list-style-type: none"> • 1: The provider supports Open Message Model Posting and the user is permissioned. • 0: The provider supports the Open Message Model Post feature, but the user is not permissioned. • If this element is not present, then the server does not support Open Message Model 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 107: Rss1RDMLLoginRefresh 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 107: Rss1RDMLginRefresh 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 108: Rss1RDMLginRefresh Flags

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

Table 108: Rss1RDMLoginRefresh Flags (Continued)

8.3.2.3 Login Refresh Utility Functions

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

Table 109: RsslRDMLLoginRefresh Utility Functions

8.3.2.4 Server Info Structure Members

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

Table 110: RsslRDMServerInfo Structure Members

8.3.2.5 Server Info Flag Enumeration Values

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

Table 111: RsslRDMServerInfo Flags

8.3.2.6 Server Info Utility Functions

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

Table 112: RsslRDMServerInfo Utility Functions

8.3.3 Refinitiv Source Sink Library Refinitiv Domain Model Login Status

Open Message Model 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 **RsslRDMLLoginStatus** represents all members of a login status message and allows for simplified use in Open Message Model applications that leverage Refinitiv Domain Models. 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 | Optional. If present, a flags value of RDM_LG_STF_HAS_AUTHN_ERROR_CODE should be specified. 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 |
| authenticationErrorText | Optional. If present, a flags value of RDM_LG_STF_HAS_AUTHN_ERROR_TEXT should be specified. Specifies any text associated with the specified authenticationErrorCode . For further information, refer to the <i>Refinitiv Real-Time Distribution System Authentication User Manual</i> . ^a |
| flags | Required. Indicates the presence of optional login status members. For details, refer to Section 8.3.3.2. |
| rdmMsgBase | Required. Contains general message information, such as streamId and domainType . |
| state | Optional. If present, a flags value of RDM_LG_STF_HAS_STATE should be specified. Indicates the state of the login stream. When rejecting a login the state should be: <ul style="list-style-type: none"> streamState = RSSL_STREAM_CLOSED or RSSL_STREAM_CLOSED_RECOVER dataState = RSSL_DATA_SUSPECT stateCode = RSSL_SC_NOT_ENTITLED For more information on RsslState , refer to the Enterprise Transport API C Edition <i>Developers Guide</i> . |

Table 113: RsslRDMLLoginStatus 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 113: 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 | Indicates the presence of state . If absent, any previously conveyed state continues to apply. |
| RDM_LG_STF_HAS_USERNAME | Indicates the presence of userName . |
| RDM_LG_STF_HAS_USERNAME_TYPE | Indicates the presence of userNameType . If absent a userNameType of RDM_LOGIN_USER_NAME is assumed. |

Table 114: Rss1RDMLoginStatus Flags

8.3.3.3 Login Status Utility Functions

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

Table 115: RsslRDMLLoginStatus Utility Functions

8.3.4 Refinitiv Source Sink Library Refinitiv Domain Model Login Close

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

8.3.4.1 Login Close Structure Member

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

Table 116: RsslRDMLLoginClose Structure Member

8.3.4.2 Login Close Utility Functions

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

Table 117: RsslRDMLLoginClose Utility Functions

8.3.5 Refinitiv Source Sink Library Refinitiv Domain Model Consumer Connection Status

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

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

The **RsslRDMLLoginConsumerConnectionStatus** relies on the **RsslGenericMsg** and represents all members necessary for applications that leverage Refinitiv Domain Models. 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 streamId and domainType . Indicates the Login Message type (for login connection status, set to LoginMsgType.CONSUMER_CONNECTION_STATUS). |
| warmStandbyInfo | Optional. Includes RsslRDMLoginWarmStandbyInfo to convey the state of the upstream provider. For details, refer to Section 8.3.5.3. If present, a flags value of RDM_LG_CCSF_HAS_WARM_STANDBY_INFO should be specified. |

Table 118: RsslRDMLoginConsumerConnectionStatus Structure Members

8.3.5.2 Login Consumer Connection Status Flag Enumeration Value

| FLAG ENUMERATION | DESCRIPTION |
|-----------------------------------|--|
| RDM_LG_CCSF_HAS_WARM_STANDBY_INFO | Indicates presence of warmStandbyInfo . |

Table 119: RsslRDMLoginConsumerConnectionStatus Flags

8.3.5.3 Login Warm Standby Info Structure Members

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

Table 120: RsslRDMLoginWarmStandbyInfo Structure Members

8.3.5.4 Login Warm Standby Mode Enumeration Values

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

Table 121: RDMLoginServerTypes Enumeration Values

8.3.5.5 Login Consumer Connection Status Utility Functions

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

Table 122: RsslRDMLLoginConsumerConnectionStatus Utility Functions

8.3.6 Login Round Trip Time Message Use

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

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

8.3.6.1 Login Round Trip Time Members

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

Table 123: 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 124: Login Round Trip Time Flag Enumeration Values

8.3.6.3 Login Round Trip Time Utility Functions

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

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

Table 125: Login Round Trip Time Utility Functions

8.3.7 Login Post Message Use

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

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

8.3.8 Login Ack Message Use

Open Message Model 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 Refinitiv Source Sink Library Refinitiv Domain Model Login Message Union

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

8.3.9.1 Login Union

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

Table 126: RsslRDMLLoginMsg Union Members

8.3.9.2 Login Message Utility Functions

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

Table 127: RsslRDMLLoginMsg Utility Functions

8.3.10 Refinitiv Source Sink Library Refinitiv Domain Model Login Encoding and Decoding

8.3.10.1 Directory Login Encoding and Decoding Functions

| FUNCTION NAME | DESCRIPTION |
|-------------------------------------|--|
| <code>rsslDecodeRDMLLoginMsg</code> | Decodes an Refinitiv Domain Model Login message. This function populates the RsslRDMLLoginMsg and leverages the Value Added Utility message buffer (refer to Section 9.2). Alternatively, rsslDecodeRDMMsg can be used to decode into an RsslRDMMsg . |
| <code>rsslEncodeRDMLLoginMsg</code> | Encodes an Refinitiv Domain Model Login message. This function takes the RsslRDMLLoginMsg as a parameter. Alternately, rsslEncodeRDMMsg can be used if encoding from an RsslRDMMsg . |

Table 128: Refinitiv Domain Model Login Encoding and Decoding Functions

8.3.10.2 Encoding a Login Request

```

RsslEncodeIterator encodeIter;
RsslRDMLLoginRequest loginRequest;

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

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

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

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

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

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

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

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

```

Code Example 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;
RsslRDMLLoginRefresh loginRefresh;

