

Diffusion 5.9 User Guide

Contents

List of Figures.....	13
List of Tables.....	18
Part I: Welcome.....	22
Introducing Diffusion.....	23
What's new in Diffusion 5.9?.....	25
What was new in Diffusion 5.8?.....	27
What was new in Diffusion 5.7?.....	27
What was new in Diffusion 5.6?.....	29
What was new in Diffusion 5.5?.....	30
What was new in Diffusion 5.1?.....	32
What was new in Diffusion 5.0?.....	34
Part II: Quick Start Guide.....	36
Get Diffusion.....	37
Install Diffusion.....	37
Start the Diffusion server.....	37
Default configuration.....	37
The Diffusion monitoring console.....	38
Develop a publishing client.....	38
Develop a subscribing client.....	40
Resources.....	43
Part III: Design Guide.....	44
Support.....	45
System requirements for the Diffusion server.....	45
Platform support for the Diffusion Unified API libraries.....	47
Feature support in the Diffusion Unified API.....	49
Protocol support.....	54
Browser support.....	56
Browser limitations.....	57
WebSocket limitations.....	57
Cross-origin resource sharing limitations.....	57
Browser connection limitations.....	58

Designing your data model.....	59
Topic tree.....	59
Topic naming.....	61
Topic selectors in the Unified API.....	61
Topic selectors in the Classic API (deprecated).....	69
Topics.....	71
JSON topics.....	71
Binary topics.....	73
Single value topics.....	74
Record topics.....	75
Stateless topics.....	77
Advanced topics.....	78
Publication.....	89
Publishing data.....	90
Subscribing to topics.....	91
Messaging.....	92
Advanced usage.....	93
Conflation.....	93
DEPRECATED: Distributing and viewing data as pages.....	96
Designing your solution.....	97
Servers.....	98
Fan-out.....	99
Using missing topic notifications with fan-out.....	102
High availability.....	104
Session replication.....	105
Topic replication.....	108
Failover of active update sources.....	110
Clients.....	111
Client types.....	111
Using clients.....	113
Using clients for control.....	114
User-written components.....	116
Publishers.....	116
Other user-written components.....	117
Third party components.....	119
Load balancers.....	119
Web servers.....	120
Push notification networks.....	123
JMS.....	124
Example solutions.....	126
Example: Simple solution.....	127
Example: A solution using clients.....	128
Example: Scalable and resilient solution.....	129
Security.....	129
Role-based authorization.....	130
Permissions.....	134
Pre-defined roles.....	138
Authentication.....	140
User-written authentication handlers.....	143
System authentication handler.....	145
Pre-defined users.....	146
DEPRECATED: Authorization handlers.....	147
Securing the console.....	150

Part IV: Developer Guide.....	152
The Diffusion SDKs.....	153
JavaScript.....	153
Apple.....	156
Android.....	158
Java.....	160
.NET.....	162
C.....	164
Feature support in the Diffusion Unified API.....	169
Best practice for developing clients.....	173
Getting started.....	174
Start subscribing with JavaScript.....	174
Start subscribing with iOS.....	178
Start subscribing with Android.....	183
Start subscribing with Java.....	188
Start subscribing with .NET.....	192
Start subscribing with C.....	197
Start publishing with JavaScript.....	202
Start publishing with OS X/macOS.....	203
Start publishing with Android.....	210
Start publishing with Java.....	215
Start publishing with .NET.....	220
Start publishing with C.....	227
Connecting to the Diffusion server.....	237
Connecting basics.....	239
Connecting securely.....	245
Connect to the Diffusion server with a security principal and credentials.....	247
Connecting through an HTTP proxy.....	250
Connecting through a load balancer.....	251
Reconnect to the Diffusion server.....	252
Detecting connection problems.....	254
Specifying a reconnection strategy.....	254
Session failover.....	264
Ping the Diffusion server.....	265
Change the security principal and credentials associated with your client session.....	266
Session properties.....	267
Session filtering.....	268
Receiving data from topics.....	271
Example: Subscribe to a topic.....	272
Example: Subscribe to a JSON topic.....	283
Example: Fetch topic state.....	288
Managing topics.....	295
Example: Create a topic.....	297
Creating a metadata definition for a record topic.....	314
Handling subscriptions to missing topics.....	317
Example: Receive missing topic notifications.....	318
Listening for topic events.....	329
Removing topics with sessions.....	330
Updating topics.....	331
Example: Make exclusive updates to a topic.....	334
Example: Make non-exclusive updates to a topic.....	349
Managing subscriptions.....	354
Example: Subscribe other clients to topics.....	356

Example: Receive notifications when a client subscribes to a routing topic.....	361
Messaging to topic paths.....	363
Example: Send a message to a topic path.....	364
Example: Send a request message to the Push Notification Bridge.....	372
Messaging to sessions.....	375
Example: Handle messages and send messages to sessions.....	377
Example: Use session property filters with messaging.....	388
Authenticating clients.....	395
Example: Register an authentication handler.....	395
Developing a control authentication handler.....	402
Developing a composite control authentication handler.....	405
Updating the system authentication store.....	407
DSL syntax: system authentication store.....	408
Example: Update the system authentication store.....	411
Updating the security store.....	420
DSL syntax: security store.....	420
Example: Update the security store.....	423
Managing clients.....	431
Handling client queues.....	436
Flow control.....	437
Logging from the client.....	438
Logging in JavaScript.....	438
Logging in Apple.....	439
Logging in Android.....	440
Logging in Java.....	440
Logging in .NET.....	442
Logging in C.....	443
DEPRECATED: Classic API.....	443
DEPRECATED: Java Client Classic API.....	443
DEPRECATED: .NET Classic API.....	446
.NET Client Classic API.....	446
DEPRECATED: JavaScript Classic API.....	449
Using the JavaScript Classic API.....	449
DEPRECATED: ActionScript Classic API.....	456
Using the ActionScript Classic API.....	456
Reconnecting with the ActionScript Classic API.....	459
Logging Flash.....	461
DEPRECATED: Silverlight Classic API.....	462
Using the Silverlight Classic API.....	462
DEPRECATED: iOS Classic API.....	466
Getting started with iOS Classic API.....	467
Using the iOS Classic API.....	470
iOS Classic API examples.....	473
DEPRECATED: Android Classic API.....	473
Getting started with Android Classic API.....	473
Using the Android Classic API.....	475
Android Classic API examples.....	477
DEPRECATED: C Classic API.....	479
Using the C Classic API.....	479
diffusion-wrapper.js.....	481
How to use Diffusion wrapper.....	481
Developing a publisher.....	483
Publisher basics.....	483
Defining publishers.....	484
Loading publisher code.....	484

Load publishers by using the API.....	485
Starting and stopping publishers.....	485
Publisher topics.....	486
Receiving and maintaining data.....	488
Publishing and sending messages.....	488
Publisher notifications.....	489
Client handling.....	490
Publisher properties.....	491
Using concurrent threads.....	491
Publisher logging.....	491
DEPRECATED: Server connections.....	491
General utilities.....	492
Writing a publisher.....	492
Creating a Publisher class.....	492
Publisher startup.....	493
Data state.....	493
Data inputs.....	494
Handling client subscriptions.....	495
Publishing messages.....	496
DEPRECATED: Topic locking.....	499
Handling clients.....	499
Publisher closedown.....	500
Testing a publisher.....	500
Client queues.....	501
Queue enquiries.....	501
Maximum queue depth.....	501
Queue notification thresholds.....	502
Tidy on unsubscribe.....	502
Client Geo and Whols information.....	502
The Diffusion Whols service.....	504
Client groups.....	505
Client notifications.....	506
Adding a ClientListener.....	507
Using DefaultClientListener.....	507
Developing other components.....	508
Local authentication handlers.....	508
Developing a local authentication handler.....	508
Developing a composite authentication handler.....	510
Push Notification Bridge persistence plugin.....	511
Using Maven to build Java Diffusion applications.....	513
Build client applications.....	514
Build publishers with Maven.....	514
Building a publisher with mvndar.....	516
Build server application code with Maven.....	518
Testing.....	519
DEPRECATED: Flex/Flash client.....	519
DEPRECATED: Java client test tool.....	523
DEPRECATED: JavaScript client test tool.....	527
Silverlight client test tool.....	528
Stress test tuning.....	531
Stress test.....	532
Benchmarking suite.....	533
Test tools.....	533

Part V: Administrator Guide.....	534
Installing the Diffusion server.....	535
System requirements for the Diffusion server.....	535
Installing the Diffusion server using the graphical installer.....	537
Installing the Diffusion server using the headless installer.....	539
Installing the Diffusion server using Red Hat Package Manager.....	540
Installing the Diffusion server using Docker.....	541
Next steps with Docker.....	542
The Diffusion license.....	543
License restrictions.....	544
Updating your license file.....	545
Installed files.....	546
Verifying the Diffusion installation.....	548
Configuring your Diffusion server.....	550
XML configuration.....	550
Programmatic configuration.....	553
Using the configuration API.....	553
Configuring the Diffusion server.....	555
Configuring fan-out.....	555
Configuring conflation.....	557
Configuring authentication handlers.....	561
Configuring performance.....	563
Server.xml.....	563
Configuring connectors.....	581
Connectors.xml.....	583
Configuring user security.....	588
Security.store.....	589
SystemAuthentication.store.....	591
Configuring logging on the Diffusion server.....	594
Configuring default logging.....	595
Logs.xml.....	595
Configuring log4j2.....	598
Log4j2.xml.....	599
Logging using another SLF4J implementation.....	600
Configuring JMX.....	601
Configuring the Diffusion JMX connector server.....	601
Configuring a remote JMX server connector.....	602
Configuring a local JMX connector server.....	603
Management.xml.....	604
Configuring the JMX adapter.....	605
Publishers.xml.....	606
Configuring replication.....	611
Configuring the Diffusion server to use replication.....	611
Configuring your datagrid provider.....	612
Replication.xml.....	614
Configuring the Diffusion web server.....	616
Configuring Diffusion web server security.....	617
WebServer.xml.....	617
Aliases.xml.....	624
ConnectionValidationPolicy.xml.....	624
Env.xml.....	626
Mime.xml.....	626
Publishers.xml.....	627

Statistics.xml.....	631
SubscriptionValidationPolicy.xml.....	634
Additional XML files.....	636
Starting the Diffusion server.....	636
Running from within a Java application.....	637
Deploying publishers on your Diffusion server.....	640
Classic deployment.....	640
Hot deployment.....	641
Deployment methods.....	641
Load balancers.....	642
Routing strategies at your load balancer.....	643
Monitoring available Diffusion servers from your load balancer.....	645
Compositing URL spaces using your load balancer.....	645
Secure Sockets Layer (SSL) offloading at your load balancer.....	646
Using load balancers for resilience.....	647
Common issues when using a load balancer.....	647
Web servers.....	648
Diffusion web server.....	649
Server-side processing.....	650
Hosting a status page on the Diffusion web server.....	651
Hosting Diffusion web clients in a third-party web server.....	651
Running the Diffusion server inside of a third-party web application server.....	652
Example: Deploying the Diffusion server within Tomcat.....	652
Other considerations when running the Diffusion server inside of a third-party web application server.....	655
Cross domain policies.....	656
Flash security model.....	656
Silverlight security model.....	657
JavaScript security model.....	658
Push Notification Bridge.....	660
Configuring your Push Notification Bridge.....	663
PushNotifications.xml.....	665
Getting an Apple certificate for the Push Notification Bridge.....	669
Getting a Google API key for the Push Notification Bridge.....	669
Running the Push Notification Bridge.....	670
JSON formats used by the Push Notification Bridge.....	670
Request and response JSON formats.....	671
Push notification JSON format.....	674
JMS adapter.....	677
Transforming JMS messages into Diffusion messages or updates.....	678
Publishing using the JMS adapter.....	681
Sending messages using the JMS adapter.....	682
Using JMS request-response services with the JMS adapter.....	685
Configuring the JMS adapter.....	686
Example: Configuring the Diffusion connection for the JMS adapter running as a standalone client.....	689
Example: Configuring JMS providers for the JMS adapter.....	689
Example: Configuring topics for use with the JMS adapter.....	691
Example: Configuring pub-sub with the JMS adapter.....	692
Example: Configuring messaging with the JMS adapter.....	693
Example: Configuring the JMS adapter to work with JMS services.....	694
JMSAdapter.xml.....	695
Running the JMS adapter.....	705
DEPRECATED: Legacy JMS adapter.....	706
DEPRECATED: Configuring the legacy JMS adapter version 5.1.....	708

DEPRECATED: JMS adapter data flow examples.....	716
Network security.....	724
Going to production.....	726
Pre-production testing.....	726
Setting up your test environment.....	726
Understanding production usage conditions.....	728
Types of testing.....	730
Testing your security.....	731
Tools you can use in your pre-production testing.....	732
Planning for production.....	733
Deploying to your production environment.....	734
Managing and monitoring your running Diffusion server.....	734
JMX.....	735
Using Java VisualVM.....	736
Using JConsole.....	738
MBeans.....	741
The JMX adapter.....	751
Statistics.....	754
Configuring statistics.....	757
Diffusion monitoring console.....	759
DEPRECATED: Introspector.....	770
Supported platforms.....	771
Installing from update site.....	771
Installing subsequent plugin updates.....	775
Uninstalling.....	775
Opening the Diffusion perspective.....	776
Adding servers.....	778
Opening servers.....	779
Exploring the topics.....	779
Getting topic values.....	779
Configuring columns.....	781
Ping servers.....	781
Count topics.....	781
Using the clients view.....	781
Ping.....	782
Statistics.....	783
Topics.....	783
Logging.....	783
Server logs.....	783
Property obfuscator.....	784
Logging.....	784
Logging back-end.....	785
Logging reference.....	786
Log messages.....	790
Connection counts.....	1049
Integration with Splunk.....	1049
Tuning.....	1052
Concurrency.....	1053
Buffer sizing.....	1055
Message sizing.....	1056
Client queues.....	1057
Client multiplexers.....	1057
Connectors.....	1058
Thread pools.....	1059
Client reconnection.....	1062

Client failover.....	1065
Client throttling.....	1067
Java memory usage.....	1068
Platform-specific issues.....	1068
Socket issues.....	1068
Publisher design.....	1070
Demos.....	1071
Demos.....	1071
Building the demos using mvndar.....	1072
Tools.....	1072
Tools for Amazon Elastic Compute Cloud (EC2).....	1072
Tools for Joyent.....	1074

Part VI: Upgrading Guide..... 1075

Interoperability.....	1076
Upgrading from version 4.x to version 5.1.....	1079
Upgrading from version 5.1 to version 5.5.....	1085
Upgrading from version 5.5 to version 5.6.....	1091
Upgrading from version 5.6 to version 5.7.....	1095
Upgrading from version 5.7 to version 5.8.....	1098
Upgrading from version 5.8 to version 5.9.....	1102
Upgrading to a new patch release.....	1105
Known issues in Diffusion 5.9.....	1106

Chapter : Appendices..... 1108

Appendix A: Document conventions..... 1109

Appendix B: Glossary..... 1110

A.....	1111
C.....	1112
D.....	1113
E.....	1115
F.....	1115
G.....	1116
H.....	1116
I.....	1117
J.....	1117
L.....	1119
M.....	1119
N.....	1120
P.....	1120
Q.....	1122
R.....	1122
S.....	1124
T.....	1126
U.....	1127
V.....	1128
W.....	1128
X.....	1128