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

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

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

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

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

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

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

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

```

Code Example 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;
RsslRDMLginRefresh *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 RsslRDMLginRefresh. */
    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 Refinitiv Source Sink Library Refinitiv Domain Model 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 Open Message Model 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 Refinitiv Source Sink Library Refinitiv Domain Model Directory Request

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

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

Table 129: RsslRDMDirectoryRequest Structure Members

8.4.1.2 Directory Request Flag Enumeration Values

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

Table 130: RsslRDMDirectoryRequest Flags

8.4.1.3 Directory Request Utility Functions

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

Table 131: RsslRDMDirectoryRequest Utility Functions

8.4.2 Refinitiv Source Sink Library Refinitiv Domain Model Directory Refresh

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

The **RsslRDMDirectoryRefresh** represents all members of a directory refresh message and is easily used in Open Message Model applications that leverage Refinitiv Domain Models. 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 RsslRDMDirectoryRequest . The available flags are: <ul style="list-style-type: none"> • RDM_DIRECTORY_SERVICE_INFO_FILTER == 0x01 • RDM_DIRECTORY_SERVICE_STATE_FILTER == 0x02 • RDM_DIRECTORY_SERVICE_GROUP_FILTER == 0x04 • RDM_DIRECTORY_SERVICE_LOAD_FILTER == 0x08 • RDM_DIRECTORY_SERVICE_DATA_FILTER == 0x10 • RDM_DIRECTORY_SERVICE_LINK_FILTER == 0x20 |
| flags | Required. Indicates the presence of optional directory refresh members. Refer to Section 8.4.2.2. |
| rdmMsgBase | Required. Contains general message information, such as streamId and domainType . |

Table 132: RsslRDMDirectoryRefresh Structure Members

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

Table 132: RsslRDMDirectoryRefresh Structure Members (Continued)

8.4.2.2 Directory Refresh Flag Enumeration Values

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

Table 133: RsslRDMDirectoryRefresh Flags

8.4.2.3 Directory Refresh Utility Functions

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

Table 134: RsslRDMDirectoryRefresh Utility Functions

8.4.3 Refinitiv Source Sink Library Refinitiv Domain Model Directory Update

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

Table 135: RsslRDMDirectoryUpdate Structure Members

8.4.3.2 Directory Update Flag Enumeration Values

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

Table 136: RsslRDMDirectoryUpdate Flags

8.4.3.3 Directory Update Utility Functions

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

Table 137: RsslRDMDirectoryUpdate Utility Functions

8.4.4 Refinitiv Source Sink Library Refinitiv Domain Model Directory Status

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

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

Table 138: RsslRDMDirectoryStatus Structure Members

8.4.4.2 Directory Status Flag Enumeration Values

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

Table 139: RsslRDMDirectoryStatus Flags

8.4.4.3 Directory Status Utility Functions

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

Table 140: RsslRDMDirectoryStatus Utility Functions

8.4.5 Refinitiv Source Sink Library Refinitiv Domain Model Directory Close

8.4.5.1 Directory Close Structure Member

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

Table 141: RsslRDMDirectoryClose Structure Member

8.4.5.2 Directory Close Utility Functions

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

Table 142: RsslRDMDirectoryClose Utility Functions

8.4.6 Refinitiv Source Sink Library Refinitiv Domain Model Consumer Status

The **Directory Consumer Status** is sent by Open Message Model 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 Refinitiv Domain Models. 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 143: RsslRDMDirectoryConsumerStatus Structure Members

8.4.6.2 Directory Consumer Status Service Structure Members

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

Table 144: RsslRDMConsumerStatusService Structure Members

8.4.6.3 Directory Consumer Status Utility Functions

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

Table 145: RsslRDMDirectoryConsumerStatus Utility Functions

8.4.7 Source Directory Refinitiv Domain Model Service

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

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

8.4.7.1 Refinitiv Source Sink Library Service Structure Members

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

Table 146: RsslRDMSERVICE Structure Members

8.4.7.2 Refinitiv Source Sink Library 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 147: RsslRDMSERVICE Flags

8.4.7.3 Refinitiv Source Sink Library Service Utility Functions

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

Table 148: RsslRDMSERVICE Utility Function

8.4.8 Source Directory Refinitiv Domain Model Service Info

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

8.4.8.1 Refinitiv Source Sink Library Service Info Members

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

Table 149: RsslRDMSERVICEINFO Structure Members

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

Table 149: RsslRDMServiceInfo Structure Members (Continued)

8.4.8.2 Refinitiv Source Sink Library 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 150: RsslRDMServiceInfo Flags

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

Table 150: Rss1RDMSERVICEINFO Flags (Continued)

8.4.8.3 Refinitiv Source Sink Library Service Info Utility Functions

| FUNCTION NAME | DESCRIPTION |
|-------------------------|--|
| rss1ClearRDMSERVICEINFO | Clears an Rss1RDMSERVICEINFO structure. Useful for structure reuse. |

Table 151: Rss1RDMSERVICEINFO Utility Functions

8.4.9 Source Directory Refinitiv Domain Model 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 Refinitiv Source Sink Library Service State Members

| STRUCTURE MEMBER | DESCRIPTION |
|-------------------|--|
| acceptingRequests | Indicates whether the immediate provider (to which the consumer is directly connected) can handle the request. Available values are: <ul style="list-style-type: none"> • 1: The service will accept new requests. • 0: The service does not currently accept new requests. If present, flags value of RDM_SVC_STF_HAS_ACCEPTING_REQS should be specified. |
| action | Required. Indicates how a cache of the service state should apply this information. For details on Rss1FilterEntry actions, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> . |
| flags | Required. Indicates the presence of optional service state members. For details refer to Section 8.4.9.2. |
| serviceState | Required. Indicates whether the original provider of the data can respond to new requests. Requests can still be made if so indicated by acceptingRequests . Available values are: <ul style="list-style-type: none"> • 1: The original provider of the data is available. • 0: The original provider of the data is not currently available. |
| status | This status should be applied to all open items associated with this service. If present, flags value of RDM_SVC_STF_HAS_STATUS should be specified. |

Table 152: Rss1RDMSERVICESTATE Structure Members

8.4.9.2 Refinitiv Source Sink Library 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 153: `RsslRDMSERVICEState` Flags

8.4.9.3 Refinitiv Source Sink Library Service State Utility Functions

| FUNCTION NAME | DESCRIPTION |
|---------------------------------------|---|
| <code>rsslClearRDMSERVICEState</code> | Clears an RsslRDMSERVICEState structure. Useful for structure reuse. |

Table 154: `RsslRDMSERVICEState` Utility Functions

8.4.10 Source Directory Refinitiv Domain Model Service Group State

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

8.4.10.1 Refinitiv Source Sink Library Service Group State Members

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

Table 155: `RsslRDMSERVICEGroupState` Structure Members

8.4.10.2 Refinitiv Source Sink Library 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 156: `RsslRDMSERVICEGROUPSTATE` Flags

8.4.10.3 Refinitiv Source Sink Library Service Group State Utility Functions

| FUNCTION NAME | DESCRIPTION |
|--|--|
| <code>rsslClearRDMSERVICEGROUPSTATE</code> | Clears an <code>RsslRDMSERVICEGROUPSTATE</code> structure. Useful for structure reuse. |

Table 157: `RsslRDMSERVICEGROUPSTATE` Utility Functions

8.4.11 Source Directory Refinitiv Domain Model Service Load

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

8.4.11.1 Refinitiv Source Sink Library Service Load Members

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

Table 158: `RsslRDMSERVICELOAD` Structure Members

8.4.11.2 Refinitiv Source Sink Library 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 159: RsslRDMSERVICELOAD Flags

8.4.11.3 Refinitiv Source Sink Library Service Load Utility Functions

| FUNCTION NAME | DESCRIPTION |
|-------------------------|--|
| rsslClearRDMSERVICELOAD | Clears an RsslRDMSERVICELOAD structure. Useful for structure reuse. |

Table 160: RsslRDMSERVICELOAD Utility Functions

8.4.12 Source Directory Refinitiv Domain Model Service Data

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

8.4.12.1 Refinitiv Source Sink Library Service Data Members

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

Table 161: RsslRDMSERVICEData Structure Members

8.4.12.2 Refinitiv Source Sink Library Service Load Data Enumeration Values

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

Table 162: RsslRDMSERVICEData Flags

8.4.12.3 Refinitiv Source Sink Library Service Data Utility Functions

| FUNCTION NAME | DESCRIPTION |
|-------------------------|--|
| rsslClearRDMSERVICEData | Clears an RsslRDMSERVICEData structure. Useful for structure reuse. |

Table 163: RsslRDMSERVICEData Utility Functions

8.4.13 Source Directory Refinitiv Domain Model Service Link Information

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

The **RsslRDMSERVICELinkInfo** structure contains an array of **RsslRDMSERVICELink** structures that each represents an upstream source.

8.4.13.1 Refinitiv Source Sink Library Service Link Info Members

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

Table 164: RsslRDMSERVICELinkInfo Structure Members

8.4.13.2 Refinitiv Source Sink Library Service Link Info Utility Functions

| FUNCTION NAME | DESCRIPTION |
|-----------------------------|--|
| rsslClearRDMSERVICELinkInfo | Clears an RsslRDMSERVICELinkInfo structure. Useful for structure reuse. |

Table 165: RsslRDMSERVICELinkInfo Utility Functions

8.4.14 Source Directory Refinitiv Domain Model 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 Refinitiv Source Sink Library 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 166: RsslRDMSERVICELink Structure Members

8.4.14.2 Refinitiv Source Sink Library 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 167: RsslRDMSERVICELink Flags

8.4.14.3 Refinitiv Source Sink Library Service Link Utility Functions

| FUNCTION NAME | DESCRIPTION |
|--------------------------------------|--|
| <code>rsslClearRDMSERVICELink</code> | Clears an <code>RsslRDMSERVICELink</code> structure. Useful for structure reuse. |

Table 168: RsslRDMSERVICELink Utility Functions

8.4.15 Source Directory Refinitiv Domain Model Sequenced Multicast Information

An `RsslRDMSERVICESeqMcastInfo` 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 *Refinitiv Domain Model Usage Guide*.
- For further information on Refinitiv Real Time Direct, refer to the *Refinitiv Real Time Direct Developers Guide*.

8.4.15.1 Refinitiv Source Sink Library Service Sequenced Multicast Information Structure

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

Table 169: RsslRDMSERVICESeqMcastInfo Structure Members

8.4.15.2 Refinitiv Source Sink Library Service Sequenced Multicast Info Enumeration Values

| FLAG ENUMERATION | DESCRIPTION |
|--|---|
| <code>RDM_SVC_SMF_HAS_SNAPSHOT_SERV</code> | Indicates the presence of <code>snapshotServer</code> . |
| <code>RDM_SVC_SMF_HAS_GAP_REC_SERV</code> | Indicates the presence of <code>gapRecoveryServer</code> . |
| <code>RDM_SVC_SMF_HAS_REF_DATA_SERV</code> | Indicates the presence of <code>refDataServer</code> . |
| <code>RDM_SVC_SMF_HAS_SMC_SERV</code> | Indicates the presence of <code>StreamingMcastChanServerList</code> . |
| <code>RDM_SVC_SMF_HAS_GMC_SERV</code> | Indicates the presence of <code>GapMcastChanServerList</code> . |

Table 170: Refinitiv Source Sink Library Refinitiv Domain Model Service Sequenced Multicast Info Enumeration Values

8.4.15.3 Refinitiv Source Sink Library 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 171: Refinitiv Source Sink Library Refinitiv Domain Model Address/Port Information Structure Members

8.4.15.4 Refinitiv Source Sink Library Sequenced Multicast Info Utility Functions

| UTILITY | DESCRIPTION |
|---------------------------------|--|
| rsslClearRDMMCAAddressPortInfo | Clears an RsslRDMAAddressPortInfo structure. |
| rsslClearRDMSERVICESeqMcastInfo | Clears an RsslRDMSERVICESeqMcastInfo structure. |

Table 172: RsslRDMSERVICESeqMcastInfo Utility Functions

8.4.16 Refinitiv Source Sink Library Refinitiv Domain Model Directory Message Union

This union can contain any of the Refinitiv Domain Model Directory message types. This is provided for use with directory-specific functionality.

8.4.16.1 Directory Union

| UNION MEMBERS | DESCRIPTION |
|----------------|--|
| rdmMsgBase | The message base information. |
| request | The RsslRDMDirectoryRequest as described in Section 8.4.1. |
| close | The RsslRDMDirectoryClose as described in Section 8.4.5. |
| refresh | The RsslRDMDirectoryRefresh as described in Section 8.4.2. |
| status | The RsslRDMDirectoryStatus as described in Section 8.4.4. |
| update | The RsslRDMDirectoryUpdate as described in Section 8.4.3. |
| consumerStatus | The RsslRDMDirectoryConsumerStatus as described in Section 8.4.6. |

Table 173: RsslRDMDirectoryMsg Union Members

8.4.16.2 Directory Message Utility Functions

| FUNCTION NAME | DESCRIPTION |
|--------------------------|--|
| rsslClearRDMDirectoryMsg | Clears an RsslRDMDirectoryMsg union. Useful for reuse. |
| rsslCopyRDMDirectoryMsg | Performs a deep copy of an RsslRDMDirectoryMsg structure. |

Table 174: RsslRDMDirectoryMsg Utility Functions

8.4.17 Source Directory Encoding and Decoding

8.4.17.1 Refinitiv Source Sink Library Refinitiv Domain Model Directory Encoding and Decoding Functions

| FUNCTION NAME | DESCRIPTION |
|--|---|
| <code>rsslEncodeRDMDirectoryMsg</code> | Used to encode an Refinitiv Domain Model Directory message. This function takes the RsslRDMDirectoryMsg as a parameter. Alternately, rsslEncodeRDMMsg can be used if encoding from an RsslRDMMsg . |
| <code>rsslDecodeRDMDirectoryMsg</code> | Used to decode an Refinitiv Domain Model Directory message. This function populates the RsslRDMDirectoryMsg and leverages the Value Added Utility message buffer (refer to Section 9.2). Alternately, rsslDecodeRDMMsg can be used to decode into an RsslRDMMsg . |

Table 175: Refinitiv Domain Model Directory Encoding and Decoding Functions

8.4.17.2 Encoding a Source Directory Request

```

RsslEncodeIterator encodeIter;
RsslRDMDirectoryRequest directoryRequest;

/* Clear the Directory Request structure. */
rsslClearRDMDirectoryRequest(&directoryRequest);

/* Set flags indicating presence of optional members. */
directoryRequest.flags =
    RDM_DR_RQF_HAS_SERVICE_ID
    | RDM_DR_RQF_STREAMING;

/* Set Service ID. */
directoryRequest.serviceId = 273;

/* Set ApplicationName. */
directoryRequest.filter =
    RDM_DIRECTORY_SERVICE_INFO_FILTER
    | RDM_DIRECTORY_SERVICE_STATE_FILTER
    | RDM_DIRECTORY_SERVICE_GROUP_FILTER;

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

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

```

Code Example 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;
RsslRDMMsg rdmMsg;
RsslRDMDirectoryRequest *pDirectoryRequest;

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

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

if (ret == RSSL_RET_SUCCESS
    && rdmMsg.rdmMsgBase.domainType == RSSL_DMT_SOURCE && rdmMsg.rdmMsgBase.rdmMsgType ==
    RDM_DR_MT_REQUEST)
{
    /* The message we decoded is an RsslRDMDirectoryRequest. */
    pDirectoryRequest = &rdmMsg.directoryMsg.request;

    /* Print if Info filter was requested. */
    if (pDirectoryRequest->filter & RDM_DIRECTORY_SERVICE_INFO_FILTER)
        printf("Info filter requested.\n");

    /* Print if State filter was requested. */
    if (pDirectoryRequest->filter & RDM_DIRECTORY_SERVICE_STATE_FILTER)
        printf("State filter requested.\n");

    /* Print if Group filter was requested. */
    if (pDirectoryRequest->filter & RDM_DIRECTORY_SERVICE_GROUP_FILTER)
        printf("Group filter requested.\n");

    /* Print service ID if present. */
    if (pDirectoryRequest->flags & RDM_DR_RQF_HAS_SERVICE_ID)
        printf("Service ID: %u\n", pDirectoryRequest->serviceId);
}

```

Code Example 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 = "Thomson Reuters";
serviceList[0].info.vendor.length = 15;

/* Build capabilities list. */
capabilitiesList[0] = RSSL_DMT_DICTIONARY;
capabilitiesList[1] = RSSL_DMT_MARKET_PRICE;
capabilitiesList[2] = RSSL_DMT_MARKET_BY_ORDER;

/* Set capabilities list. */
serviceList[0].info.capabilitiesList = capabilitiesList;
serviceList[0].info.capabilitiesCount = 3;

/* Build dictionary list to use with dictionariesProvidedList and dictionariesUsedList. */
dictionariesList[0].data = "RWFFld";
dictionariesList[0].length = 6;
dictionariesList[1].data = "RWFEEnum";
dictionariesList[1].length = 7;

/* Set dictionaries provided. */
serviceList[0].info.dictionariesProvidedList = dictionariesList;
serviceList[0].info.dictionariesProvidedCount = 2;

/* Set dictionaries used. */
serviceList[0].info.dictionariesUsedList = dictionariesList;
serviceList[0].info.dictionariesUsedCount = 2;

/* Build QoS list. */
qosList[0].timeliness = RSSL_QOS_TIME_REALTIME;
qosList[0].rate = RSSL_QOS_RATE_TICK_BY_TICK;
qosList[1].timeliness = RSSL_QOS_TIME_REALTIME;
qosList[1].rate = RSSL_QOS_RATE_JIT_CONFLATED;

/* Set QoS list. */
serviceList[0].info.qosList = qosList;
serviceList[0].info.qosCount = 2;

/** Build Service State for MY_SERVICE */
serviceList[0].state.flags = RDM_SVC_STF_HAS_ACCEPTING_REQS;
serviceList[0].state.serviceState = 1;
serviceList[0].state.acceptingRequests = 1;

/** Finish and encode. */

```



```

/* Set the array of services on the message. The refresh will information about 2 services*/
directoryRefresh.serviceList = serviceList;
directoryRefresh.serviceCount = 1;

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

/* Encode the message. */
ret = rsslEncodeRDMMMsg(&encodeIter, (RsslRDMMMsg*)&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;
RsslRDMMMsg 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 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_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 **RsslFieldEntry** or additional content related to its **fieldId**. For more information about the different types of dictionaries and their usage, refer to the Enterprise Transport API C Edition *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 Refinitiv Source Sink Library Refinitiv Domain Model Dictionary Request

A **Dictionary Request** message is encoded and sent by Open Message Model consumer applications. This message requests a dictionary from a service.

The **RsslRDMDictionaryRequest** represents all members of a dictionary request message and is easily used in Open Message Model applications that leverage Refinitiv Domain Models. 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 176: RsslRDMDictionaryRequest Structure Members

8.5.1.2 Dictionary Request Flag Enumeration Value

| FLAG ENUMERATION | DESCRIPTION |
|----------------------|---|
| RDM_DC_RQF_STREAMING | Indicates that the dictionary stream should remain open after the initial refresh. An open stream can listen for status messages that indicate changes to the dictionary version. For more information, see the Enterprise Transport API C Edition <i>Refinitiv Domain Model Usage Guide</i> . |

Table 177: RsslRDMDictionaryRequest Flag

8.5.1.3 Dictionary Request Utility Functions

| FUNCTION NAME | DESCRIPTION |
|--|--|
| <code>rsslClearRDMDictionaryRequest</code> | Clears an RsslRDMDictionaryRequest structure. Useful for structure reuse. |
| <code>rsslCopyRDMDictionaryRequest</code> | Performs a deep copy of an RsslRDMDictionaryRequest structure. |

Table 178: RsslRDMDictionaryRequest Utility Functions

8.5.2 Refinitiv Source Sink Library Refinitiv Domain Model Dictionary Refresh

A **Dictionary Refresh** message is encoded and sent by Open Message Model provider applications. This message transmits dictionary content in response to a request.