Appendix C: Trademarks..... 1130

Appendix D: Copyright Notices..... 1132

ANTLR.....	1134
apns.....	1134
Apache Commons Codec.....	1134
Apache Portable Runtime.....	1134
Bootstrap.....	1135
CocoaAsyncSocket.....	1135
concurrent-trees.....	1135
CQEngine.....	1135
cron4j.....	1135
d3.....	1136
disruptor.....	1136
FastColoredTextBox.....	1136
Fluent validation.....	1136
Fluidbox.....	1136
gcm-server.....	1137
GeoIP API.....	1137
GeoLite City Database.....	1137
geronimo-jms_1.1_spec.....	1137
Google code prettify.....	1138
hashmap.....	1138
Hazelcast.....	1138
HPPC.....	1139
htmlcompressor.....	1139
inherits.....	1139
jackson-core.....	1139
jackson-dataformat-cbor.....	1139
JCIP Annotations.....	1140
JCTools.....	1140
jQuery.....	1140
json-simple.....	1140
JZlib.....	1140
Knockout.....	1141
libwebsockets.....	1141
log4j2.....	1141
loglevel.....	1141
long.....	1141
Metrics.....	1142
Minimal JSON.....	1142
Modernizr.....	1142
NLog.....	1142
opencsv.....	1143
OpenSSL.....	1143
PCRE.....	1143
Picocontainer.....	1144
Protocol Buffers.....	1144
Rickshaw.....	1144
Servlet API.....	1144
SLF4J.....	1144
slf4j-android-logger.....	1144

SocketRocket.....	1145
Tabber.....	1145
Tapestry (Plastic).....	1145
TrueLicense.....	1145
when.....	1146
ws.....	1146
Licenses.....	1146
Apache License 2.0.....	1146
BSD 3-clause License.....	1149
Common Development and Distribution License.....	1150
Eclipse Public License – v 1.0.....	1154
ISC License –.....	1157
The GNU Lesser General Public License, version 2.1 (LGPL-2.1).....	1157
The GNU Lesser General Public License, version 3.0 (LGPL-3.0).....	1163
The MIT License (MIT).....	1165
OpenSSL and SSLeay Licenses.....	1166

List of Figures

Figure 1: Example topic tree.....	60
Figure 2: Pub-sub model.....	89
Figure 3: A client registers a handler on part of the topic tree.....	92
Figure 4: A client can send messages through a topic path to known client sessions..	92
Figure 5: Message flow without conflation enabled.....	94
Figure 6: Message flow with simple replace conflation enabled.....	94
Figure 7: Message flow with simple append conflation enabled.....	95
Figure 8: Message flow with merge and replace conflation enabled.....	95
Figure 9: Fan-out.....	100
Figure 10: Missing topic notification propagation.....	103
Figure 11: Information sharing using a datagrid.....	104
Figure 12: Session replication.....	105
Figure 13: Topic replication.....	108
Figure 14: Using a web server with Diffusion.....	121
Figure 15: Deploying Diffusion inside a web application server.....	122
Figure 16: A simple solution.....	127
Figure 17: Clients for different purposes.....	128
Figure 18: Architecture using replication and fan-out.....	129
Figure 19: Topic scope example.....	136

Figure 20: Authentication process for clients.....	141
Figure 21: A composite authentication handler.....	144
Figure 22: Session state model.....	238
Figure 23: Flow of requests and responses when connecting to Diffusion through a proxy.....	250
Figure 24: Flow from a subscribing client to the client that handles a missing topic subscription.....	318
Figure 25: Diffusion wrapper.....	481
Figure 26: The message queue.....	501
Figure 27: Example folder structure inside a DAR file.....	515
Figure 28: Flex client: Connection tab.....	520
Figure 29: Flex client: Send tab.....	521
Figure 30: Flex client: Messages tab.....	522
Figure 31: Flex client: Log tab.....	523
Figure 32: External client tester: Connection tab.....	524
Figure 33: External client tester: Send tab.....	525
Figure 34: External client tester: Messages tab.....	526
Figure 35: External client tester: Message details window.....	527
Figure 36: JavaScript test tool.....	528
Figure 37: Silverlight test tool: Connection tab.....	529
Figure 38: Silverlight test tool: Send tab.....	530
Figure 39: Silverlight test tool: Messages tab.....	531
Figure 40: Sticky-IP in F5 BIG-IP.....	644
Figure 41: Using a load balancer to composite two URL spaces into one.....	659
Figure 42: Requests to the Push Notification Bridge.....	661
Figure 43: Notifications from the Push Notification Bridge.....	662
Figure 44: JMS message structure.....	678
Figure 45: Basic mapping from a JMS message to a Diffusion message.....	679

Figure 46: Basic mapping from a Diffusion message to a JMS message.....	679
Figure 47: Mapping from a JMS message to and from JSON in a Diffusion message..	680
Figure 48: JMS adapter: Publishing from JMS to Diffusion.....	681
Figure 49: JMS adapter: Message flow from Diffusion to JMS.....	683
Figure 50: JMS adapter: Message flow from JMS to Diffusion.....	683
Figure 51: JMS adapter: Request-response message flow.....	685
Figure 52: Subscription flow.....	718
Figure 53: Sending flow from a Diffusion client to a JMS topic (or queue).....	719
Figure 54: Request-reply initiated by a JMS client and serviced by a Diffusion client.	721
Figure 55: Request-reply initiated by a Diffusion client and serviced by a JMS client.	723
Figure 56: Connecting to Diffusion JMX.....	735
Figure 57: Java VisualVM: Overview tab.....	737
Figure 58: JConsole New Connection dialog: Remote Process.....	738
Figure 59: JConsole New Connection dialog: Remote Process.....	739
Figure 60: JConsole New Connection dialog: Local Process.....	740
Figure 61: The server MBean stopController operation showing in JConsole.....	742
Figure 62: Reflecting MBeans as topics.....	752
Figure 63: Showing a composite attribute as a topic nest.....	753
Figure 64: Topics reflecting an ArrayType MBean attributes.....	754
Figure 65: Logging in the monitoring console.....	760
Figure 66: The default console layout.....	760
Figure 67: The table of publishers.....	761
Figure 68: Publisher statistics graphs.....	762
Figure 69: The table of topics.....	762
Figure 70: Details of the topic publishing the CPU load of the host server.....	763
Figure 71: The table of clients.....	763
Figure 72: The table of log entries.....	764

Figure 73: Security tables.....	765
Figure 74: Editing the Access Policy.....	766
Figure 75: Notification that the Diffusion server has stopped.....	766
Figure 76: The default Diffusion Details panel.....	767
Figure 77: Editing the properties of the Diffusion Details panel.....	768
Figure 78: Visualizing the CPU load on a server at a specific time.....	769
Figure 79: Editing and adding to the set of topics for this panel.....	769
Figure 80: Adding a repository.....	772
Figure 81: Install dialog.....	772
Figure 82: Accept the license agreement.....	773
Figure 83: Click OK.....	774
Figure 84: Restarting.....	775
Figure 85: About Eclipse dialog.....	775
Figure 86: Installed plugins.....	776
Figure 87: Perspective.....	777
Figure 88: Views.....	777
Figure 89: Add a server.....	778
Figure 90: Edit server details.....	779
Figure 91: View topic values.....	780
Figure 92: Re-order columns.....	781
Figure 93: Ping a server.....	781
Figure 94: Topic count.....	781
Figure 95: Ping clients.....	782
Figure 96: Server log entries.....	783
Figure 97: Property Obfuscator dialog.....	784
Figure 98: Welcome tab of the Splunk web UI.....	1050
Figure 99: The Splunk Set source type dialog.....	1051

Figure 100: The Data Preview panel..... 1051

Figure 101: The Splunk search summary panel..... 1052

Figure 102: Reconnection scenario..... 1064

Figure 103: Normal and throttled client queues..... 1067

List of Tables

Table 1: Supported platforms and transport protocols for the client libraries.....	47
Table 2: Capabilities provided by the Diffusion client libraries.....	49
Table 3: Supported protocols by client.....	54
Table 4: Supported protocols by client.....	55
Table 5: Supported browsers.....	56
Table 6: Support for WebSocket.....	57
Table 7: Support for CORS.....	57
Table 8: Maximum supported connections.....	58
Table 9: Restricted characters for topics used by Classic API clients.....	61
Table 10: Types of topic selector.....	62
Table 11: Descendant pattern qualifiers.....	63
Table 12: Selector examples.....	70
Table 13: Data types for metadata fields.....	76
Table 14: Handling Responses.....	87
Table 15: Handling Errors.....	87
Table 16: Error types.....	88
Table 17: Supported protocols by client.....	113
Table 18: List of topic-scoped permissions.....	134
Table 19: List of global permissions.....	137

Table 20: Client operations that require authentication.....	142
Table 21: Types of authentication handler.....	144
Table 22: Authorization handler methods.....	147
Table 23: Supported platforms and transport protocols for the client libraries.....	154
Table 24: Supported platforms and transport protocols for the client libraries.....	156
Table 25: Supported platforms and transport protocols for the client libraries.....	159
Table 26: Supported platforms and transport protocols for the client libraries.....	161
Table 27: Supported platforms and transport protocols for the client libraries.....	163
Table 28: Supported platforms and transport protocols for the client libraries.....	165
Table 29: Capabilities provided by the Diffusion client libraries.....	169
Table 30: Session filter search clause operators.....	269
Table 31: Session filter boolean operators.....	269
Table 32: Log levels.....	438
Table 33: Log levels.....	439
Table 34: Log levels.....	440
Table 35: Log levels.....	440
Table 36: Connection types.....	443
Table 37: Types of connection that can be specified from the .NET client.....	446
Table 38: JavaScript functions called on events.....	451
Table 39: Location of the flashlog.txt file.....	461
Table 40: Location of the policyfiles.txt file.....	461
Table 41:.....	466
Table 42:.....	473
Table 43: Start publisher.....	486
Table 44: Stop publisher.....	486
Table 45: Notification methods.....	489
Table 46: General publisher utilities.....	492

Table 47: Usable methods with ordered topic data.....	498
Table 48: Usable methods with unordered topic data.....	498
Table 49: Whols.....	503
Table 50: Whols service.....	504
Table 51: Client listener notifications.....	506
Table 52: Artifacts.....	513
Table 53: Tuning changes for stress testing.....	531
Table 54: Testing tools.....	533
Table 55: Installed files.....	546
Table 56: Tools and utilities.....	547
Table 57: XML Value types.....	551
Table 58: Conflation policy elements.....	557
Table 59: Conflation policy modes.....	558
Table 60: Action depending upon merge result.....	559
Table 61: Connectors properties.....	581
Table 62: Connection restrictions.....	581
Table 63: Examples of routing strategies.....	643
Table 64: Properties that can be specified when configuring the JMS adapter.....	710
Table 65: Notifications as topics.....	752
Table 66: Client properties in the Eclipse client view.....	782
Table 67: Log levels.....	787
Table 68: Fields included in the logs.....	787
Table 69: Values that can be configured for a thread pool.....	1059
Table 70: Events that a thread pool notification handler can act on.....	1060
Table 71: Demos provided with the Diffusion server.....	1071
Table 72: Targets.....	1073
Table 73: Properties for targets start, stop and status.....	1074

Table 74: Additional properties for targets deploy and undeploy.....	1074
Table 75: Unified API interoperation.....	1076
Table 76: Classic API (deprecated) interoperation.....	1076
Table 77: API features removed in version 5.0 and 5.1.....	1080
Table 78: API features deprecated in version 5.0 and 5.1.....	1081
Table 79: API features removed in version 5.5.....	1086
Table 80: API features deprecated in version 5.5.....	1086
Table 81: API features removed in version 5.6.....	1092
Table 82: API features deprecated in version 5.6.....	1093
Table 83: API features removed in version 5.7.....	1096
Table 84: API features deprecated in version 5.7.....	1096
Table 85: API features removed in version 5.8.....	1099
Table 86: API features deprecated in version 5.8.....	1100
Table 87: API features removed in version 5.9.....	1103
Table 88: API features deprecated in version 5.9.....	1103
Table 89: Typographic conventions used in this manual.....	1109

Part I

Welcome

Welcome to the Push Technology User Manual for Diffusion™

The manual is regularly updated, but if you require further help, see the articles and forums in our Support Center: <http://support.pushtechnology.com>.

New to Diffusion?

- Learn what Diffusion is and what it can do for your organization: [Introduction](#)
- Get started with Diffusion: [Quick Start Guide](#) on page 36

Ready to start building your Diffusion solution?

- Decide what your Diffusion solution will look like: [Design Guide](#) on page 44
- Develop your Diffusion clients: [Developer Guide](#) on page 152
- Set up and manage your Diffusion server and solution: [Administrator Guide](#) on page 534

About to upgrade from an earlier version of Diffusion?

- See what's new in the latest version of Diffusion: [What's new in Diffusion 5.9?](#) on page 25
- Check how changes might affect your existing Diffusion solution: [Upgrading](#)

In this section:

- [Introducing Diffusion](#)
- [What's new in Diffusion 5.9?](#)

Introducing Diffusion

Diffusion from Push Technology provides realtime messaging, optimized for streaming data over the internet.

Flexibility, responsiveness and interactivity are some of the fundamental requirements for today's application architecture. But challenges created by unreliable and congested networks stand in the way – particularly for mobile and IoT.

The answer? A realtime integration model, with better data efficiency to address these challenges, while reducing data costs at the same time.

With Push Technology's realtime messaging products – and our unique, data-efficient approach to streaming data – developers are armed with an integration and data delivery platform optimized for today's internet-connected world.

Why realtime messaging?

Many of today's apps are *point-in-time* representations of data, refreshing information only when a user explicitly asks for an update, or continuously polling the backend. However, reactive apps are infinitely more engaging, and interactive – pushing updates in real time as new data becomes available. This allows you to scale your applications, without adding unnecessary load to back-end systems and creates a layer of decoupling that protects applications from data model changes. Realtime messaging integrates seamlessly with modern development frameworks for the web (Angular, Meteor, React, etc) delivering data updates in real time to end users. Stop focusing on point-in-time data, and instead, focus on realtime, event-driven, responsive, and engaging applications.

Benefits of Diffusion

Diffusion offers the most intelligent and data-efficient realtime messaging products available today. Designed to be easily integrated to new and existing application architecture, our technology gives you a flexible, reactive, and efficient data layer for all your business needs.

Publish-subscribe integration

The Diffusion server provides a publish-subscribe integration model. The Diffusion server stores data in a tree of topics, where each topic has a value. This value can be fetched in an ad-hoc fashion, but, more commonly, a client session subscribes to the topics that are of interest to it. The Diffusion server pushes each update to the topic to the client session as a stream of values.

Dynamic data model

Client sessions subscribe to data topics using wildcard selectors. When new topics are added, sessions with matching selectors are automatically subscribed. This avoids the need for separate topic life-cycle processes, and data objects can be frequently created and deleted without impacting existing applications.

Value-oriented programming

A value-oriented programming model is a fundamental feature of a reactive data model. Applications are built against an API that provides streams of values, rather than individual messages that need further decoding. Client SDKs provide a common programming model, making best use of the features particular to their implementation language. This frees developers to focus on application functionality, rather than data integration.