The **RsslRDMDictionaryRefresh** represents all members of a dictionary refresh message and is easy to use in Open Message Model applications that leverage Refinitiv Domain Models. 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 |
|-----------------------------|---|
| <code>dataBody</code> | When decoding, this points to the encoded data buffer with dictionary content. This buffer should be set on an RsslDecodeIterator and passed to the appropriate decode function according to the type . Not used when encoding. The dictionary is retrieved from the RsslDataDictionary structure. |
| <code>pDictionary</code> | Conditional (required when encoding). Points to an RsslDataDictionary object that contains content to encode. For more information on the RsslDataDictionary structure, refer to the Enterprise Transport API C Edition <i>Refinitiv Domain Model Usage Guide</i> . Not used when decoding. |
| <code>dictionaryId</code> | When decoding, this will be populated with the dictionary's ID. The Dictionary is retrieved from the RsslDataDictionary structure. This structure's presence is indicated by the RDM_DC_RFF_HAS_INFO flag. Not used when encoding. |
| <code>dictionaryName</code> | Required. Indicates the name of the dictionary being provided. |
| <code>flags</code> | Required. Indicates the presence of optional dictionary refresh members. For details, refer to Section 8.5.2.2. |
| <code>rdmMsgBase</code> | Required. Contains general message information like streamId and domainType . |
| <code>sequenceNumber</code> | Optional. A user-specified, item-level sequence number that the application can use to sequence messages in this stream. If present, a flags value of RDM_DC_RFF_HAS_SEQ_NUM should be specified. |
| <code>serviceId</code> | Required. Indicates the service ID of the service from which the dictionary is provided. |
| <code>startFid</code> | Maintains the state when encoding a dictionary across multiple messages. |
| |  WARNING! To ensure that all dictionary content is correctly encoded, the application should not modify this. |

Table 179: RsslRDMDictionaryRefresh Structure Members

| STRUCTURE MEMBER | DESCRIPTION |
|------------------|---|
| state | Required. Indicates the state of the dictionary stream. Defaults to a streamState of RSSL_STREAM_OPEN and a dataState of RSSL_DATA_OK . For more information on RsslState , refer to the Enterprise Transport API C Edition <i>Developers Guide</i> . |
| type | Required. Indicates the type of dictionary being provided. The dictionary encoder and decoder support the following types: <ul style="list-style-type: none"> • RDM_DICTIONARY_FIELD_DEFINITIONS == 1 • RDM_DICTIONARY_ENUM_TABLES == 2 |
| verbosity | Required. Indicates the amount of information desired from the dictionary. Available enumerations are: <ul style="list-style-type: none"> • RDM_DICTIONARY_INFO == 0x00: Provides version information only • RDM_DICTIONARY_MINIMAL == 0x03: Provides information needed for caching • RDM_DICTIONARY_NORMAL == 0x07: Provides all information needed for decoding • RDM_DICTIONARY_VERBOSE == 0x0F: Provides all information (including comments) Providers do not need to support the MINIMAL and VERBOSE filters. |
| version | When decoding, this will be populated with the dictionary's version string. Presence is indicated by the RDM_DC_RFF_HAS_INFO flag. Not used when encoding. The Dictionary is retrieved from the RsslDataDictionary structure. |

Table 179: RsslRDMDictionaryRefresh Structure Members (Continued)

8.5.2.2 Dictionary Refresh Flag Enumeration Values

| FLAG ENUMERATION | DESCRIPTION |
|------------------------|--|
| RDM_DC_RFF_CLEAR_CACHE | Indicates that stored payload information associated with the dictionary stream should be cleared. This might happen if some portion of data is known to be invalid. |
| RDM_DC_RFF_HAS_INFO | Indicates the presence of dictionaryId , version , and type . Not used when encoding. The encode function adds information to the encoded message when appropriate. |
| RDM_DC_RFF_HAS_SEQ_NUM | Indicates 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 180: RsslRDMDictionaryRefreshFlags

8.5.2.3 Dictionary Refresh Utility Functions

| FUNCTION NAME | DESCRIPTION |
|--|--|
| <code>rsslClearRDMDictionaryRefresh</code> | Clears an RsslRDMDictionaryRefresh structure. Useful for structure reuse. |
| <code>rsslCopyRDMDictionaryRefresh</code> | Performs a deep copy of an RsslRDMDictionaryRefresh structure. |

Table 181: RsslRDMDictionaryRefresh Utility Functions

8.5.3 Refinitiv Source Sink Library Refinitiv Domain Model Dictionary Status

Open Message Model 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 **RsslRDMDictionaryStatus** represents all members of a dictionary status message and allows for simplified use in Open Message Model applications that leverage Refinitiv Domain Models. 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 |
|-------------------------|---|
| <code>flags</code> | Required. Indicate the presence of optional dictionary status members. For details, refer to Section 8.5.3.2. |
| <code>rdmMsgBase</code> | Required. Contains general message information like streamId and domainType . |
| <code>state</code> | Optional. Indicates the state of the dictionary stream. For more information on RsslState , refer to the Enterprise Transport API C Edition <i>Developers Guide</i> . If present, a flags value of RDM_DC_STF_HAS_STATE should be specified. |

Table 182: RsslRDMDictionaryStatus Structure Members

8.5.3.2 Dictionary Status Flag Enumeration Value

| FLAG ENUMERATION | DESCRIPTION |
|-------------------------------------|--|
| <code>RDM_DC_STF_CLEAR_CACHE</code> | Indicates that any stored payload information associated with the dictionary stream should be cleared. This might happen if some portion of data is known to be invalid. |
| <code>RDM_DC_STF_HAS_STATE</code> | Indicates the presence of state . If absent, any previously conveyed state continues to apply. |

Table 183: RsslRDMDictionaryStatus Flags

8.5.3.3 Dictionary Status Utility Functions

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

Table 184: RsslRDMDictionaryStatus Utility Functions

8.5.4 Refinitiv Source Sink Library Refinitiv Domain Model 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 streamId and domainType . |

Table 185: RsslRDMDictionaryClose Structure Members

8.5.4.2 Dictionary Close Utility Functions

| FUNCTION NAME | DESCRIPTION |
|-----------------------------|--|
| rsslClearRDMDictionaryClose | Clears an RsslRDMDictionaryClose structure. Useful for structure reuse. |
| rsslCopyRDMDictionaryClose | Performs a deep copy of an RsslRDMDictionaryClose structure. |

Table 186: RsslRDMDictionaryClose Utility Functions

8.5.5 Refinitiv Source Sink Library Refinitiv Domain Model Dictionary Message Union

This union can contain any of the Refinitiv Domain Model Dictionary message types. This is provided for use with Dictionary specific functionality.

8.5.5.1 Dictionary Union

| UNION MEMBERS | DESCRIPTION |
|---------------|--|
| rdmMsgBase | The message base information. |
| request | The RsslRDMDictionaryRequest as described in Section 8.5.1. |
| close | The RsslRDMDictionaryClose as described in Section 8.5.4. |
| refresh | The RsslRDMDictionaryRefresh as described in Section 8.5.2. |
| status | The RsslRDMDictionaryStatus as described in Section 8.5.3. |

Table 187: RsslRDMDictionaryMsg Union Members

8.5.5.2 Dictionary Message Utility Functions

| FUNCTION NAME | DESCRIPTION |
|---------------------------|---|
| rsslClearRDMDictionaryMsg | Clears an RsslRDMDictionaryMsg union. Useful for reuse. |
| rsslCopyRDMDictionaryMsg | Performs a deep copy of an RsslRDMDictionaryMsg structure. |

Table 188: RsslRDMDictionaryMsg Utility Functions

8.5.6 Dictionary Encoding and Decoding

8.5.6.1 Refinitiv Source Sink Library Refinitiv Domain Model Dictionary Encoding and Decoding Functions

| FUNCTION NAME | DESCRIPTION |
|---|--|
| <code>rsslEncodeRDMDictionaryMsg</code> | Used to encode an Refinitiv Domain Model Dictionary message. This function takes the RsslRDMDictionaryMsg as a parameter. Alternately, rsslEncodeRDMMsg can be used if encoding from an RsslRDMMsg . |
| <code>rsslDecodeRDMDictionaryMsg</code> | Used to decode an Refinitiv Domain Model Directory message. This function populates the RsslRDMDictionaryMsg and leverages the Value Added Utility message buffer (refer to Section 9.2). Alternately, rsslDecodeRDMMsg can be used to decode into an RsslRDMMsg . |