Inverted data grid

Like a data grid, Diffusion stores values in memory. Traditionally, data grids are optimized for query and primarily support a polling paradigm. In Diffusion, the data grid is optimized for a realtime, event-driven communication. Often deployed as a specialized cache, data grids offload processing from a backend system. Diffusion offers the same benefit but is designed to deliver realtime streams to a high number of subscribed clients.

Non-blocking I/O

Network communication is performed by an event-driven kernel that uses non-blocking I/O to interact efficiently with the host networking. Client sessions are partitioned and each partition is assigned to a dedicated thread that manages all subscription matching and outbound communication for that session. This lock-free design avoids contention between these sessions and allows the platform to scale linearly across CPU resources, achieving very high message rates.

Protocol optimization

Messages are serialized into a compact binary representation called CBOR. This reduces bandwidth consumption for every message – up to 30% compared with ASCII. A small binary header is used to frame each message, but this is typically only two bytes when using the default WebSocket-based protocol.

Delta streaming

Topics provide stateful streams of data to each session. Once a client session is subscribed to a topic and receives the current state, all subsequent updates are sent as a *delta* (the difference from the previous value). This happens transparently to the application, with the client SDK reconstructing the full data payload with the delta applied – so neither application or backend require changes.

Message queue optimization

Messages that cannot be immediately delivered to a session are queued. If the network connections fail, bandwidth is limited, or the client is simply slow, messages can back up on the queue. Diffusion can conflate the queue to remove messages that are stale or no longer relevant, or to combine multiple related messages into a single consolidated message.

Bandwidth management

The rate of messages delivered to a session can be artificially constrained by the platform. This can be used to prioritize one client session over another, to place limits on bandwidth utilization, to improve batching, or to encourage conflation. These measures might be desirable in some circumstances, but obviously lead to increased latency for the throttled sessions.

Reliable reconnection

Client connectivity is continuously monitored by the Diffusion server using a variety of bandwidth efficient mechanisms. If a client session is disconnected due to network failure, both the client and server queue messages for a period of time until that session can be re-established. Once the session is reconnected, the client and server reconcile the messages successfully received to ensure none are lost.

Assured delivery

During an active session, messages between the client and the Diffusion server are delivered without loss regardless of the protocol in use. Delivery is considered *assured* rather than guaranteed to account for client/server or network failures that terminate an active session.

Extensive SDK support

With simple client SDKs (Software Development Kits) available for a range of languages and environments, there are no new protocols or technologies to learn. The SDK includes a runtime library, API, documentation, and examples. The supported SDKs include JavaScript® (browser and Node.js), Apple® (iOS®, OS X®/macOS® and tvOS™), Android™, Java™, .NET, and C

Protocol fallback

By default, clients use the WebSocket protocol to establish bi-directional communication with the Diffusion server. In some cases the WebSocket protocol is not available. For example, WebSocket connections can be blocked by a firewall or load balancer, or disallowed by the network provider. In this case, clients can automatically fallback (*cascade*) to long polling over HTTP.

Session administration

A session with appropriate security permissions can act as a *control client* and instruct the Diffusion server to add and remove topics, send data updates to topics, and receive notifications about events – such as when the number of subscribers for a topic falls to zero. A control client session can also authenticate connection requests, receive notifications about new or modified sessions, modify or close sessions, and send and receive messages to and from individual sessions. Every session has session properties: a set of key-value pairs that can be modified or used to filter groups of related sessions.

Security framework

The Diffusion server has a capable security framework. Authentication can use the built-in security database or be federated to control clients that can integrate with third-party databases. Authorization is declarative and based on a configurable hierarchy of roles. Each session is assigned to one or more roles. Roles are granted permissions such as the ability to access, modify, subscribe to, or update specific topics; or the ability to control other sessions. Secure communication over TLS is supported by the Diffusion server and client libraries for all protocols.

What's new in Diffusion 5.9?

The latest version of Diffusion contains new functions, performance enhancements and bug fixes.

A complete list of the latest updates to Diffusion can be found in the Release Notes available at <http://docs.pushtechology.com/docs/5.9.24/ReleaseNotice.html>.

TypeScript support in the JavaScript API

The JavaScript API now includes TypeScript definitions. These definitions enable developers to more efficiently develop JavaScript clients for Diffusion while minimizing the risk of runtime errors caused by type mismatches.

For more information, see [JavaScript](#) on page 153.

Transport cascading in the Android and Java Unified API

You can specify that your client attempts connection using more than one transport. If a connection attempt using the first specified transport fails, the client cascades to the next specified transport and uses it to attempt connection.

For more information, see [Connecting basics](#) on page 239.

Additional support for JSON and binary topics in the Apple API

Apple Unified API clients now provide the ability to create and subscribe to JSON and binary topics.

JSON topics enable you to structure your data using JSON. The data is transmitted in a CBOR binary form for increased efficiency.

For more information, see [JSON topics](#) on page 71.

Binary topics enable you to stream your data in pure binary form without the overhead of having to encode the binary data into string form.

For more information, see [Binary topics](#) on page 73.

Reliable reconnection for automated fan-out

The automated fan-out feature now makes use of all the advantages of Diffusion's reliable reconnection, which was [introduced in version 5.8](#).

Previously, fan-out could not make use of standard reconnection due to the possibility of message loss. Now a fan-out secondary server that loses its connection to the primary server can reconnect without any loss of topics.

For more information, see [Configuring fan-out](#) on page 555.

Propagation of missing topic notifications over automated fan-out connections

Missing topic notifications generated by subscription or fetch requests to a secondary server are now propagated to missing topic handlers registered against the primary servers.

Control clients can use these notifications to monitor the activity of end-user clients. It was previously necessary for control clients to establish separate sessions with secondary servers to receive these notifications. Now clients can receive missing topic notifications through a single session with a primary server.

For more information, see [Using missing topic notifications with fan-out](#) on page 102.

Topics can be created without first creating parent topics

The relationship between topics and the topic tree has changed significantly in this release.

In previous releases, creating a topic at a topic path – for example, a/b – required that there was a topic at all parent paths – for example, a. If not present, these parent topics were created as stateless topics at the time the child topic was created.

From Diffusion version 5.9, a topic can be created at any topic path where there is not an existing topic. No additional parent topics are created. Empty intermediate topic paths remain empty and topics can be created at these empty paths at a later point.

Because there is no longer a requirement for every topic to have a parent topic, topics can now be deleted without affecting or deleting any child topics.

Note: These changes only apply to Unified API interactions with the topic tree. Publishers still create required parent topics when creating a topic and still delete all topics below a topic selected for deletion.

JMS adapter now available as a standalone application

The Diffusion JMS adapter, which previously could only be run as part of the Diffusion server process, can now also be run as a standalone application on a separate system to the Diffusion server.

For more information, see [JMS](#) on page 124.

Related concepts

[Upgrading Guide](#) on page 1075

If you are planning to move from an earlier version of Diffusion to version 5.9, review the following information about changes between versions.

What was new in Diffusion 5.8?

The latest version of Diffusion contains new functions, performance enhancements and bug fixes.

A complete list of the latest updates to Diffusion can be found in the Release Notes available at <http://docs.pushtechnology.com/docs/5.9.24/ReleaseNotice.html>.

Reconnection improvements

Previously, when a client lost connection to the Diffusion server and then reconnected, messages might have been lost in transmission.

Diffusion now re-synchronizes the streams of messages from client to the Diffusion server and from the Diffusion server to client when the client reconnects. When reconnecting, the client notifies the Diffusion server of the last message received and the earliest message it can send again. Diffusion sends any missing messages again and instructs the client to resume from the appropriate message.

The Android and Java Unified API clients also maintain a buffer of messages that can be sent again to the Diffusion server.

If any messages have been lost, the Diffusion server aborts the reconnection. If reconnection succeeds, no messages have been lost.

For more information, see [Reconnect to the Diffusion server](#) on page 252

JSON and binary topics in .NET

.NET Unified API clients now provide the ability to create, update, and subscribe to JSON and binary topics.

JSON topics enable you to structure your data using JSON. The data is transmitted in a CBOR binary form for increased efficiency.

For more information, see [JSON topics](#) on page 71.

Binary topics enable you to stream your data in pure binary form without the overhead of having to encode the binary data into string form.

For more information, see [Binary topics](#) on page 73.

What was new in Diffusion 5.7?

Diffusion 5.7 contains new functions, performance enhancements and bug fixes.

A complete list of the latest updates to Diffusion can be found in the Release Notes available at <http://docs.pushtechnology.com/docs/5.9.24/ReleaseNotice.html>.

JSON and binary topics

The Java, JavaScript, and Android Unified API clients now provide the ability to create, update, and subscribe to JSON and binary topics.

JSON topics enable you to structure your data using JSON. The data is transmitted in a CBOR binary form for increased efficiency.

For more information, see [JSON topics](#) on page 71.

Binary topics enable you to stream your data in pure binary form without the overhead of having to encode the binary data into string form.

For more information, see [Binary topics](#) on page 73.

Create and update topics from the Apple Unified API

The TopicControl and TopicUpdateControl features are now available in the Apple Unified API.

For more information, see [and Updating topics](#) on page 331.

Enhancements to the C Unified API

The following new features have been added to the C Unified API:

- WebSocket support.

The WebSocket implementation provides a browser-based full duplex connection, built on top of WebSocket framing. This complies with the WebSocket standards and is usable with any load balancer or proxy with support for WebSocket.

- The ClientControl feature.

The C Unified API now enables you to receive notifications when client sessions open or close and get session properties for a client session. For more information, see [Managing clients](#) on page 431.

New security permission: select_topic

The select_topic security permission has been added to the list of topic-scoped permissions. This permission controls whether sessions can subscribe to or fetch from particular parts of the topic tree. For more information, see [Permissions](#) on page 134.

If you are upgrading an existing configuration to work with Diffusion 5.7, update your `Security.store` file to grant this new permission to the appropriate roles. For more information, see [Upgrading from version 5.6 to version 5.7](#) on page 1095.

HTTP Polling transport in the JavaScript Unified API

HTTP polling uses HTTP to make a long poll request. This transport is supported by most proxies and load balancers and enables your clients to connect to the Diffusion server in situations where WebSocket connections are not allowed.

For more information, see [Client types](#) on page 111.

Transport cascading in the JavaScript Unified API

You can specify that your client attempts connection using more than one transport. If a connection attempt using the first specified transport fails, the client cascades to the next specified transport and uses it to attempt connection.

For more information, see [Connecting basics](#) on page 239.

Log4j2 logging

Diffusion can now be configured to use log4j2 as an alternative logging implementation. The log4j2 implementation of SLF4J supports a wide range of appenders and allows fine-grained tuning of the events that are logged.

For more information, see [Logging back-end](#) on page 785. For the log4j2 documentation, see <http://logging.apache.org/log4j/2.x/manual>.

Increased interoperability of client and server versions

The Diffusion server and Diffusion clients can now negotiate well-defined interoperability of services based on their respective versions, for version 5.6 and later. This enables clients to connect to a Diffusion server with a later version and for both the client and the server to respond gracefully to differences of capability.

For more information, see [Interoperability](#).

Paged topic support in automated fan-out

Paged string topics and paged record topics are now added to the list of topic types that can be distributed by automated fan-out.

For more information, see [Fan-out](#) on page 99.

What was new in Diffusion 5.6?

Diffusion 5.6 contains new functions, performance enhancements and bug fixes.

A complete list of the latest updates to Diffusion can be found in the Release Notes available at <http://docs.pushtechology.com/docs/5.9.24/ReleaseNotice.html>.

Apple Unified API

An Apple version of the Unified API is now available for iOS and OS X/macOS.

For version 5.6, the iOS Classic API (the API used in version 4 and earlier) is still supported.

For more information, see [Apple](#) on page 156.

Android Unified API

The Java version of the Unified API is now available for Android.

For version 5.6, the Android Classic API (the API used in version 4 and earlier) is still supported.

For more information, see [Android](#) on page 158.

Push notifications

This release introduces the Push Notification Bridge. The bridge is a Diffusion client that subscribes to topics on behalf of other Diffusion client applications and uses a push notification network to relay topic updates to the device where the client application is located.

For more information, see [Push notification networks](#) on page 123.

Automatic fan-out

A Diffusion server can be configured to as a secondary server to a specified primary Diffusion server and automatically replicate all or part of the topic tree of the primary server at the secondary server. In previous releases, this fan-out distribution was only achieved using topic notification topics and publishers at both the primary and secondary server.

For more information, see [Fan-out](#) on page 99.

Session failover for Unified API clients

Unified API clients can now reconnect to a replicated session on a secondary server if the primary server they first connected to becomes unavailable.

In previous releases, only Classic API clients had this capability.

For more information, see [Session replication](#).

Session properties enhancements

The SubscriptionControl feature now takes advantage of session properties to enable you to select client sessions by their properties and subscribe the selected sessions to a topic or topics.

For more information, see [Managing subscriptions](#) on page 354.

The ClientControl feature can now register a session properties listener which is notified of state changes of clients along with selected session properties values. The ClientControl feature can also get the properties of any connected client session.

For more information, see [Managing clients](#) on page 431.

TopicControl feature enhancements

You can now listen for topic events on branches of the topic tree and receive a notification when any topic in that branch changes from having zero subscribers to having one or more subscribers or from having one or more subscribers to having zero subscribers.

For more information, see [Listening for topic events](#) on page 329.

Optimizations to topic subscription evaluation

Subscription processing has been optimized.

The memory required to record standard topic subscriptions is significantly reduced. The memory footprint for routing or slave topic subscriptions has also been reduced.

When topics are created, the process of evaluating topic selections for existing sessions is more efficient. Subscriptions are evaluated in parallel using multiple CPU cores, and the processing is now asynchronous and batched. This provides a more resilient response to a sudden load.

This change to subscription evaluation also changes the performance of certain publisher features. For more information, see [Handling client subscriptions](#) on page 495.

What was new in Diffusion 5.5?

Diffusion 5.5 contains new functions, performance enhancements and bug fixes.

A complete list of the latest updates to Diffusion can be found in the Release Notes available at <http://docs.pushtechology.com/docs/5.9.24/ReleaseNotice.html>.

JavaScript Unified API

A JavaScript version of the Unified API is now available. This API contains both control and standard capabilities.

For version 5.5, the JavaScript Classic API (the API used in version 4 and earlier) is still supported.

For more information, see .

New security model

Diffusion is now secured using a role-based model. Actions that can be performed on the server by client sessions are controlled by permissions. The client session must be assigned a role that has the required permission to be able to perform the action.

For more information, see [Role-based authorization](#) on page 130.

Console authentication

To access the Diffusion console you must now authenticate with a principal and a password. The information that you can view through the console and the actions you can take are controlled by the permissions assigned to the roles associated with that principal.

For more information, see [Diffusion monitoring console](#) on page 759.

Session properties

Information about client sessions is now available to Unified API clients using control features. These session properties can be used to filter client sessions.

For more information, see [Session properties](#) on page 267.

New JMS adapter

A new JMS adapter is available that provides enhanced functionality. The new JMS adapter provides more powerful configuration options and abstracts all JMS-specific behavior. This enables Diffusion clients to interact with data exchanged with a JMS server as Diffusion topics and data, without having to be aware of specifics of the JMS implementation.

The legacy JMS adapter v5.1 is deprecated and will be removed in a future release. We recommend you move to the new version.

For more information, see [JMS](#) on page 124.

Removal of message fragmentation

The message fragmentation capability has been removed from Diffusion.

Topic message fragmentation was intended to prevent head-of-line blocking by large messages. The API allowed messages for a given topic messages to be delivered out of order, which is incompatible with snapshot/delta processing.

If you applied topic message fragmentation to work around the maximum message size limitations, particularly for large topic load messages, we recommend instead that you increase the maximum message size to accommodate the largest possible application message.

Increasing the maximum message size to support large topic load messages will also require increasing the client input buffer and server output buffer sizes. The peak memory requirement is lower than needed when topic message fragmentation is enabled, but is approximately twice the maximum message size. In a future release, we will improve the buffer handling to allow the maximum message size to exceed the network buffer size.

Pooled input buffers

Changes to how input buffers are allocated improve the performance and memory usage of the Diffusion server.

Input buffers are no longer bound to clients, instead they are shared by all reading tasks. Where previously the number of connected clients defined the number of input buffers, now the maximum number of input buffers is bounded by the configured thread pool size.

The maximum amount of memory used for input buffers is less than the thread pool size multiplied by the input buffer size plus any small memory usage resulting from partial reads.

Non-exclusive updating

The Unified API now provides a simpler way for clients to update topics if the order of updates from different clients is unimportant. This is supported by the Java, .NET, and JavaScript Unified API clients.

A client can register an update source against a topic. If the topic update source is active, it prevents updates to the topic from any other clients. However, a client can also update a topic without registering as an update source against it. If there is no update source registered against that topic, this non-exclusive update is successful. If multiple clients make non-exclusive updates to a topic, the most recent update is the one applied.

For more information, see .

Windows and OSX support for the C Unified API

We provide libraries for the C Unified API that are compiled for Windows and OSX.

For more information, see [C](#) on page 164.

Proxy support (.NET Unified API)

Clients that use the .NET Unified API can now connect to the Diffusion server through a proxy. The Unified API enables you to connect through the proxy unauthenticated, with basic authentication, or through any other authentication process by implementing your own challenge handlers.

For more information, see [Connecting through an HTTP proxy](#) on page 250.

Flow control (.NET Unified API)

The .NET client assemblies can now control the flow of requests from a client to decrease the likelihood of the client's outbound queue or the client queue on the server overflowing and causing the client to be disconnected.

This process happens automatically when the client detects conditions that might cause a queue overflow and increases the reliability of the .NET client.

What was new in Diffusion 5.1?

Diffusion 5.1 contains new functions, performance enhancements and bug fixes.

Key features

A complete list of the latest updates to Diffusion can be found in the Release Notes available at <http://download.pushtechnology.com>.

Paged topic support in the Java and .NET Unified API

You can use the TopicControl feature to create paged record topics and paged string topics and to define rule-based comparators to use for ordering the lines in the paged topic.

You can use the TopicUpdateControl feature to update paged record topics and paged string topics.

UpdateSource capabilities in the Unified API

The UpdateSource capabilities replace the TopicSource capabilities. UpdateSource includes the ability to build more complex updates, support for more topic types, and better handling of unexpected closes.

For more information, see the API documentation.

Remove topics after the control client closes using the Unified API

The TopicControl feature now enables you to specify whether to remove sections of the topic tree after a control client session closes.

For more information, see [Removing topics with sessions](#) on page 330.

.NET and C Unified API production support

The .NET and C Unified API are now supported for production use. Both APIs contain functionality to implement a control client.

For more information, see [.NET](#) on page 162 and [C](#) on page 164.

Proxy support

Clients that use the Java Unified API can now connect to the Diffusion server through a proxy. The Unified API enables you to connect through the proxy unauthenticated, with basic authentication, or through any other authentication process by implementing your own challenge handlers.

For more information, see [Connecting through an HTTP proxy](#) on page 250.

WebSocket support in the iOS Classic API

Clients implemented using the iOS Classic API can now connect to the Diffusion server through the WebSocket protocol.

Streams replace listeners for receiving content through the Unified API

The `Listeners` in the Topics and Messaging features are now deprecated. Listeners have been replaced by streams. `Topics.TopicStream` receives topic events, such as topic updates for a topic or topics. `Messaging.MessageStream` receives messages sent through a topic or topics.

Streams provide advantages over listeners as a stream has a logical end. A stream can be closed or discarded, at which time the stream has the opportunity to do any required cleanup or take any required actions.

Flow control (Java Unified API)

The Java client library can now control the flow of requests from a client to decrease the likelihood of the client's outbound queue or the client queue on the server overflowing and causing the client to be disconnected.

This process happens automatically when the client detects conditions that might cause a queue overflow and increases the reliability of the Java client.

What was new in Diffusion 5.0?

Diffusion 5.0 contains new functions, performance enhancements and bug fixes.

Key features

A complete list of the latest updates to Diffusion can be found in the Release Notes available at <http://download.pushtechnology.com>.

New high availability features

In version 5.0, Diffusion introduces the following new high availability features: session replication, topic replication, and failover of the active update source. These features use a datagrid to share data between multiple Diffusion servers.

Session replication shares client session information between servers. If a client loses connection to a server, it is reconnected through a load balancer to another server that has access to all of the client's session information.

Topic replication shares topic information – such as the topic definition and metadata – and topic data between servers. If a server becomes unavailable, the topic information and data is available on another server.

Only one server can act as the active update source for a topic or branch of the topic tree. If that server becomes unavailable, other servers can take over as the active update source for those topics.

For more information, see [High availability](#) on page 104.

Control client

Control clients are a way to package application logic that controls a Diffusion server. Unlike publishers, control clients run as a separate process outside of the server and use the Diffusion client library to communicate with the server.

Control clients use the Unified API to provide a secure remote control experience that can use all of the supported protocols to communicate with the Diffusion server. They can be implemented in any of the supported languages.

Introducing the Unified API

Beginning in version 5.0, Diffusion is transitioning to a new public API. The Unified API will make available the capabilities of standard clients, control clients, and event publishers in one consistent, modular interface.

For version 5.0, the control features are now available. This enables you to replace remote control with the richer experience of control client.

The Classic API (the API used in version 4 and earlier) is still supported in 5.0 for clients and event publishers. The remote control API is no longer supported.

Improved performance

Diffusion can now serve up to 150% more messages per second to 60% more clients by using a new queuing mechanism.

In benchmark tests, using 50 topics and 125-byte messages, Diffusion served 15 million messages per second to 87,000 clients. Diffusion used 24 threads and three client processes to achieve this performance.

New authentication model

In Diffusion 5.9 we have split out the concept of authentication from that of authorization. You can write and configure both remote and local authentication handlers.

In previous versions, the authentication capability was provided by authorization handlers. Using authorization handlers for authentication is now deprecated. We recommend that you re-implement your authentication logic using the version 5 authentication APIs.

For more information, see [User access control](#).

JavaScript API for paged topics

The JavaScript API now includes improved capability to work with paged topics.

Iframe streaming

Iframe streaming connections are now available over the HTTP protocol.

Liveness monitoring in Flex[®] and JavaScript

The Flex and JavaScript client libraries now include liveness monitors that listen for activity from the server and raise an event if the lack of activity indicates that the connection has been lost. This enables the client to reconnect in the event of a lost connection.

Part II

Quick Start Guide

Push Technology's Diffusion is the ingredient software required to resolve the limitations and challenges of data distribution by speeding up the delivery of content, enabling rapid scale and optimizing data sent and received.

This guide gives you an overview of installing the Diffusion server and starting to develop your own clients.

In this section:

- [Get Diffusion](#)
- [Install Diffusion](#)
- [Start the Diffusion server](#)
- [Default configuration](#)
- [The Diffusion monitoring console](#)
- [Develop a publishing client](#)
- [Develop a subscribing client](#)
- [Resources](#)

Get Diffusion

To install Diffusion you require the product jar, `Diffusion5.9.24.jar`, and the standalone installer jar, `install.jar`.

You can get both of these files from the Push Technology Download site: <http://download.pushtechnology.com/releases/5.9>

Install Diffusion

Java 8 is required to install Diffusion.

1. Double-click on the `install.jar` file to launch the graphical installer.

The installer locates the Diffusion JAR file if that file is in the same directory as the installer.

2. If the installer cannot locate the Diffusion JAR file, select **File > Load install file** and navigate to the Diffusion JAR file and click **Open**.
3. If you have a production license, select **File > Load license file** and navigate to the license file and click **Open**.

If you do not have a production license, you can use the developer license that is included in the Diffusion server

4. Follow the steps in the graphical installer to install the Diffusion server.

For more information about system requirements and installing the Diffusion server, see [Installing the Diffusion server](#) on page 535

Start the Diffusion server

The Diffusion start scripts are located in the `bin` directory of your Diffusion installation.

On Windows: Use `diffusion.bat` to start the Diffusion server.

On Linux or OS X/macOS: Use `diffusion.sh` to start the Diffusion server.

Default configuration

The Diffusion server is configured by the files in the `etc` directory.

License

Diffusion includes a development license that allows up to 5 concurrent connections to the Diffusion server.

You can upgrade your Diffusion server to use a production license by copying the production license file into the `etc` directory of your Diffusion installation.

Security

Diffusion uses role-based security for authorization. A client session must have a role with sufficient permissions to perform specific actions on the Diffusion server.

The default security configuration is located in the `etc/SystemAuthentication.store` and `etc/Security.store` files of your Diffusion installation.

This configuration includes usernames and passwords for development use:

Username	Password
client	password
control	password
admin	password
operator	password

We recommend that you change these passwords as soon as possible by editing the `etc/SystemAuthentication.store` file.

Configuration

The Diffusion server is configured by the XML files in the `etc` directory of your Diffusion server.

The default configuration makes port 8080 available to connecting clients.

The Diffusion monitoring console

Diffusion includes a monitoring console that shows realtime information about the Diffusion server and its clients, publishers, and topics.

In a browser, go to `http://localhost:8080`. If your browser is not on the same system as your Diffusion server, replace `localhost` with the hostname or IP address of the Diffusion server.

From this landing page you can go to either the demos page, where you can access the Diffusion demos if you opted to deploy them during the install process, or to the Diffusion monitoring console.

The Diffusion monitoring console is secured by username and password. Use the default user 'admin' and password 'password' to log in to the console.

For more information about the Diffusion monitoring console, see [Diffusion monitoring console](#) on page 759

Develop a publishing client

Use the Diffusion JavaScript API to develop a Node.js client that creates a JSON topic and publishes updates to it.

1. Install Node.js on your development system.

For more information, see <https://nodejs.org/en/>

2. Install the Diffusion JavaScript library on your development system.

```
npm install --save diffusion
```

3. Create a JavaScript file, `publisher.js`, to contain your client.
4. Include the Diffusion JavaScript library at the top of your `publisher.js` file.

```
const diffusion = require('diffusion');
```

5. Create a connection from the page to the Diffusion server.

```
diffusion.connect({
  host : 'localhost',
  port : '8080',
  principal : 'client',
  credentials : 'password'
}).then(function(session) {
  console.log('Connected!');
});
```

6. Create a JSON topic called topic/json:

```
.then(function(session) {
  console.log('Connected!');

  session.topics.add('topic/json',
    diffusion.topics.TopicType.JSON);
})
```

The `add()` method takes the name of the topic and the topic type.

7. Update the topic with a timestamp:

```
var d = new Date();
session.topics.update('topic/json', {
  "timestamp" : d.getTime()
});
```

The `update()` method takes the name of the topic and a JSON object.

8. Wrap the update in a loop function that causes an update to occur every second:

```
setInterval(function() {
  var d = new Date();
  session.topics.update('topic/json', {
    "timestamp" : d.getTime()
  });
}, 1000);
```

9. Run the Node.js client:

```
node publisher.js
```

Full example

The following example code shows a Node.js JavaScript client that connects to the Diffusion server, creates a JSON topic, and publishes an update to it.

```
const diffusion = require('diffusion');

diffusion.connect({
  host : 'localhost',
  port : '8080',
  principal : 'control',
  credentials : 'password'
}).then(function(session) {
  console.log('Connected!');

  // Create a JSON topic
```

```

    session.topics.add('topic/json',
diffusion.topics.TopicType.JSON);

    // Start updating the topic every second
    setInterval(function() {
        var d = new Date();
        session.topics.update('topic/json', {
            "timestamp" : d.getTime()
        });
    }, 1000);
});

```

To run the example:

1. Install Node.js.

For more information, see <https://nodejs.org/en/>

2. Install the Diffusion JavaScript library on your development system.

```
npm install --save diffusion
```

3. Copy the provided code into a file called `publisher.js`
4. Update the `connect` method to include the URL of your Diffusion server.
5. If you have changed the default security configuration, change the principal and credentials to those of a user that has the `modify_topic` and `update_topic` permissions.
6. Use Node.js to run your publishing client from the command line.

```
node publisher.js
```

The JavaScript client opens a connection to the Diffusion server, creates the topic `topic/json`, and updates it each second with a timestamp.

Publish using other Diffusion APIs:

- [Java](#)
- [.NET](#)
- [JavaScript](#)
- [Apple](#)
- [Android](#)
- [C](#)

Develop a subscribing client

Use the Diffusion JavaScript API to develop a client that subscribes to a JSON topic and receives updates published through it.

1. Ensure that the `diffusion.js` file, located in the `clients/js` directory of your Diffusion installation, is available on your development system.
2. Create a template HTML page which displays the information.

For example, create the following `index.html` in your project's HTML directory.

```

<html>
  <head>
    <title>JSON example</title>
  </head>
  <body>
    <span>The value of topic/json is: </span>

```

```

    <span id="display">Unknown</span>
  </body>
</html>

```

3. Include the Diffusion JavaScript library in the `<head>` section of your `index.html` file.

```

<head>
  <title>JSON example</title>
  <script type="text/javascript" src="path_to_library/
diffusion.js"></script>
</head>

```

4. Create a connection from the page to the Diffusion server. Add a `script` element to the `body` element.

```

<body>
  <span>The value of topic/json is: </span>
  <span id="display">Unknown</span>
  <script type="text/javascript">
    diffusion.connect({
      // Edit this line to include the URL of your Diffusion
server
      host : 'localhost',
      port : 8080,
      principal : 'client',
      credentials : 'password'
    }).then(function(session) {
      alert('Connected: ' + session.isConnected());
    }
  );
</script>
</body>

```

If you run the client now, it displays a dialog box when it successfully connects to the Diffusion server.

5. Subscribe to a topic and create a stream that receives data from it.

Add the following function before the `diffusion.connect()` call:

```

function subscribeToJsonTopic(session) {
  session.subscribe('topic/json');
  session.stream('topic/
json').asType(diffusion.datatypes.json()).on('value',
function(topic, specification, newValue, oldValue) {
  console.log("Update for " + topic, newValue.get());
  document.getElementById('display').innerHTML =
JSON.stringify(newValue.get());
});
}

```

The `subscribe()` method of the `session` object takes the name of the topic to subscribe to. The `stream()` method of the `session` creates a value stream that receives updates from the topic and emits an event on each update. The attached function takes the data from the topic and updates the `display` element of the web page with the topic data.