Table 189: Refinitiv Domain Model Dictionary Encoding and Decoding Functions

8.5.6.2 Encoding a Dictionary Request

```

RsslEncodeIterator encodeIter;
RsslRDMDictionaryRequest dictionaryRequest;

/* Clear the Dictionary Request structure. */
rsslClearRDMDictionaryRequest(&dictionaryRequest);

/* Set flags. */
dictionaryRequest.flags = RDM_DC_RQF_STREAMING;

/* Set serviceId. */
dictionaryRequest.serviceId = 273;

/* Set verbosity. */
dictionaryRequest.verbosity = RDM_DICTIONARY_NORMAL;

/* Set dictionary name. */
dictionaryRequest.dictionaryName.data = "RWFFld";
dictionaryRequest.dictionaryName.length = 6;

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

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

```

Code Example 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;
RsslRDMMsg rdmMsg;
RsslRDMDictionaryRequest *pDictionaryRequest;

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

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

if (ret == RSSL_RET_SUCCESS && rdmMsg.rdmMsgBase.domainType == RSSL_DMT_DICTIONARY &&
    rdmMsg.rdmMsgBase.rdmMsgType == RDM_DC_MT_REQUEST)
{
    /* The message we decoded is an RsslRDMDictionaryRequest. */
    pDictionaryRequest = &rdmMsg.dictionaryMsg.request;

    /* Print if streaming. */
    if (pDictionaryRequest->flags & RDM_DC_RQF_STREAMING)
        printf("Request is streaming.\n");

    /* Print serviceId. */
    printf("Service ID: %u\n", pDictionaryRequest->serviceId);

    /* Print verbosity. */
    printf("Verbosity: %u\n", pDictionaryRequest->verbosity);

    /* Print dictionary name. */
    printf("Dictionary Name: %.*s\n", pDictionaryRequest->dictionaryName.length, pDictionaryRequest->
        dictionaryName.data);
}

```

Code Example 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. */
rsslClearRDMDictionaryRefresh(&dictionaryRefresh);

/* Set flags. */
dictionaryRefresh.flags = RDM_DC_RFF_SOLICITED;

/* Set dictionary name. */
dictionaryRefresh.dictionaryName.data = "RWFFld";
dictionaryRefresh.dictionaryName.length = 6;

/* Set type. */
dictionaryRefresh.type = RDM_DICTIONARY_FIELD_DEFINITIONS;

/* Set the dictionary. */
dictionaryRefresh.pDictionary = &dataDictionary;

/* Set serviceId. */
dictionaryRefresh.serviceId = 273;

/* Set verbosity. */
dictionaryRefresh.verbosity = RDM_DICTIONARY_NORMAL;

do
{
    /* (Represents the application getting a new buffer to encode the message into.) */
    getNextEncodeBuffer(&msgBuffer);

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

    /* Encode the message. This will return RSSL_RET_DICT_PART_ENCODED if it only a part
    * was encoded. We must keep encoding the message until RSSL_RET_SUCCESS is returned. */
    ret = rsslEncodeRDMMsg(&encodeIter, (RsslRDMMsg*)&dictionaryRefresh, &msgBuffer.length,
        &rsslErrorInfo);
} while (ret == RSSL_RET_DICT_PART_ENCODED);

```

Code Example 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 rdmmMsg;
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, &rdmmMsg, &memoryBuffer, &rsslErrorInfo);

    if (ret == RSSL_RET_SUCCESS && rdmmMsg.rdmMsgBase.domainType == RSSL_DMT_DICTIONARY &&
        rdmmMsg.rdmMsgBase.rdmMsgType == RDM_DC_MT_REFRESH)
    {
        /* The message we decoded is an RsslRDMDictionaryRefresh. */
        pDictionaryRefresh = &rdmmMsg.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 Refinitiv Domain Model 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 Open Message Model 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 6.8.3

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 Open Message Model 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 190: RsslRDMQueueRequest Members

8.6.3 Queue Refresh

A Queue Provider encodes and sends a **Queue Refresh** message to Open Message Model 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 191: Rss1RDMQueueRefresh Members

8.6.4 Queue Status

A Queue Provider encodes and sends **Queue Status** messages to Open Message Model 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 192: Rss1RDMQueueStatus Members

8.6.5 Queue Close

An Open Message Model 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 193: Rss1RDMQueueClose Members

8.6.6 Queue Data

Both Open Message Model 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 Open Message Model 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 Open Message Model 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 RsslRDMQueueTimeCodes 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 RsslRDMQueueDataExpired message. If not specified, this defaults to QueueMsgTimeoutCodes.INFINITE (i.e., the content never times out). |

Table 194: RsslRDMQueueData Members

8.6.6.2 Queue Data Flag

RsslRDMQueueData messages and **RsslRDMQueueDataExpired** 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 195: Queue Data Flag

8.6.6.3 Queue Message Timeout Codes

Queue message timeout codes are special codes that can be set on the `RsslRDMQueueData.timeout` member to specify timeout behavior.

| ENUMERATION | DESCRIPTION |
|------------------------------|--|
| RDM_QMSG_TC_INFINITE | This message persists in the system for an infinite amount of time. |
| RDM_QMSG_TC_IMMEDIATE | This message immediately times out if any portion of its delivery path is unavailable. |
| RDM_QMSG_TC_PROVIDER_DEFAULT | This message persists in the system for a duration set by the provider. |

Table 196: RsslRDMQueueTimeoutCodes

8.6.6.4 Queue Data Encoding

The `RsslRDMQueueData` message allows users to encode both Open Message Model and non-Open Message Model/opaque content. There are several methods available to help with encoding.

| FUNCTION NAME | DESCRIPTION |
|--|--|
| <code>rsslEncodeRDMQueueMsg</code> | When sending no payload or payload content is preencoded and specified on the <code>RsslRDMQueueData.encDataBody</code> buffer, this method encodes the <code>RsslRDMQueueData</code> message in a single call. |
| <code>rsslEncodeRDMQueueMsgComplete</code> | Completes the content encoding into this <code>RsslRDMQueueData</code> message. |
| <code>rsslEncodeRDMQueueMsgInit</code> | <p>Begins the process of encoding content into this <code>RsslRDMQueueData</code> message. This method takes an <code>EncodeIterator</code> as a parameter, where the <code>EncodeIterator</code> is associated with the buffer into which content is encoded.</p> <p>When this method returns, users should call additional methods required to encode the content. After all remaining encoding is completed, call the <code>encodeComplete</code> method.</p> |

Table 197: 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 **RsslRDMQueueDataExpired** messages on the queue stream to Open Message Model consumer applications.

Open Message Model consumer applications do not send this message.

8.6.7.1 RsslRDMQueueDataExpired Structure Members

| MEMBER | DESCRIPTION |
|-------------------|---|
| containerType | Required. Indicates the type of contents in the message. |
| destName | Required. destName specifies the name of the queue from which content is sourced (i.e., the value of sourceName as set in the original RsslRDMQueueData message). |
| encDataBody | Optional. Contains the payload contents (if any) of the original Queue Data message. |
| flags | Required. flags indicate more information about this message. For details, refer to Section 8.6.6.2. |
| identifier | Required. A user-specified, unique identifier for the message (which is the same as the identifier from the original RsslRDMQueueData message). |
| queueDepth | Required. Indicates how many Queue Data or Queue Data Expired messages are still inbound on this queue stream (following this message). |
| rdmMsgBase | Required. Specifies the queue message type (i.e., RDM_QMSG_MT_DATA_EXPIRED) and contains general message information, including the stream's ID (streamId) and domain type (domainType). |
| sourceName | Required. sourceName specifies the name of the queue to which content was sent (i.e., the value of destName as set in the original RsslRDMQueueData message). |
| undeliverableCode | Required. Specifies a code explaining why the content was undeliverable. For more information on undeliverable codes and their meanings, refer to Section 8.6.7.2. |

Table 198: RsslRDMQueueDataExpired Structure Members

8.6.7.2 Queue Message Undeliverable Codes

Undeliverable codes are used in the **QueueDataExpired.undeliverable** member, and specify why the message could not be delivered.

| ENUMERATION | REASON FOR DELIVERY FAILURE |
|----------------------------|---|
| RDM_QMSG_UC_EXPIRED | Indicates that the timeout value specified for this message has expired. |
| RDM_QMSG_UC_INVALID_SENDER | Indicates that the sender of this message has now become invalid. |
| RDM_QMSG_UC_INVALID_TARGET | Indicates that the specified destination of this message does not exist. |
| RDM_QMSG_UC_MAX_MSG_SIZE | Indicates that the message was too large. |
| RDM_QMSG_UC_NO_PERMISSION | Indicates that the source/sender of this message is not permitted to send or is not permitted to send to the specified destination. |
| RDM_QMSG_UC_QUEUE_DISABLED | Indicates that the specified destination of this message has a disabled queue. |

Table 199: RsslRDMQueueDataUndeliverableCodes

| ENUMERATION | REASON FOR DELIVERY FAILURE |
|----------------------------|--|
| RDM_QMSG_UC_QUEUE_FULL | Indicates that the specified destination of this message has a full queue and cannot receive any additional content. |
| RDM_QMSG_UC_TARGET_DELETED | Indicates that the target queue was deleted before the message was delivered. |
| RDM_QMSG_UC_UNSPECIFIED | Indicates that the delivery failed for unspecified reasons. |

Table 199: `Rss1RDMQueueDataUndeliverableCodes` (Continued)

8.6.8 Queue Ack

A Queue Provider encodes and sends a **Queue Ack** message to Open Message Model 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 200: `Rss1RDMQueueAck`

9 Value Added Utilities

9.1 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.