6. Change the function that is called on connection to the `subscribeToJsonTopic` function you just created.

```

.then(subscribeToTopic);

```

Now, when the client connects to the Diffusion server it subscribes to `topic/json` and creates a stream to receive updates from that topic. Those JSON updates are displayed in the browser window as a string.

Full example

The following example code shows a browser JavaScript client that connects to the Diffusion server and subscribes to a JSON topic.

```
<html>
  <head>
    <title>JSON example</title>
    <script type="text/javascript" src="path_to_library/
diffusion.js"></script>
  </head>
  <body>

    <span>The value of topic/json is: </span>
    <span id="display">Unknown</span>
    <script type="text/javascript">
      function subscribeToJsonTopic(session) {
        session.subscribe('topic/json');
        session.stream('topic/
json').asType(diffusion.datatypes.json()).on('value', function(topic,
specification, newValue, oldValue) {
          console.log("Update for " + topic, newValue.get());
          document.getElementById('display').innerHTML =
JSON.stringify(newValue.get());
        });
      }

      diffusion.connect({
        host : 'localhost',
        port : 8080,
        principal : 'client',
        credentials : 'password'
      }).then(subscribeToJsonTopic);
    </script>
  </body>
</html>
```

To run the example:

1. Copy the provided code into a file called `index.html`
2. Update the `connect` method to include the URL of your Diffusion server.
3. If you have changed the default security configuration, change the principal and credentials to those of a user that has the `read_topic` and `select_topic` permissions.
4. Open `index.html` in a browser.

The JavaScript client opens a connection to the Diffusion server, subscribes to the topic `topic/json`, and prints the updates it receives to the browser window.

Subscribe using other Diffusion APIs:

- [JavaScript](#)
- [Apple](#)
- [Android](#)
- [Java](#)
- [.NET](#)
- [C](#)

Resources

- Download site: <http://download.pushtechnology.com/releases/5.9>
- API documentation: <http://docs.pushtechnology.com/5.9>
- Support center: <http://support.pushtechnology.com>
- Stack Overflow: <http://stackoverflow.com/questions/tagged/push-diffusion>
- Github: <https://github.com/pushtechnology/>

Part III

Design Guide

This guide describes the factors to consider when designing your Diffusion solution.

In this section:

- [Support](#)
- [Designing your data model](#)
- [Designing your solution](#)
- [Security](#)

Support

When designing your solution, refer to the support information to ensure compatibility between the Diffusion server and your hardware, software, and operating systems. This section also provides information about the capabilities of the Diffusion clients and the platforms the clients are supported on.

You can also refer to the Upgrading Guide to review the compatibility between different versions of Diffusion. For more information, see [Interoperability](#).

System requirements for the Diffusion server

Review this information before installing the Diffusion server.

The Diffusion server is certified on the system specifications listed here. In addition, the Diffusion server is supported on a further range of systems.

Certification

Push Technology classes a system as certified if the Diffusion server is fully functionally tested on that system.

We recommend that you use certified hardware, virtual machines, operating systems, and other software when setting up your Diffusion servers.

Support

In addition, Push Technology supports other systems that have not been certified.

Other hardware and virtualized systems are supported, but the performance of these systems can vary.

More recent versions of software and operating systems than those we certify are supported.

However, Push Technology can agree to support Diffusion on other systems. For more information, contact Push Technology.

Physical system

The Diffusion server is certified the following physical system specification:

- Intel™ Xeon™ E-Series Processors
- 8 Gb RAM
- 8 CPUs
- 10 Gigabit NIC

Network, CPU, and RAM (in decreasing order of importance) are the components that have the biggest impact on performance. High performance file system and disk are required. Intel hardware is used because of its ubiquity in the marketplace and proven reliability.

Virtualized system

The Diffusion server certified on the following virtualized system specification:

Host

- Intel Xeon E-Series Processors
- 32 Gb RAM

- VMware vSphere® 5.5

Virtual machine

- 8 VCPUs
- 8 Gb RAM

Operating system

Diffusion is certified on the following operating systems:

- Red Hat 6.5, 6.6, and 7.2
- Windows Server 2012 R2

We recommend you install your Diffusion server on a Linux-based operating system with enterprise-level support available, such as Red Hat Enterprise Linux.

Operating system configuration

If you install your Diffusion server on a Linux-based operating system and do SSL offloading of secure client connections at the Diffusion server, you must disable transparent huge pages.

If you install your Diffusion server on a Linux-based operating system but do not do SSL offloading of secure client connections at the Diffusion server, disabling transparent huge pages is still recommended.

Having transparent huge pages enabled on the system your Diffusion server runs on can cause extremely long pauses for garbage collection. For more information, see <https://access.redhat.com/solutions/46111>.

Java

The Diffusion server is certified on Oracle Java 8 64-bit JDK

Only the Oracle® JDK is certified.

Ensure that you use the Oracle JDK and not the JRE.

JVM configuration

If you do SSL offloading of secure client connections at the Diffusion server, you must ensure that you constrain the maximum heap size and the maximum direct memory size so that together these to values do not use more than 80% of your system's RAM.

Networking

Push Technology recommends the following network configurations:

- 10 Gigabit network
- Load balancers with SSL offloading
- In virtualized environments, enable SR-IOV.

For more information about how to enable SR-IOV, see the documentation provided by your virtual server provider. SR-IOV might be packaged using a vendor-specific name.

Client requirements

For information about the supported client platforms, see [Platform support for the Diffusion Unified API libraries](#) on page 47.

Related concepts

[The Diffusion license](#) on page 543

Diffusion includes a development license that enables you to use make up to 5 concurrent connections to the Diffusion server.

[Installed files](#) on page 546

After installing Diffusion the following directory structure exists:

Related tasks

[Installing the Diffusion server using the graphical installer](#) on page 537

The Diffusion binary files are available from the Push Technology website. You can install Diffusion using the graphical installer.

[Installing the Diffusion server using the headless installer](#) on page 539

The Diffusion binary files are available from the Push Technology website. You can install Diffusion from the command line.

[Installing the Diffusion server using Red Hat Package Manager](#) on page 540

Diffusion is available as an RPM file from the Push Technology website.

[Installing the Diffusion server using Docker](#) on page 541

Diffusion is available as a Docker® image from Docker Hub.

[Verifying the Diffusion installation](#) on page 548

Start your Diffusion server, review the logs, and connect to the console to verify that your installation is correct.

Platform support for the Diffusion Unified API libraries

Review this information when designing your clients to determine what platforms and transports the Diffusion Unified API client libraries are supported on.

Supported platforms and protocols for the client libraries

Table 1: Supported platforms and transport protocols for the client libraries

Platform	Minimum supported versions	Supported transport protocols
JavaScript	es6 (TypeScript 1.8)	WebSocket HTTP (Polling XHR)
Apple for iOS	Development environment Xcode 7 (iOS 9.0 SDK) Runtime support Deployment target: iOS 7.0 or later	WebSocket

Platform	Minimum supported versions	Supported transport protocols
	Device architectures: armv7, armv7s, arm64 Simulator architectures: i386, x86_64	
Apple for OS X/macOS	Development environment Xcode 7 (OS X 10.11 SDK) Runtime support Deployment target: OS X 10.9 or later Device architectures: x86_64	WebSocket
Apple for tvOS	Development environment Xcode 7 (tvOS 9.0 SDK) Runtime support Deployment target: tvOS 9.0 or later Device architectures: arm64 Simulator architectures: x86_64	WebSocket
Android	API 19 / v4.4 / KitKat Note: Push Technology provides only best-effort support for Jelly Bean (API 16-18, v4.1-4.3).	WebSocket HTTP (polling) DEPRECATED: DPT DEPRECATED: HTTP (Full duplex)

Platform	Minimum supported versions	Supported transport protocols
Java	8 (recommended), 7 (supported) Note: We recommend that you run your clients on the JDK rather than the JRE.	WebSocket HTTP (Polling) DEPRECATED: DPT DEPRECATED: HTTP (Full duplex)
.NET	4.5	WebSocket DEPRECATED: DPT DEPRECATED: HTTP (Full duplex)
C for Linux	Red Hat and CentOS™, version 7.2 and later Ensure that you use a C99-capable compiler.	WebSocket DEPRECATED: DPT
C for Windows	Visual C Compiler 2013 or later, Windows 7 or later	WebSocket DEPRECATED: DPT
C for OS X/macOS	For building using GCC, use Xcode 7.1 or later	WebSocket DEPRECATED: DPT

Note: Protocols are supported for both secure and standard connections.

Feature support in the Diffusion Unified API

Review this information when designing your clients to determine which APIs provide the functionality you require.

Features are sets of capabilities provided by the Diffusion Unified API. Some features are not supported or not fully supported in some APIs.

The Diffusion libraries also provide capabilities that are not exposed through their APIs. Some of these capabilities can be configured.

Table 2: Capabilities provided by the Diffusion client libraries

Capability	JavaScript	Apple	Android	Java	.NET	C
Connecting						
Connect to the Diffusion server	✓	✓	✓	✓	✓	✓
Cascade connection through multiple transports	✓	✗	✓	✓	✗	✗

Capability	JavaScript	Apple	Android	Java	.NET	C
Connect asynchronously	✓	✓	✓	✓	✓	✓
Connect synchronously	✗	✗	✓	✓	✓	✓
Connect using a URL-style string as a parameter	✗	✓	✓	✓	✓	✓
Connect using individual parameters	✓	✗	✓	✓	✗	✗
Connect securely	✓	✓	✓	✓	✓	✓
Configure SSL context or behavior	✓	✓	✓	✓	✓	✗
Connect through an HTTP proxy	✗	✓	✓	✓	✓	✗
Connect through a load balancer	✓	✓	✓	✓	✓	✓
Pass a request path to a load balancer	✓	✗	✓	✓	✗	✗
Reconnecting						
Reconnect to the Diffusion server	✓	✓	✓	✓	✓	✓
Failover to a replicated session on a different Diffusion server	✓	✓	✓	✓	✓	✓
Configure a reconnection timeout	✓	✓	✓	✓	✓	✗
Define a custom reconnection strategy	✓	✓	✓	✓	✓	✓
Resynchronize message streams on reconnect	✓	✓	✓	✓	✓	✗
Abort reconnect if resynchronization fails	✓	✓	✓	✓	✓	✗
Maintain a recovery buffer of messages on the client to resend to the	✓	✗	✓	✓	✗	✓

Capability	JavaScript	Apple	Android	Java	.NET	C
Diffusion server on reconnect						
Configure the client-side recovery buffer	✗	✗	✓	✓	✗	✗
Detect disconnections by monitoring activity	✓	✓	✓	✓	✓	✗
Detect disconnections by using TCP state	✓	✓	✓	✓	✓	✓
Ping the Diffusion server	✗	✓	✓	✓	✓	✓
Change the principal used by the connected client session	✓	✓	✓	✓	✓	✓
Receiving data from topics						
Subscribe to a topic or set of topics	✓	✓	✓	✓	✓	✓
Receive data as a value stream (JSON, binary, and single value topics)	✓	✓	✓	✓	✓	✗
Receive data as content (all topic types)	✓	✓	✓	✓	✓	✓
Fetch the state of a topic	✓	✓	✓	✓	✓	✓
Managing topics						
Create a JSON or binary topic	✓	✓	✓	✓	✓	✗
Create a topic (not including JSON or binary topics)	✓	✓	✓	✓	✓	✓
Create a topic from an initial value	✓	✗	✓	✓	✓	✗
Create a topic with metadata	✓	✓	✓	✓	✓	✓
Listen for topic events (including topic has subscribers and topic has zero subscribers)	✗	✓	✓	✓	✓	✗

Capability	JavaScript	Apple	Android	Java	.NET	C
Delete a topic	✓	✓	✓	✓	✓	✓
Delete a branch of the topic tree	✓	✓	✓	✓	✓	✓
Mark a branch of the topic tree for deletion when this client session is closed	✓	✓	✓	✓	✓	✗
Updating topics						
Update a JSON or binary topic	✓	✗	✓	✓	✓	✗
Update a topic (not including JSON and binary topics)	✓	✓	✓	✓	✓	✓
Perform exclusive updates	✓	✓	✓	✓	✓	✓
Perform non-exclusive updates	✓	✓	✓	✓	✓	✗
Managing subscriptions						
Subscribe or unsubscribe another client to a topic	✓	✗	✓	✓	✓	✓
Subscribe or unsubscribe another client to a topic based on session properties	✓	✗	✓	✓	✓	✗
Handling subscriptions to routing topics	✗	✗	✓	✓	✓	✗
Handling subscriptions to missing topics	✓	✓	✓	✓	✓	✓
Messaging						
Send a message to a path	✓	✓	✓	✓	✓	✓
Send a message directly to a client	✓	✗	✓	✓	✓	✓
Send a message directly to a client based on session properties	✓	✗	✓	✓	✓	✓

Capability	JavaScript	Apple	Android	Java	.NET	C
Receive direct messages	✓	✓	✓	✓	✓	✓
Handle messages sent to a topic path	✓	✓	✓	✓	✓	✓
Managing security						
Authenticate client sessions and assign roles to client sessions	✗	✗	✓	✓	✓	✓
Configure how the Diffusion server authenticates client sessions and assign roles to client sessions	✓	✗	✓	✓	✓	✓
Configure the roles assigned to anonymous sessions and named sessions	✓	✗	✓	✓	✓	✓
Configure the permissions associated with roles assigned to client sessions	✓	✗	✓	✓	✓	✓
Managing other clients						
Receive notifications about client session events including session properties	✓	✗	✓	✓	✓	✓
Get the properties of a specific client session	✓	✗	✓	✓	✓	✓
Receive notifications about client queue events	✗	✗	✓	✓	✓	✗
Conflate and throttle clients	✗	✗	✓	✓	✓	✗
Close a client session	✗	✗	✓	✓	✓	✗
Push notifications (The Push Notification Bridge must be enabled)						
Receive push notifications	✗	✓	✓	✗	✗	✗
Request that push notifications be sent	✓	✓	✓	✓	✓	✓

Capability	JavaScript	Apple	Android	Java	.NET	C
from a topic to a client						
Publish an update to a topic that sends push notifications	✓	✓	✓	✓	✓	✓
Other capabilities						
Flow control	✗	✗	✓	✓	✓	✓

Protocol support

Each client supports varying transports. A table of the supported transports for each client is presented here.

All protocols supported by Diffusion can be used for both secure (using TLS) and standard connections. For more information, see [SSL and TLS support](#) on page 54.

The following table lists the protocols supported for each client:

Table 3: Supported protocols by client

Client	WebSocket	HTTP Polling	DEPRECATED: DPT	DEPRECATED: HTTP Full Duplex
JavaScript Unified API	✓	✓		
Apple Unified API	✓			
Android Unified API	✓	✓		
Java Unified API	✓	✓	✓	✓
.NET Unified API	✓		✓	✓
C Unified API	✓		✓	
Publisher client	✓		✓	✓

The JavaScript client is fully supported only on certain browsers. Best effort support is provided for other browsers but the software/hardware combination might not be reproducible, particularly for mobile browsers. For more information about supported browsers, see [Browser support](#) on page 56.

SSL and TLS support

Diffusion supports only those SSL versions and cipher suites with no known vulnerabilities.

The following SSL and TLS versions are supported by default:

- SSLv2Hello

- TLSv1
- TLSv1.1
- TLSv1.2

The following cipher suites are supported by default:

- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA

For more information, see [Network security](#) on page 724.

DPT limitations

The following capabilities are not supported in for client sessions that connect to the Diffusion server over DPT:

- Client-proposed reconnection time
- Connection activity monitoring
- Reliable reconnection

In addition, there are some behavior differences between client sessions that connect using DPT and those that connect using other protocols:

- In the case where the roles a client session has are updated and that client no longer has authorization to receive data from a topic, DPT provides CONTROL rather than AUTHORIZATION as the reason for unsubscription.

Table 4: Supported protocols by client

Client	WebSocket	HTTP Polling	DEPRECATED: DPT	DEPRECATED: HTTP Full Duplex	DEPRECATED: HTTP Chunked Streaming
Java Classic API	✓		✓	✓	
.NET Classic API	✓		✓	✓	
JavaScript Classic API	✓ ²	✓	✓ ¹		✓
Flash Classic API		✓	✓		✓
Silverlight		✓	✓		
iOS Classic API	✓		✓		

¹ Supported by Flash®/Silverlight®

² Supported natively and by Flash

³ Recommended

Client	WebSocket	HTTP Polling	DEPRECATED: DPT	DEPRECATED: HTTP Full Duplex	DEPRECATED: HTTP Chunked Streaming
Android Classic API			✓		
C Classic API			✓		

Browser support

Some of the client libraries are intended to be run within browser environments. Diffusion clients can use most commercial browsers and their variants. However, some Diffusion API protocols might not be available.

Diffusion supports the latest release of the following browser versions based on the original Diffusion release date.

Table 5: Supported browsers

Browser	Version
Google Chrome™	53
Mozilla Firefox®	49
Microsoft® Internet Explorer®	11
Apple Safari®	8

- Push Technology runs full regression tests on the browsers and versions documented above for every patch release.
- Push Technology carries out basic validation testing on the latest versions of these browsers but full support is available only at the next minor release.
- Support for older versions of browsers is provided on a best-effort basis, unless specified otherwise.
- Support for other browsers is provided on a best-effort basis.

For details about the operating systems and browsers supported by the Silverlight plugin, refer to the “System Requirements” section on the following web page: [Get Microsoft Silverlight](#)

Mobile browsers

We do not test our JavaScript client libraries with mobile browsers or within mobile applications that wrap a browser application in native code. If you are developing a Diffusion client for a mobile platform, such as iOS or Android, we recommend that you use the provided client libraries for these platforms to develop a native application.

Diffusion JavaScript clients running within a native wrapper or in a mobile browser are supported on a best effort basis and we might not be able to provide support for older versions of the mobile platform.

Browser limitations

Some browsers cannot use all the Diffusion protocols or features. If you experience problems when developing with protocols or client libraries that use the browser, check whether the browser supports this function.

Browser environments are not uniform and some browsers might have functional limitations. It is important to be aware of these limitations when developing for compliance with target browsers.

WebSocket limitations

WebSocket is an Internet Engineering Task Force (IETF) protocol used by Diffusion to provide a full-duplex communication channel over a single TCP connection. It is not supported by all browser versions.

Table 6: Support for WebSocket

Version	WebSocket support?
Internet Explorer 9.0 and earlier	NO
Internet Explorer 10.0 and later	YES (see note)
Firefox	YES
Chrome	YES
Safari	YES
Opera®	YES
iOS	YES
Android	YES

Note: Internet Explorer 11 contains a bug that causes WebSocket connections to be dropped after 30 seconds of inactivity. To work around this problem set the system ping frequency to 25 seconds or less. You can set the system ping frequency in the `Server.xml` configuration file. For more information, see [Server.xml](#) on page 563

Cross-origin resource sharing limitations

CORS allows resources to be accessed by a web page from a different domain. Some browsers do not support this capability.

Table 7: Support for CORS

Version	WebSocket support?
Internet Explorer 9.0 and earlier	NO
Internet Explorer 10.0 and later	YES (see note)
Firefox	YES
Chrome	YES

Version	WebSocket support?
Safari	YES
Opera	YES
iOS	YES
Android	YES

Related concepts

[JavaScript security model](#) on page 658

When a JavaScript client uses the XHR transport, this imposes security constraints on using cross domain requests.

Browser connection limitations

Browsers limit the number of HTTP connections with the same domain name. This restriction is defined in the HTTP specification (RFC2616). Most modern browsers allow six connections per domain. Most older browsers allow only two connections per domain.

The HTTP 1.1 protocol states that single-user clients should not maintain more than two connections with any server or proxy. This is the reason for browser limits. For more information, see [RFC 2616 – Hypertext Transfer Protocol, section 8 – Connections](#).

Modern browsers are less restrictive than this, allowing a larger number of connections. The RFC does not specify how to prevent the limit being exceeded. Either connections can be blocked from opening or existing connections can be closed.

- [Internet Explorer](#)
- [Firefox](#)
- [Chrome](#)
- [Safari](#)
- [Opera](#)
- [iOS](#)
- [Android](#)

Table 8: Maximum supported connections

Version	Maximum connections
Internet Explorer 7.0	2
Internet Explorer 8.0 and 9.0	6
Internet Explorer 10.0	8
Internet Explorer 11.0	13
Firefox	6
Chrome	6
Safari	6
Opera	6
iOS	6

Version	Maximum connections
Android	6

Some Diffusion protocols like HTTP Polling (XHR) use up to two simultaneous connections per Diffusion client. It is important to understand that the maximum number of connections is per browser and not per browser tab. Attempting to run multiple clients within the same browser might cause this limit to be reached.

Reconnection can be used to maintain a larger number of clients than is usually allowed. When TCP connections for HTTP requests are closed, the Diffusion sends another HTTP request which the server accepts. Be aware of cases where Diffusion tries to write a response to closed polling connections before the client can re-establish them. This behavior results in an IO Exception and the Diffusion server closes the client unless reconnection is enabled. When the client tries to re-establish the poll, it is aborted.

Another way to get around browser limits is by providing multiple subdomains. Each subdomain is allowed the maximum number of connections. By using numbered subdomains, a client can pick a random subdomain to connect to. Where the DNS server allows subdomains matching a pattern to be resolved as the same server, tab limits can be mitigated substantially.

Designing your data model

Distribute your data in a data model that fits the needs of your organization and customers.

There are a number of things to consider when designing your data model:

- The structure of your topic tree
- The types of topic to use
- The format of your data
- How you publish data to topics
- Your conflation strategy
- Whether you also use messaging to send data.

These considerations are not separate. The decisions you make about one aspect of your data model affect other aspects.

The data model is defined on the Diffusion server by your publishers or clients. The topic structure, topic types, and data format are not persisted on the Diffusion server through a restart or upgrade.

Design your solution to create your data model on the Diffusion server afresh after the Diffusion server is restarted or upgraded.

Topic tree

Diffusion primarily distributes data using a pub-sub model, where content is published to topics. These topics are arranged as a tree.

What is the topic tree?

The topic tree is the organizational structure of the Diffusion topics. A topic of any type can be created any point in the topic tree where a topic does not already exist.

Locations in the topic tree are referred to by their topic path, which is the level names in the tree that lead to the topic, separated by the slash character (/).

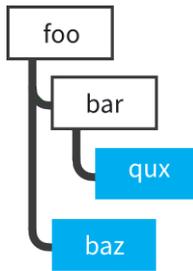


Figure 1: Example topic tree

In the preceding image, topics exist at `baz` and `qux`. The topic path for `baz` is `/foo/baz`. The topic path for `qux` is `/foo/bar/qux`

You can create a topic at `/foo/bar/qux` without having to create topics at `/foo` or `/foo/bar` beforehand.

There can be multiple topics that have the same name, but topic paths are unique.

When interacting with topics in the topic tree, your clients can reference a topic by its topic path or by a *topic selector* with a pattern expression that matches the topic path. Clients can use topic selectors to reference sets of topics in the tree that all match the topic selector's pattern expression.

Considerations when designing your topic tree

- Does the source information have a logical organization?

If the data structure of the source information is complex, it can be mapped to a hierarchical structure.

- How many topics?

If the number of topics is small, a flat topic tree might be appropriate.

- How do clients subscribe to the data?

If there are topics that clients generally subscribe to together, these topics can be organized in the same branch of the topic tree. This enables a client use a topic selector to subscribe to the branch of the topic tree and receive updates for all topics in that branch.

- The size of your topic tree can be constrained by your hardware.

An extremely large topic tree can cause long GC pauses. Ensure that you do sufficient testing with your topic tree before going to production.

If the size of your topic tree structure is caused by a lot of duplication, use routing topics to reduce it.

- A topic can not be bound to the `/` topic path. This is because a topic name must have one or more characters. This means there can be no single topic that acts as the root topic for all possible topics in the topic tree. Instead each top-level topic whose path contains a single part acts as the root topic for their branch of the topic tree.

However, the `/` topic path can be used as a routing path when sending or receiving messages, which uses paths but does not use any topics that are bound to them.

Related concepts

[Topic selectors in the Unified API](#) on page 61

A topic selector identifies one or more topics. You can create a topic selector object from a pattern expression.

[Topic selectors in the Classic API \(deprecated\)](#) on page 69

A topic selector is a string that can be used by the Classic API to select more than one topic by indicating that subordinate topics are to be included or by fuzzy matching on topic names or both.

[Topic selectors in the Classic API \(deprecated\)](#) on page 69

A topic selector is a string that can be used by the Classic API to select more than one topic by indicating that subordinate topics are to be included or by fuzzy matching on topic names or both.

Topic naming

Consider the following restrictions when deciding on your topic names.

Restricted characters

A topic name can be made up of one or more Unicode characters but must not contain any of the restricted characters mentioned below. The topic path is made up of the names of all topics in its path separated by the slash character (/).

The slash character (/) and the exclamation mark (!) are not permitted in any topic names.

Classic API restricted characters

In addition to the slash character (/) and the exclamation mark (!), which are restricted in all topic names, the following characters are not permitted in topic names for topics that are accessed by any clients that use the Classic API (deprecated):

Table 9: Restricted characters for topics used by Classic API clients.

Character	Reason for restriction
<code>[]\^\$.?*(+)</code>	These are all metacharacters used in regular expressions. Any topic String that contains any of these characters is assumed to be a topic selector. These characters cannot be used in topic names.
Control/Reserved	No characters with a hexadecimal value of less than 0x2D. This includes some punctuation characters such as comma (,).
Whitespace	No characters defined as whitespace in Java (as indicated by the <code>isWhiteSpace</code> method of the Java Character class).

Reserved spaces

The Diffusion branch and the @ branch of the topic tree are reserved for internal use.

Recommendations

Although all Unicode characters (other than the restricted ones mentioned above) are supported to allow for language variations it is highly recommended that only alphanumeric characters are ever used in topic names. Hyphen (-) or underscore (_) can be used as break characters.

Topic selectors in the Unified API

A topic selector identifies one or more topics. You can create a topic selector object from a pattern expression.

Supported in: JavaScript Unified API, Java Unified API, .NET Unified API, Apple Unified API, Android Unified API, C Unified API

Pattern expressions

Use pattern expressions to create a topic selector of one of the types described in the following table. The type of the topic selector is indicated by the first character of the pattern expression.

Table 10: Types of topic selector

Topic selector type	Initial character	Description
Path	>	<p>A path pattern expression must contain a valid topic path. A valid topic path comprises topic names separated by path separators (/). A topic name comprises one or more UTF characters except for slash (/).</p> <p>A PATH selector returns only the topic with the given path. See Path examples on page 63</p>
Abbreviated path	Any character except the following: <ul style="list-style-type: none">• Hash symbol (#)• Question mark (?)• Greater than symbol (>)• Asterisk (*)• Dollar sign (\$)• Percentage symbol (%)• Ampersand (&)• Less than symbol (<)	<p>An abbreviated path pattern expression is an alternative syntax for a path pattern selector. It must be a valid topic path.</p> <p>A valid topic path comprises topic names separated by path separators (/). A topic name comprises one or more UTF characters except for slash (/).</p> <p>Abbreviated path pattern expressions are converted into PATHselectors and return only the topic with the given path. See Abbreviated path examples on page 64</p>
Split-path	?	<p>A split-path pattern expression contains a list of regular expressions separated by the / character. Each regular expression describes a part of the topic path. A SPLIT_PATH_PATTERN selector returns topics for which each regular expression matches the part of the topic path at the corresponding level. See Split-path examples on page 65</p>
Full-path	*	<p>A full-path pattern expression contains a regular expression. A FULL_PATH_PATTERN selector returns topics for which the regular expression matches the full topic path. See Full-path examples on page 66</p> <p>Note: Full-path pattern topic selectors are more powerful than split-path pattern topic selectors, but are evaluated less efficiently at the server. If you are combining expressions, use selector sets instead.</p>
Selector set	#	<p>A selector set pattern expression contains a list of selectors separated by the separator ///. A SELECTOR_SET topic selector returns topics that match any of the selectors.</p>

Topic selector type	Initial character	Description
		<p>Note: Use the <code>anyOf ()</code> method for a simpler method of constructing <code>SELECTOR_SET</code> topic selectors.</p> <p>See Selector set examples on page 66</p>

Regular expressions

Diffusion topic selectors use regular expressions. Any regular expression can be used in split-path and full-path patterns, with the following restrictions:

- It cannot be empty
- In split-path patterns, it cannot contain the path separator (/)
- In full-path patterns, it cannot contain the selector set separator (////)

Depending on what the topic selector is used for, regular expressions in topic selectors can be evaluated on the client or on the Diffusion server. For more information, see [Regular expressions](#) on page 67.

Descendant pattern qualifiers

You can modify split-path or full-path pattern expressions by appending a descendant pattern qualifier. These are described in the following table:

Table 11: Descendant pattern qualifiers

Qualifier	Behavior
None	Select only those topics that match the selector.
/	Select only the descendants of the matching topics and exclude the matching topics.
//	Select both the matching topics and their descendants.

Topic path prefixes

The topic selector capabilities in the Unified API provide methods that enable you to get the topic path prefix from a topic selector.

A topic path prefix is a concrete topic path to the most specific part of the topic tree that contains all topics that the selector can specify. For example, for the topic selector `?foo/bar/baz/.*/bing`, the topic path prefix is `foo/bar/baz`.

The topic path prefix of a selector set is the topic path prefix that is common to all topic selectors in the selector set.

Path examples

The following table contains examples of path pattern expressions:

Expression	Matches alpha/beta?	Matches alpha/beta/gamma?	Notes
>alpha/beta	yes	no	

Expression	Matches alpha/beta?	Matches alpha/beta/gamma?	Notes
>/alpha/beta/	yes	no	This pattern expression is equivalent to the pattern expression in the preceding row. In an absolute topic path, single leading or trailing slash characters (/) are removed because the topic path is converted to canonical form. A path pattern expression can return a maximum of one topic. The trailing slash in this example is not treated as a descendant qualifier and is removed.
>alpha/beta/gamma	no	yes	
>beta	no	no	The full topic path must be specified for a path pattern expression to match a topic.
>.*/*	no	no	For clients using the Unified API, the period (.) and asterisk (*) characters are valid in topic names. In a path pattern expression these characters match themselves and are not evaluated as part of a regular expression.