9.2 Memory Buffer

The Memory Buffer utilities provide a simple method to apportion space from a block of memory space. This allows for the creation of complex objects without expensively requesting and releasing memory from the operating system. This also allows for easy reuse of the memory block. The memory is provided to the functions via an **RsslBuffer** that has its **data** member set to the memory block and **length** member indicating the byte length of the memory.

| FUNCTION NAME | DESCRIPTION |
|---|---|
| <code>rsslReserveBufferMemory</code> | Reserves memory from an RsslBuffer . The buffer passed in is modified to point to the unused portion of the memory block. Subsequent calls reserve adjacent memory, so this can be called multiple times to generate a C-style array of objects without knowing the full length in advance. |
| <code>rsslReserveAlignedBufferMemory</code> | Reserves memory from an RsslBuffer . Similar to <code>rsslReserveBufferMemory</code> , but will ensure that the memory is aligned on an appropriate word boundary. Subsequent calls to the non-aligned <code>rsslReserveBufferMemory</code> will reserve adjacent memory, so this can be called multiple times to generate a C-style array of objects without knowing the full length in advance. |
| <code>rsslCopyBufferMemory</code> | Requires an input RsslBuffer , output RsslBuffer , and an RsslBuffer pointing to an available memory block. Sets the output buffer to a deep copy of the input RsslBuffer , using the space provided by the memory block. |

Table 201: Memory Buffer Functions

9.3 Using the Memory Buffer

The following example reserves an aligned block of memory to represent an array of five user-defined **MyStruct** structures. The memory for the first structure is reserved and aligned. Each subsequent member of the array is then reserved in a loop, wherein memory is reserved based on the initial aligned offset. The memory associated with each **MyStruct** is initialized after it is reserved. Once completed, **myStructArray** can be accessed like an array of **MyStructs** (**myStructArray[index]**, etc.).

```
/* Represents some complex user-defined struct.
 * This example will create an array of these structs. */
typedef struct
{
    int number;
    char letter;
} MyStruct;

int i = 0;

/* The block of memory that we will use as storage of the array. */
char memoryBlock[128];
RsslBuffer memoryBuffer;

MyStruct *myStructArray, *myStructElem;

memoryBuffer.data = memoryBlock;
memoryBuffer.length = 128;

/* Create first element on a word boundary, then initialize */
myStructArray = (MyStruct*)rsslReserveAlignedBufferMemory(&memoryBuffer, 1, sizeof(MyStruct));
myStructArray->number = i;
myStructArray->letter = 'a';

for(i = 1; i < 5; ++i)
{
    /* Reserve space for subsequent elements and initialize them in. */
    myStructElem = (MyStruct*)rsslReserveBufferMemory(&memoryBuffer, 1, sizeof(MyStruct));

    myStructElem->number = i;
    myStructElem->letter = 'a';
}

/* Change the letter of MyStruct in position 2, can access just like an array */
myStructArray[2]->letter = 'b';
```

Code Example 33: Memory Buffer Example

10 Payload Cache Detailed View

10.1 Concepts

The Value Added Payload Cache component provides a facility for storing Open Message Model containers (the message's data payload). Typical use of a payload cache is to store the current image of Open Message Model 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 Open Message Model containers.

A cache is defined as a collection of Open Message Model 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 Refinitiv Domain Model 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 Refinitiv Source Sink Library 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 12 shows an example consumer type application which has created two cache instances to store data from two services on an Open Message Model provider.

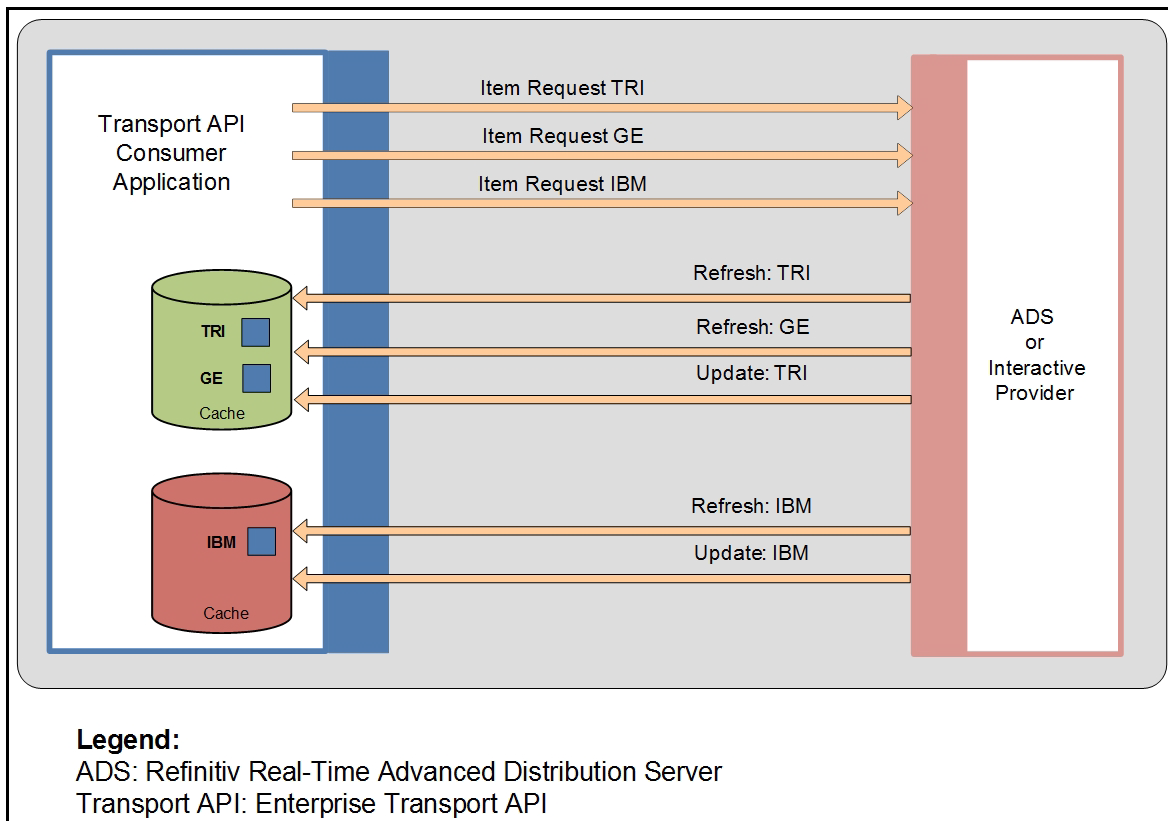


Figure 12. Consumer Application using Cache to Store Payload Data for Item Streams

10.2 Payload Cache

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

10.2.1 Managing the Payload Cache

To use the Value Added Payload Cache, the application must first call the **rsslPayloadCacheInitialize** function for global static resource initialization. When the payload cache is no longer needed, the application should call **rsslPayloadCacheUninitialize** to cleanup and release all resources used by the cache.

Use the following functions to manage the cache:

| FUNCTION NAME | DESCRIPTION |
|-------------------------------|---|
| rsslPayloadCacheInitialize | The first function the application must call prior to using the payload cache. The method only needs to be called one time by the application, but may be called more than once. A reference count is incremented for each call to this function. An equal number of calls to rsslPayloadCacheUninitialize must be made before the component is uninitialized. |
| rsslPayloadCacheUninitialize | The last call an application should make when it is finished using the payload cache interface. The initialization reference count is decremented for each call to this function. Uninitialization only occurs if the initialization reference count is zero. During uninitialization, all remaining cache instances, entries, and resources will be destroyed. |
| rsslPayloadCacheIsInitialized | This function can be used by an application to determine if the payload cache component has already been initialized (by a call to rsslPayloadCacheInitialize). |

Table 202: Payload Cache Management Functions

10.2.2 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.