Abbreviated path examples

The following table contains examples of abbreviated path pattern expressions:

Expression	Matches alpha/beta?	Matches alpha/beta/gamma?	Notes
alpha/beta	yes	no	
/alpha/beta/	yes	no	This pattern expression is equivalent to the pattern expression in the preceding row. In an absolute topic path, single leading and trailing slash

Expression	Matches alpha/beta?	Matches alpha/beta/gamma?	Notes
			characters (/) are removed because the topic path is converted to canonical form. A path pattern expression can return a maximum of one topic. The trailing slash in this example is not treated as a descendant qualifier and is removed.
alpha/beta/gamma	no	yes	
beta	no	no	The full topic path must be specified for a path pattern expression to match a topic.

Split-path examples

The following table contains examples of split-path pattern expressions:

Expression	Matches alpha/beta?	Matches alpha/beta/gamma?	Notes
?alpha/beta	yes	no	
?alpha/beta/	no	yes	The trailing slash character (/) is treated as a descendant pattern qualifier in split-path pattern expressions. It returns descendants of the matching topics, but not the matching topics themselves.
?alpha/beta//	yes	yes	Two trailing slash characters (//) is treated as a descendant pattern qualifier in split-path pattern expressions. It returns matching topics and their descendants.
?alpha/beta/gamma	no	yes	
?beta	no	no	
?.*	no	no	Each level of a topic path must have a regular expression specified for it for a split-path pattern expression to match a topic.
?.*/*	yes	no	
?alpha/.*/	yes	yes	In this pattern expression, "alpha/.*" matches all topics in the alpha branch of the topic tree. The descendant

Expression	Matches alpha/beta?	Matches alpha/beta/gamma?	Notes
			pattern qualifier (//) indicates that the matching topics and their descendants are to be returned.

Full-path examples

The following table contains examples of full-path pattern expressions:

Expression	Matches alpha/beta?	Matches alpha/beta/gamma?	Notes
*alpha/beta	yes	no	
*alpha/beta/gamma	no	yes	
*alpha/beta/	no	yes	The trailing slash character (/) is treated as a descendant pattern qualifier in split-path pattern expressions. It returns descendants of the matching topics, but not the matching topics themselves.
*alpha/beta//	yes	yes	Two trailing slash characters (//) is treated as a descendant pattern qualifier in split-path pattern expressions. It returns matching topics and their descendants.
*beta	no	no	In a full-path pattern selector the regular expression must match the full topic path for a topic to be matched.
*.*beta	yes	no	The regular expression matches the whole topic path including topic separators (/).

Selector set examples

Use the `anyOf` methods to create a `SELECTOR_SET TopicSelector` object.

The following example code shows how to use the `anyOf(TopicSelector... selectors)` method to create a selector set topic selector:

```
// Use your session to create a TopicSelectors instance
TopicSelectors selectors = Diffusion.topicSelectors();

// Create topic selectors for the individual pattern expressions
TopicSelector pathSelector = selectors.parse(">foo/bar");
TopicSelector splitPathSelector = selectors.parse("?f.*\/bar\d+");
TopicSelector fullPathSelector = selectors.parse("*f.*\d+");

// Use the individual topic selectors to create the selector set
// topic selector
TopicSelector selector = selectors.anyOf(pathSelector,
splitPathSelector, fullPathSelector);
```

```
// Use the topic selector as a parameter to methods that perform
actions on topics or groups of topics
```

The following example code shows how to use the `anyOf(String... selectors)` method to create the same topic selector as in the previous code example, but in fewer steps:

```
// Use your session to create a TopicSelectors instance
TopicSelectors selectors = Diffusion.topicSelectors();

// Create the selector set topic selector by passing in a list of
// pattern expressions
TopicSelector selector = selectors.anyOf(">foo/bar", "?f.*/bar\d+",
    "*f.*\d+");

// Use the topic selector as a parameter to methods that perform
actions on topics or groups of topics
```

Related concepts

[Topic tree](#) on page 59

Diffusion primarily distributes data using a pub-sub model, where content is published to topics. These topics are arranged as a tree.

[Topic selectors in the Classic API \(deprecated\)](#) on page 69

A topic selector is a string that can be used by the Classic API to select more than one topic by indicating that subordinate topics are to be included or by fuzzy matching on topic names or both.

Regular expressions

Depending on what the topic selector is used for, regular expressions in topic selectors can be evaluated on the client or on the Diffusion server. On the Diffusion server, regular expressions are evaluated as Java-style regular expressions. On clients, regular expressions are evaluated according to the conventions of the client library.

The following client uses of topic selectors are evaluated on the Diffusion server:

- Subscribing to topics
- Unsubscribing from topics
- Subscribing another client to topics
- Unsubscribing another client from topics
- Fetching topic states
- Removing topics

The following client uses of topic selectors are evaluated on the client:

- Creating a stream to receive updates published to topics
- Creating a stream to receive messages sent on topic paths

The regular expression evaluation on each of the client libraries and on the Diffusion server are all based on the same style of regular expressions. The behavior of topic selectors on the clients and on the Diffusion server is the same for all standard uses of regular expressions. More advanced or complex regular expressions might differ slightly in behavior.

See the following sections for platform-specific information.

On the Diffusion server

Topic selectors evaluated on the Diffusion server are evaluated as Java-style regular expressions and are based on `java.util.regex.Pattern`.

For more information about Java-style regular expressions, see [Java regular expressions](#).

On Java and Android clients

Topic selectors evaluated on these clients are evaluated as Java-style regular expressions. There are no differences between how a regular expression is evaluated on these clients and how it is evaluated on the Diffusion server.

For more information about Java-style regular expressions, see [Java regular expressions](#).

On .NET clients

Topic selectors evaluated on the .NET client are evaluated as .NET Framework regular expressions, these are compatible with Perl 5 regular expressions. For more information about .NET regular expressions, see [https://msdn.microsoft.com/en-us/library/az24scfc\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/az24scfc(v=vs.110).aspx).

The .NET evaluation of regular expressions can differ from the Java evaluation of the same regular expression on the Diffusion server. For examples of how these can differ, see <http://stackoverflow.com/a/545348>.

Ensure that you test the behavior of complex regular expressions that you use with the .NET client.

On Apple clients

Topic selectors evaluated on the Apple client are evaluated according to the `NSRegularExpression` class, which uses ICU regular expression syntax. For more information about Apple regular expressions, see:

- https://developer.apple.com/library/ios/documentation/Foundation/Reference/NSRegularExpression_Class/
- <http://userguide.icu-project.org/strings/regexp>
-

The Apple evaluation of regular expressions can differ from the Java evaluation of the same regular expression on the Diffusion server. Ensure that you test the behavior of complex regular expressions that you use with the Apple client.

On JavaScript clients

Topic selectors evaluated on the JavaScript client are based on the ECMAScript standard. For more information about JavaScript regular expressions, see:

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions
- <http://www.ecma-international.org/ecma-262/5.1/#sec-7.8.5>

The JavaScript evaluation of regular expressions can differ from the Java evaluation of the same regular expression on the Diffusion server. Ensure that you test the behavior of complex regular expressions that you use with the JavaScript client.

On C clients

Topic selectors evaluated on the C client use PCRE. For more information about C regular expressions, see <http://www.pcre.org/>.

The C evaluation of regular expressions can differ from the Java evaluation of the same regular expression on the Diffusion server. Ensure that you test the behavior of complex regular expressions that you use with the C client.

Topic selectors in the Classic API (deprecated)

A topic selector is a string that can be used by the Classic API to select more than one topic by indicating that subordinate topics are to be included or by fuzzy matching on topic names or both.

Supported in: JavaScript Classic API, Java Classic API, .NET Classic API, C Classic API, iOS Classic API, Android Classic API, Flash Classic API, Silverlight Classic API

Including subordinate topics

When specifying a topic name you can also indicate that all of its subordinate topics are to be included by suffixing the name with a slash character (/).

For example, to select all of the subordinate topics of a topic named MyTopic use a selector of the format `MyTopic/`.

This notation can also be used with topic paths. So to select all topics subordinate to the topic named A/B, use a selector of the following format:

`A/B/`

Specifying a suffix of “/” does not include the topic named prior to the final “/”. To include the specified topic and all of its subordinates, use “//”.

For example, to select the topic A/B and all of its subordinates, specify the following selector:

`A/B//`

Fuzzy matching

A number of topics can be selected using a single topic selector which uses regular expressions to match against topic names.

Regular expressions provide a powerful mechanism for String pattern matching which is not discussed here. A Java tutorial is available for those familiar with Java or a more generic tutorial can be consulted for other language users. The regular expression syntax supported by Diffusion is defined by the Java Pattern class.

The important point to note about the use of regular expressions in topic selectors is that they have multiple parts (separated by /) and that each part of a selector can be a regular expression pattern. A multi-part selector is evaluated part by part starting from the top level of the topic tree in an attempt to find a matching topic in the tree.

If there is no “/” in the selector string but there are regex characters then the whole string is applied as a regex against the full topic name. This can be useful in some circumstances, for example when you do not know how many topic levels exist but it is far less efficient than per-level matching.

Example of selector regular expression processing

Consider the following topic selector:

`A.*B/. *X`

As the above selector has three specifications it only matches with topic names with three parts.

The first check is to select any topics at the top-level of the topic tree whose name matches the pattern “A.*”. In regular expression notation this pattern matches with any String that starts with “A” and is followed by zero or more other characters. So if there are no top-level topics that start with an “A” then this selector matches with no topics at all. However, for any top-level topic that does match, processing goes to the next part which is a simple String “B” and only matches with a subordinate topic called “B”. So if any topics called “B” are found under the top-level topics starting with “A”, processing goes onto the final pattern “.*X” which in regex notation indicates any String with any number of characters before a final “X” (a String ending with “X”).

So to summarize, the above selector matches only with topic paths with 3 elements where the first element starts with an “A”, the second element is “B” and the third element ends with an “X”.

The following are matches:

Accounts/B/TAX

A/B/X

Admin/B/ProjectX

But the following are not:

Accounts/TAX

Admin/B/ProjectYHR/B/TAX

If you do not know how many levels you are dealing with but want to select topics where the lowest level topic name is ABC, you must use a whole topic name selector as follows:

.*\x2FABC

The use of “\x2F” is necessary to represent a “/” as otherwise the selector evaluates per level.

So this selector matches with:

A/B/ABCX/Y/ZZZ/wwwww/ABC

but not with:

A/B/CABC

Mixed Mode selectors

It is permitted to mix regex pattern handling and subordinate topic suffixes in the same topic selector pattern.

So it is permitted to use a selector of the form A.* /Address/ which has the effect of selecting all of the topics subordinate to the topic named Address within any top level topic starting with “A”.

Selector examples

Table 12: Selector examples

Selector pattern	Selects	Examples
A/	All topics subordinate to, but not including the top-level topic named “A”.	A/B, A/B/C
A//	The top level topic named “A” and all topics subordinate to it.	A, A/B, A/B/C
A.*	All topics that start with the letter “A”	A, Accounts, Admin
A.*//	All topics that start with the letter “A” and their subordinates.	A, Accounts, A/B, Admin/X/Y
Counties/. *ex	All topics under the top-level topic called Counties whose name ends in “ex”.	Counties/Middlesex, Counties/Sussex

Selector pattern	Selects	Examples
.*folk	All topics whose full name ends in “folk” regardless of level.	UK/Counties/Suffolk, Counties/Norfolk

When is a topic string a selector?

In certain parts of the APIs only topic names can be specified (for example, when adding a topic) but in other areas selectors are allowed (for example, in subscription).

A topic string is considered to be a selector if it is terminated by a “/” or it contains any one of the regular expression metacharacters which are defined as the following characters: [] \ ^ \$. | ? * + ()

Related concepts

[Topic tree](#) on page 59

Diffusion primarily distributes data using a pub-sub model, where content is published to topics. These topics are arranged as a tree.

[Topic selectors in the Unified API](#) on page 61

A topic selector identifies one or more topics. You can create a topic selector object from a pattern expression.

Topics

Consider the types of topic you want to use and how. You can also consider the attributes that topics have. The attributes a topic can have can change depending on the topic type.

Topics that provide data

You can publish data to these topics and the data is streamed to subscribing clients.

- [JSON](#)
- [Binary](#)
- [Single value](#)
- [Record](#)
- [Stateless](#)

Advanced topics

Diffusion includes advanced topic types that provide additional capabilities such as routing subscriptions to other topics or presenting data in paged form.

JSON topics

A topic that provides data in JSON format. The data is transmitted in a binary form for increased efficiency. JSON topics are stateful: their state is maintained on the Diffusion server.

Note: JSON topics are fully supported by the JavaScript Unified API, Android Unified API, Java Unified API, and .NET Unified API,

Why use a JSON topic?

JSON provides the ability for you to structure your data in a human-readable, industry-standard format. For more information about JSON, see <http://www.json.org/>.

The value of the topic is transmitted as CBOR. For more information about CBOR, see <http://cbor.io/>.

The value of the topic is accessible both as JSON and CBOR.

Why use a JSON topic?

JSON is a human-readable, industry-standard format for your data. JSON is natively supported by JavaScript and there are third-party libraries available for other platforms.

A JSON topic enables multiple fields to be maintained in the same topic as part of a composite data type. All updates made at the same time to parts of a JSON topic are sent out to the client together. This enables a set of parts to be treated as a transactional group.

Deltas of change are calculated at the Diffusion server such that only those parts that have changed since the last update are sent out to the subscribed clients. This ensures that the minimum amount of data is sent to clients.

The current value of the topic is cached on the client. When deltas are sent, the client can automatically apply these deltas to the value to calculate the new value.

If your data structure is too complex to be represented by a topic tree or might make the topic tree structure difficult to manage, it might be more appropriate to represent part of the data structure inside a JSON topic as JSON objects.

Properties of a JSON topic

When you create a JSON topic you can specify the following properties in the topic specification:

PUBLISH_VALUES_ONLY

If set to `true`, all values are published in full. Otherwise, deltas are published if they are more efficient to send than the full values.

If there is little or no relationship between values published to a topic, delta streams will not reduce the amount of data transmitted. For such topics, it is better to set `PUBLISH_VALUES_ONLY`.

VALIDATE_VALUES

If set to `true`, the topic checks that the update made to it by an updating client is a valid instance of the JSON datatype. If the update is not valid, the update is discarded and the updating client notified of the failure.

If set to anything other than `true`, no validation is performed and all values are streamed to subscribing clients whether they are valid JSON data or not.

Validation has a performance overhead and is disabled by default.

Considerations when using a JSON topic

JSON is only native to JavaScript. Other languages must parse it as text. However, there are many third-party libraries you can use for this parsing.

JSON does not support binary data. If you want to use binary data, you can use binary topics. For more information, see [Binary topics](#) on page 73.

Classic API clients (deprecated) do not support any interaction with JSON topics.

JSON topics are currently fully supported by only the following APIs:

- JavaScript Unified API
- Java Unified API
- Android Unified API
- .NET Unified API

The following APIs offer limited support:

- C Unified API
- Apple Unified API

Use this API to create JSON topics and to receive data from JSON topics. Updating JSON topics is not yet fully supported.

The Publisher API provides the capability to create and update JSON topics using `TopicDataFactory.newUniversalData`.

If you use value streams to receive updates from a JSON topic, this can cause problems in the event of client reconnection. We recommend that when using value streams, you configure your clients not to attempt to reconnect to a session, but instead to connect again with a new session and resubscribe to topics.

Binary topics

A topic that streams binary data as bytes and uses efficient binary deltas to stream only the data that changes between updates. Binary topics are stateful: their state is maintained on the Diffusion server.

Note: Binary topics are fully supported by the JavaScript Unified API, Android Unified API, Java Unified API, and .NET Unified API,

Why use a binary topic?

You can use a binary topic to transmit any arbitrary binary data without the overhead of encoding it to a string or the risk of the binary data being incorrectly escaped.

Binary topics can use binary deltas to send only the data that has changed when this is more efficient than sending the full value.

You can use a binary topic to transmit very large strings. This enables a client to use the binary delta capability to transmit only the changed parts of a string rather than the whole value. This reduces the amount of data transmitted over the network.

Binary formats, such as Google protocol buffers, can be streamed using a binary topic. Though Diffusion provides a Google protocol buffer topic type, the protocol buffer topic requires that a compiled `.proto` class must be present on the Diffusion server classpath. We recommend that you use a binary topic instead and handle the serialization and deserialization of the protocol buffers in your clients.

Properties of a binary topic

When you create a binary topic you can specify the following properties in the topic specification:

PUBLISH_VALUES_ONLY

If set to `true`, all values are published in full. Otherwise, deltas are published if they are more efficient to send than the full values.

If there is little or no relationship between values published to a topic, delta streams will not reduce the amount of data transmitted. For such topics, it is better to set `PUBLISH_VALUES_ONLY`.

Considerations when using a binary topic

Data on binary topics contains no implicit information about its structure.

Data on binary topics cannot be viewed in the console.

Classic API clients (deprecated) do not support any interaction with binary topics.

Binary topics are currently fully supported by only the following APIs:

- JavaScript Unified API

- Java Unified API
- Android Unified API
- .NET Unified API

The following APIs offer limited support:

- C Unified API
- Apple Unified API

Use this API to create binary topics and to receive data from binary topics. Updating binary topics is not yet fully supported.

The Publisher API provides the capability to create and update JSON topics using `TopicDataFactory.newUniversalData`.

If you use value streams to receive updates from a binary topic, this can cause problems in the event of client reconnection. We recommend that when using value streams, you configure your clients not to attempt to reconnect to a session, but instead to connect again with a new session and resubscribe to topics.

Single value topics

A topic that streams data as a single value that can be constrained to a defined data type. Single value topics are stateful: their state is maintained on the Diffusion server.

The value of the topic state is stored as a string. However, the type of the single value can be constrained in certain ways, for example, to hold a string, an integer, or a decimal number. The type of the single value is described using field metadata in the schema. For more information, see [Metadata](#) on page 75.

Why use a single value topic?

For the majority of use cases, single value topics are the most appropriate model for your data. Single value topics are easy to create and update, and the data the topic contains is simpler.

If a structure is required for your data, you can use the design of your topic tree to define the structure.

Single value topics are supported by all client APIs.

By defining the type of the single value of the topic, you benefit from automatic validation and formatting of the data.

Considerations when using a single value topic

A single value topic can only hold textual data. This type of topic data cannot be used to publish non-textual data, such as a PNG or PDF file.

Single value topics cannot be used to make multiple updates transactionally. If you have multiple items of data that you want to publish at the same time and have received by subscribing clients at the same time, you cannot split these items of data across multiple single value topics. In this case, it is more appropriate to use a topic with a composite data type such as a JSON topic, CBOR topic, or record topic. A topic with a composite data type can contain fields for each of the data items.

If you use value streams to receive updates from a single value topic, this can cause problems in the event of client reconnection. We recommend that when using value streams, you configure your clients not to attempt to reconnect to a session, but instead to connect again with a new session and resubscribe to topics.

Record topics

A topic that streams data in Diffusion record format. Record format comprises strings separated by field or record delimiters or both. Record topics are stateful: their state is maintained on the Diffusion server.

Describe the layout of the data by using content metadata in the schema. For more information, see [Metadata](#) on page 75.

Why use a record topic?

A record topic enables multiple fields to be maintained in the same topic as part of a composite data type. All updates made at the same time to fields on a record topic are sent out to the client together. This enables a set of fields to be treated as a transactional group.

Deltas of change are calculated at the server such that only those fields that have changed since the last update are sent out to the subscribed clients. This ensures that the minimum amount of data is sent to clients.

Considerations when using a record topic

You must define the metadata when you create the record topic. This is more complex than using a JSON or CBOR topic for composite data.

The metadata format used by record topics is only used by Diffusion. If you require a topic with a composite data type, you can use JSON or CBOR topics

Record topic updates represent all possible records and fields within the content. Fields that have not changed are sent within delta updates as zero-length strings. Because unchanged fields are represented this way a client cannot differentiate between a field that has not changed and an empty field. You can specify a special character that is used to represent an empty field.

The current value of a record topic is not cached on the client. Because of this, deltas are not processed automatically and your application code must cache the topic value and correctly apply incoming deltas. This lack of caching also requires that a client must register a stream against a topic before subscribing to the topic in order to receive a value. If a client subscribes to a record topic then registers a stream, the client receives only deltas until the next time a full value is published.

Metadata

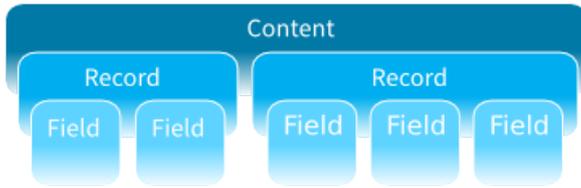
Metadata defines how data is formatted when it is published on a topic. Define the metadata structure for a topic that describes the grouping, order, type, and multiplicity of data items published on a topic.

Updates and messages contain byte data. This byte data can be formatted in whatever way your application requires. For example,

- When creating a record topic, define a metadata structure that describes the data format both for updates published on that topic and those sent on to the subscribing clients.
- When creating a single value topic, you can define field metadata that constrains the data type that updates published to the topic can have.
- When sending a message through a topic path, you can use metadata to create the content of your message.

Metadata structure

The metadata structure is made up of nested records and fields. The outer container is the *content*. This contains one or more *records*. Each record can contain one or many *fields*.



Fields and records are identified by a name. Every record must have a name that is unique within the content. Every field must have a name that is unique within the enclosing record.

Every field or record defined in the metadata structure can represent one or more possible occurrences of that field or record in the byte data. The number of possible occurrences of a record or field is described by its multiplicity.

The order in which records and fields are defined within their enclosing container defines the order that they appear in byte data.

Field metadata

A metadata field defines an elementary data item within a record.

Every field has the following properties:

- [Data type](#)
- [Multiplicity](#)
- [Default value](#)

Data type

The data type of a field defines its actual representation within the byte data. The following table describes the data types that are available.

Table 13: Data types for metadata fields

Data type	Description	Default
String	A character string.	Zero-length string
Integer string	An integer represented in the content as a character string. If a field is defined as this type, it can only contain numeric digits with an optional leading sign. Fields of this type cannot be empty.	0
Decimal string	A decimal number represented in the content as a character string. Decimal fields have the number of places to the right of the decimal point defined by the scale, the default being 2. Such values can be parsed from a character string with any number of digits to the right of the decimal point. Half-up rounding is applied to achieve the target scale. Output of the field is rendered with the specified scale. Fields of this type cannot be empty. For comparison purposes the scale is ignored: a value of 1.50 is the same as 1.5.	0.00 (depending on scale)

Data type	Description	Default
Custom string	This is a special type where the behavior is delegated to a user-written custom field handler. This type is available in all topic data types.	-

Multiplicity

The multiplicity of a metadata field or record defines the number of times the corresponding byte data can occur within the enclosing record or content.

Multiplicity is defined in terms of the minimum and maximum number of occurrences. Some byte data representations support variable numbers of records and field, whereas others (such a record data) only support fixed number of records and fields (where minimum=maximum) except in the last position.

Fixed multiplicity is defined by a single number. For example, a multiplicity of 5 on a field indicates that there must be exactly five occurrences of the field within its enclosing record.

Variable multiplicity is defined by a minimum value and a maximum value and is represented with the notation *n..n*. For example, multiplicity of 1..5 on a field specifies that there must be between one and five occurrences of the field within its enclosing record.

A special maximum value of -1 is used to represent no maximum. For example, a multiplicity of 1..-1 on a field specifies there can be any number of occurrences of the field, but there must be at least one.

Optional nodes are defined by a minimum value of 0. For example, a multiplicity of 0..1 on a field specifies that there can be zero or one occurrences of the field within its enclosing record. A fixed multiplicity of 0 is not allowed.

Variable multiplicity fields must be defined at the end of their containing record. Variable multiplicity records must be defined at the end of the content.

Default value

You can specify a default value for a field. If you do not specify a default value, the default value for the data type is used. When content is created using metadata, default initialization applies the default values specified for each field.

Stateless topics

A topic that has no state held on the Diffusion server or on the client that publishes to it.

A stateless topic has no state. It can be used for publishing and receiving updates, but not for fetching the current topic state.

Stateless topics are the only type of topic provided in other typical pub-sub solutions.

Why use a stateless topic?

You can use stateless topics for data streams where there is no current state of the data, only updates. For example, a feed of news items.

All handling of the topic and topic data of a stateless topic is done by your client application. Because of this, the format of the data published on a stateless topic is entirely flexible. The topic content is treated as byte data by the Diffusion server. How that byte data is handled and interpreted is determined by your client applications.

Considerations when using a stateless topic

A stateless topic does not store state. You cannot fetch the topic state and when you first subscribe to a stateless topic you do not receive the topic state as a value as you do with a stateful topic.

You must write all of the logic in your client to handle the byte data that is published on a stateless topic. This might mean it takes longer to get started using a stateless topic compared to topics that are handled by the Diffusion server.

You also lose some of the benefits of having a topic whose content is understood by the Diffusion server, such as validation, formatting, conflation, and deltas.

Advanced topics

Advanced topics are those types of topic that either provide data in a tabular form, do not provide data directly, or provide additional capabilities.

Advanced topics can be divided into the following categories:

Advanced data-providing topics

You can publish data to these topics, but they require additional configuration on the Diffusion server:

- [Protocol buffer](#)
- [Custom](#)

Functional topics

You cannot publish data to these topics. Instead, functional topics provide other capabilities to subscribing clients, for example notifications.

The following types of topic are functional topics:

- [Slave](#)
- [Routing](#)
- [DEPRECATED: Child list](#)
- [DEPRECATED: Service](#)
- [DEPRECATED: Topic notify](#)

DEPRECATED: Paged topics

You can transmit data through these topics, but paged topics do not use the same pub-sub mechanism as publishing topics.

Paged topics have state which is stored in a tabular format, as pages of lines. Unlike publishing topics, clients do not publish updates to a paged topic to change the state. Instead clients add, update, or remove lines in the topic.

Paged topics can be ordered or unordered. You can define the ordering of a paged topic by using a user-defined comparator class that is located on the Diffusion server or by declaring the rules that are used for the ordering when you create the topic. Lines that are added to unordered paged topics are added at the end. Lines that are added to ordered paged topics are added at the position defined by the comparator or rules.

A client must subscribe to a paged topic to be able to access the data on it. However, unlike publishing topics clients do not receive updates whenever changes are made to the paged topic state. To access the data, the client must open a view on the paged topic and specify how many lines per page and what page to open the view on. The client can then page through the data. If the state of the client's current page changes, the client is notified and can choose to refresh the page.

The following types of topic are paged topics:

- [Paged string](#)
- [Paged record](#)

Routing topics

A special type of topic, which can map to a different real topic for every client that subscribes to it. In this way, different clients can see different values for what is effectively the same topic from the client point of view.

When a client subscribes to a routing topic, the request is either passed to a client that has registered as a routing subscription handler for the topic or handled by a server-side routing handler. The routing handler assigns a linked topic to represent it to that client.

The routing handler can assign a different linked topic to each client that subscribes to the routing topic. The linked topic can be a topic of one of the following types:

- JSON
- Binary
- Single value
- Record
- Stateless
- Protocol buffer
- Custom

When updates are received on the linked topic, those updates are propagated through the routing topic to the subscribing clients.

The subscribing client is not aware of the linked topic. It is subscribed to the routing topic and all the updates that the client receives contain only the routing topic path and not the linked topic path.

Why use a routing topic?

Use routing topics when you want your subscribing clients to all have the same subscription behavior, but the data they receive to be decided by a routing handler depending on criteria about that client.

For example, your subscribing clients can subscribe to a routing topic called Price, but the routing handler assigns each client a different linked topic depending on the client's geographic location. This way, clients in different countries can act in the same way, but receive localized information.

Considerations when using a routing topic

Using routing topics requires that you write a routing handler that is either hosted on the server or registered by a client with the required permissions. The following client APIs can register a routing handler: Java, .NET, or Android Unified API.

A subscribing client only needs permission to subscribe to the routing topic. Permission to subscribe to the linked topic is not required.

If the linked topic is removed, subscribing clients are automatically unsubscribed from the routing topic.

If you attempt to fetch from a routing topic that routes to a stateless topic, no data is returned.

You cannot use topic replication to replicate routing topics between Diffusion servers.

When using automatic fan-out to propagate topics from a primary server to one or more secondary servers, the routing subscription handlers for a routing topic must be registered at the primary and all secondary servers. The routing logic provided by the handlers on the primary and secondary server must be identical.

Slave topics

A special type of topic that has no state of its own but is a reference to the state of another topic.

A slave topic acts as an alias to another topic, the master topic. Updates published to the master are fanned out to subscribers of the slave. The slave cannot be updated directly. The master topic must be one of the following types of topic:

- JSON
- Binary
- Single value
- Record
- Protocol buffer
- Custom

The link between a slave topic and a master topic is defined when the slave topic is created. This is different to routing topics where the link between topics is defined when a client subscribes.

Why use a slave topic?

You can use slave topics to provide the same data from multiple topic paths and manage the topics from only one topic.

You can use a slave topic to act as a redirection to a succession of master topics. For example, you can create a slave topic called `latest` that is linked to a master topic where data is published about a current event. When that event is no longer current, you can remove the slave topic and recreate it now linked to the master topic where data is published about what is now the current event.

The subscribing clients can subscribe to the `latest` slave topic and they continue to be subscribed to the slave topic and receive the latest data, even as the master topic that provides the data changes.

Considerations when using a slave topic

A client only needs permissions on the slave topic. Permission to subscribe to the linked topic is not required.

More than one slave can point to the same master topic.

A slave topic cannot also act as a master topic to another slave topic.

Removing a master topic causes all linked slave topics to be removed.

When using topic replication to replicate slave topics between Diffusion servers, be aware that a replicated slave topic is linked to a master topic located on the same Diffusion server as the replicated slave topic. This is true whether that master topic is created by replication or directly. If the master topic does not exist on the Diffusion server that a slave topic is replicated to, the slave topic is not created. Instead it is added to a pending set of topics. If the master topic is subsequently created by replication or directly, the slave topic is removed from the pending set and is created on the