10.2.2.1 Cache Error Structure Members

The **RsslCacheError** has the following structure members:

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

Table 203: RsslCacheError Structure Members

10.2.2.2 Clearing a Cache Error

The following function clears the **RsslCacheError**.

| STRUCTURE MEMBER | DESCRIPTION |
|----------------------------------|---|
| <code>rsslCacheErrorClear</code> | Clears the RsslCacheError structure. Use this function prior to passing the structure to a cache interface function. |

Table 204: Function for Cache Error Handling

10.2.3 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 10.2.3.1. Before using payload caches, you must first have initialized this function using **rsslPayloadCacheInitialize** as described in Section 10.2.1.

10.2.3.1 Managing Payload Instances

| FUNCTION NAME | DESCRIPTION |
|--------------------------------------|--|
| <code>rsslPayloadCacheCreate</code> | Creates a payload cache instance, and returns the RsslPayloadCacheHandle . All operations on the cache require this handle. Options are passed in via the RsslPayloadCacheConfigOptions defined in Section 10.2.3.2. |
| <code>rsslPayloadCacheDestroy</code> | Destroys a payload cache instance. Any entries remaining in the cache are also destroyed at this time. |

Table 205: Functions for Managing Cache Instances

10.2.3.2 Payload Cache Structure Member

| STRUCTURE MEMBER | DESCRIPTION |
|-----------------------|--|
| <code>maxItems</code> | Sets the maximum number of entries allowed in the cache. When the maximum number of items is reached, the cache refuses new entries until existing entries are removed. The rsslPayloadEntryCreate function will return a null RsslPayloadEntryHandle when the maximum number of items is reached. When set to zero, the cache allows an unlimited number of items. Refer to Section 10.3.1. |

Table 206: RsslPayloadCacheConfigOptions Structure Members

10.2.4 Managing Refinitiv Domain Model Field Dictionaries for Payload Cache

Each cache instance requires an Refinitiv Domain Model Field Dictionary, to define the set of fields that may be encoded in the Open Message Model containers stored in the cache.

A cache is associated with a field dictionary through a binding process, which requires an **RsslDataDictionary** structure loaded with the field dictionary. The dictionary structure can be loaded from a file (using the **rsslLoadFieldDictionary** function) or from an encoded dictionary message from a provider (using the **rsslDecodeFieldDictionary** function). The cache does not use the enumerated dictionary content, so loading the enumeration dictionary is not required. For more information on using **RsslDataDictionary**, refer to the *Enterprise Transport API Reference Manual*.

After the **RsslDataDictionary** loads, it is bound to a cache instance using a key (an arbitrary string identifier assigned by the application to name the dictionary). The key allows multiple cache instances to share the same dictionary. After the first binding of a dictionary, it can be bound to additional cache instances by simply providing the same key on additional bindings. For a list of functions used in binding a dictionary to a cache, refer to Section 10.2.4.1.

The cache builds its own field definition database from the **RsslDataDictionary** definitions. After binding, the application does not need to retain the dictionary structure, because the cache does not refer to the **RsslDataDictionary** used during the binding. In typical usage, the application will likely retain the dictionary for use with other encoding and decoding operations.

NOTE: A cache can be bound to a dictionary only once during its lifetime. While a cache cannot be switched to a new dictionary, the dictionary in use can be extended with new definitions. Refer to Section 10.2.4.3.

10.2.4.1 Setting Functions

| FUNCTION NAME | DESCRIPTION |
|--|--|
| <code>rsslPayloadCacheBindDictionary</code> | Deprecated. For details on using this function, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> . Otherwise, for equivalent functionality, refer to rsslPayloadCacheSetDictionary in this table. |
| <code>rsslPayloadCacheBindSharedDictionaryKey</code> | Deprecated. For details on using this function, refer to the Enterprise Transport API C Edition <i>Developers Guide</i> . Otherwise, for equivalent functionality, refer to rsslPayloadCacheSetSharedDictionaryKey in this table. |
| <code>rsslPayloadCacheSetDictionary</code> | <p>This function sets an RsslDataDictionary to a cache instance (identified by RsslPayloadCacheHandle). Use this function the first time a dictionary is set to a cache. The application must provide a key parameter to this function to name the dictionary for future reference. This key is used in future setting operations when the application wants to share a dictionary between cache instances or to extend the definitions in the dictionary.</p> <p>The first time a particular key is used with this function will be the initial setting of that dictionary to a cache. The second time the same key is used in this function; it will reload the field definitions from the given RsslDataDictionary structure, enabling the dictionary to be extended. Refer to Section 10.2.4.3.</p> |
| <code>rsslPayloadCacheSetSharedDictionaryKey</code> | <p>Use this function when sharing a dictionary among multiple caches. This function sets a cache (identified by the RsslPayloadCacheHandle) to a previously set dictionary (identified by the dictionary key name). To share a dictionary, the dictionary named by the key passed to this function must have previously had an initial setting to another cache using the rsslPayloadCacheSetDictionary function.</p> <p>This function does not require the RsslDataDictionary structure, since that was already loaded during the initial setting with this dictionary key.</p> |

Table 207: Functions for Setting Dictionary to Cache

10.2.4.2 Setting Example

In the following example, two cache instances are created and set to a single shared field dictionary.

```
RsslRet ret;
RsslCacheError cacheError;
RsslPayloadCacheConfigOptions cacheConfig;
const char* dictionaryKey = "SharedKey1";
char errorDataArray[256];
RsslBuffer errorBuffer = {256, &errorDataArray[0]};
RsslPayloadCacheHandle cacheHandle1 = 0;
RsslPayloadCacheHandle cacheHandle2 = 0;
RsslDataDictionary dataDictionary;

/* For simplicity in this code fragment, CHK is assumed to be a macro for error handling
   (performing cleanup and returning from function). */

/* Initialize cache component and create cache instances */
ret = rsslPayloadCacheInitialize(); CHK(ret)
cacheConfig.maxItems = 0; /* unlimited */
cacheHandle1 = rsslPayloadCacheCreate(&cacheConfig, &cacheError);
if (cacheHandle1 == 0)
{
    printf("rsslPayloadCacheCreate failure: %s\n", cacheError.text);
    CHK(cacheError.rsslErrorId)
}
cacheHandle2 = rsslPayloadCacheCreate(&cacheConfig, &cacheError);
if (cacheHandle2 == 0)
{
    printf("rsslPayloadCacheCreate failure: %s\n", cacheError.text);
    CHK(cacheError.rsslErrorId)
}

/* Load an RDM Field Dictionary structure from file: set to each cache. */
rsslClearDataDictionary(&dataDictionary);
ret = rsslLoadFieldDictionary("RDMFieldDictionary", &dataDictionary, &errorBuffer); CHK(ret)
/* Initial setting of the dictionary to the first cache */
ret = rsslPayloadCacheSetDictionary(cacheHandle1, &dataDictionary, dictionaryKey, &cacheError);
    CHK(ret)
/* Shared setting of the same dictionary to the second cache */
ret = rsslPayloadCacheSetSharedDictionaryKey(cacheHandle2, dictionaryKey, &cacheError); CHK(ret)
/* The dataDictionary can be destroyed after setting, but is typically retained by the application
   for encoding and decoding. */

/* Two cache instances are now ready for applying and retrieving data */
/* ... */

/* Cleanup */
rsslPayloadCacheDestroy(cacheHandle1); /* destroys all entries and the cache instance */
rsslPayloadCacheDestroy(cacheHandle2);
```

```

/* After all cache instances bound to a dictionary are destroyed, the cache API will clean up the
   internal field dictionary database used by the cache. */
rsslPayloadCacheUninitialize(); /* final call to cache interface */
rsslDeleteDataDictionary(&dataDictionary);

```

Code Example 34: Creating Cache and Setting to Dictionary

10.2.4.3 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.

10.2.5 Payload Cache Utilities

Use the following functions for managing cache instances. These utilities provide a count of the cache entries and a list of handles to each cache entry.

| FUNCTION NAME | DESCRIPTION |
|-------------------------------|---|
| rsslPayloadCacheGetEntryCount | Returns the number of item payload entries in this cache instance (RsslPayloadCacheHandle). |
| rsslPayloadCacheGetEntryList | Populates an array provided by the caller with entry handles (RsslPayloadEntryHandle) for this cache instance (RsslPayloadCacheHandle). Because each cache entry is likely associated with an entry in the application's item list, an application would typically manage the set of entry handles. This utility provides access to the entire entry handle list if needed. |
| rsslPayloadCacheClearAll | Destroys all entries in the cache instance (RsslPayloadCacheHandle). The empty cache can be reused and remains bound to its data dictionary. |

Table 208: Payload Cache Utility Functions

10.3 Payload Cache Entries

A payload cache entry stores a single Open Message Model container (whose data types are defined by **RsslContainerType**). While a cache entry can store any arbitrary Open Message Model 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 Open Message Model data and sets the current value by following the Open Message Model 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 Open Message Model container.

10.3.1 Managing Payload Cache Entries

Payload cache entries are created within a cache instance. A cache entry is defined by the **RsslPayloadEntryHandle** returned from the **rsslPayloadEntryCreate** function. You cannot move entries between different cache instances, due to their dependency on the field dictionary bound to the cache where they are created.

Cache entries only store the payload container of an item. Maintain other item data (e.g. message key attributes, domain, state) as needed in an item list managed by the application, which will identify the source or sink associated with the cache entry data. This item list will likely include the **RsslPayloadEntryHandle** if the payload of the item is cached.

For a list of basic utilities provided by the payload cache to manage the collection of entries in the cache, refer to Section 10.2.5.

Use the following functions to manage cache entries:

| FUNCTION NAME | DESCRIPTION |
|-------------------------|--|
| rsslPayloadEntryCreate | This method returns a RsslPayloadEntryHandle to the newly created entry in the cache defined by the given RsslPayloadCacheHandle . The RsslPayloadEntryHandle is required for all operations on this entry. This function will return a null handle if it cannot create the entry (e.g., if the maximum number of entries as defined in RsslPayloadCacheConfigOptions would be exceeded). |
| rsslPayloadEntryDestroy | This method destroys the cache entry defined by RsslPayloadEntryHandle and removes it from its cache. |
| rsslPayloadEntryClear | This method deletes any data in the cache entry RsslPayloadEntryHandle and returns the entry to its initial state. The entry itself remains in the cache and can be re-used. |

Table 209: Payload Cache Entry Management Functions

10.3.2 Applying Data

Data is applied to a cache entry from the payload of an Open Message Model 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 Open Message Model data so that the cache entry data can be set or updated.

Some caching behaviors are controlled by flags in the **RsslMsg**. When an **RsslRefreshMsg** is applied to the cache entry, the following **RsslRefreshFlags** take effect:

- **RSSL_RFMF_CLEAR_CACHE**: Cache entry data will be cleared prior to applying this message.
- **RSSL_RFMF_DO_NOT_CACHE**: The payload will not be applied to the cache entry.

When an **RsslUpdateMsg** is applied to cache, the following **RsslUpdateFlags** take effect:

- **RSSL_UPMF_DO_NOT_CACHE**: The payload data will not be applied to the cache entry.
- **RSSL_UPMF_DO_NOT_RIPPLE**: When applying the data, entry rippling is not performed.

The following example demonstrates how to create a payload entry in a cache instance and apply the payload of an **RsslMsg** to the cache entry.

```
/* Apply buffer containing an encoded RsslMsg to cache entry */
RsslRet applyBufferToCache(RsslChannel *pChannel, RsslBuffer *pBuffer, RsslPayloadCacheHandle
    cacheHandle, RsslPayloadEntryHandle *pEntryHandle)
{
    RsslDecodeIterator dIter;
    RsslMsg msg;
    RsslRet ret;
    RsslCacheError cacheError;

    /* If the caller did not provide a cache entry handle, create a new entry */
    if (*pEntryHandle == 0)
    {
        rsslCacheErrorClear(&cacheError);
        *pEntryHandle = rsslPayloadEntryCreate(cacheHandle, &cacheError);
        if (*pEntryHandle == 0)
        {
            printf("Error (%d) creating cache entry: %s\n", cacheError.rsslErrorId, cacheError.text);
            return cacheError.rsslErrorId;
        }
    }

    /* Perform message decoding. */
    rsslClearDecodeIterator(&dIter);
    rsslSetDecodeIteratorRwfVersion(&dIter, pChannel->majorVersion, pChannel->minorVersion);
    rsslSetDecodeIteratorBuffer(&dIter, pBuffer);
    rsslClearMsg(&msg);
    ret = rsslDecodeMsg(&dIter, &msg);
    if (ret < RSSL_RET_SUCCESS)
    {
        printf("Failure (%d) decoding message from buffer\n");
        return ret;
    }
}
```

```

/* Apply the decoded RsslMsg to cache, with iterator positioned at the start of the payload */
rsslCacheErrorClear(&cacheError);
ret = rsslPayloadEntryApply(*pEntryHandle, &dIter, &msg, &cacheError);
if (ret < RSSL_RET_SUCCESS)
{
    printf("Error (%d) applying data to cache entry: %s\n", cacheError.rsslErrorId,
        cacheError.text);
    return ret;
}

return ret;
}

```

Code Example 35: Applying Data to a Payload Cache Entry

10.3.3 Retrieving Data

Data is retrieved from a cache entry as an encoded Open Message Model 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 Open Message Model 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 |
|---------------------------------|---|
| rsslPayloadEntryGetType | Returns the RsslContainerType stored in the cache entry (RsslPayloadEntryHandle). When initially created (or after the entry is cleared), the data type will be RSSL_DT_UNKNOWN . The data type is defined by the container type of the first refresh message applied to the entry. |
| rsslPayloadEntryApply | Applies the Open Message Model data in the payload of the RsslMsg to the cache entry (RsslPayloadEntryHandle). The first message applied must be a refresh message (class RSSL_MC_REFRESH). |
| rsslPayloadEntryRetrieve | Retrieves data from the cache entry by encoding the Open Message Model container into the buffer provided with the RsslEncodeIterator given by the application. For single-part retrieval, the RsslPayloadCursorHandle parameter is optional. For details on multi-part retrieval, refer to Section 10.3.3.1. |

Table 210: Functions for Applying and Retrieving Cache Entry Data

10.3.3.1 Multi-Part Retrieval

For data types that support fragmentation, the container can be retrieved in multiple parts by calling **rsslPayloadEntryRetrieve** until the complete container is returned. To support multi-part retrieval, the optional **RsslPayloadCursorHandle** parameter is required when calling **rsslPayloadEntryRetrieve**. The cursor is used to maintain the position where the next retrieval will resume. The application must check the state of the cursor after each call to **rsslPayloadEntryRetrieve** to determine when the retrieval is complete. The following functions are needed when using the payload cursor.

| FUNCTION NAME | DESCRIPTION |
|------------------------------------|--|
| rsslPayloadCursorCreate | Creates a cursor for optional use in the rsslPayloadEntryRetrieve function (required for multi-part retrieval). Returns the RsslPayloadCursorHandle . |
| rsslPayloadCursorDestroy | Destroys the cursor referenced by the RsslPayloadCursorHandle . |
| rsslPayloadCursorClear | Clears the state of the cursor for the given RsslPayloadCursorHandle . Whenever retrieving data from a cache entry, the cursor must be cleared prior to the first call to rsslPayloadEntryRetrieve . Clearing the cursor also allows it to be reused with a retrieval on a different container. |
| rsslPayloadCursorIsComplete | Returns the completion state of a retrieval where the RsslPayloadCursorHandle was used. The state must be checked after each call to rsslPayloadEntryRetrieve to determine whether additional data needs to be encoded for the cache entry container. When the cursor state is complete, the entire container of the cache entry has been retrieved. |

Table 211: Functions for Using the Payload Cursor

10.3.3.2 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 Open Message Model 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 Open Message Model 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.

10.3.3.3 Example: Cache Retrieval with Multi-Part Support

The following example illustrates data retrieval from a cache entry, which supports multi-part encoding of a container.

```
/* Code fragment showing use of rsslPayloadEntryRetrieve for multi-part retrieval. */

RsslRet ret;
RsslCacheError cacheError;
RsslBuffer buffer;
RsslEncodeIterator eIter;
int arraySize = DEFAULT_BUFFER_SIZE;
unsigned char* bufferArray = (char*) malloc(arraySize);
buffer.data = bufferArray;
buffer.length = arraySize;
RsslPayloadCursorHandle cursorHandle = rsslPayloadCursorCreate();
rsslPayloadCursorClear(cursorHandle);
while (!rsslPayloadCursorIsComplete(cursorHandle))
{
    buffer.length = arraySize;
    rsslClearEncodeIterator(&eIter);
    rsslSetEncodeIteratorBuffer(&eIter, &buffer);
    rsslCacheErrorClear(&cacheError);
    /* _entryHandle created outside the scope of this code fragment */
    ret = rsslPayloadEntryRetrieve(_entryHandle, &eIter, cursorHandle, &cacheError);
    if (ret == RSSL_RET_SUCCESS)
        /* Number of bytes encoded is buffer.length. Application can use encoded data, e.g. set the
           payload on RsslMsgBase.encDataBody and encode a message to be transmitted. */
    else if (ret == RSSL_RET_BUFFER_TOO_SMALL)
        /* Increase arraySize and reallocate bufferArray. */
    else
        /* Handle terminal error condition. See cacheError.text[] for additional information. */
}
rsslPayloadCursorDestroy(cursorHandle);
free(bufferArray);
```

Code Example 36: Cache Retrieval with Multi-Part Support

Z

© 2015 - 2020 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: ETAC351UMVAC.200
Date of issue: September 2020

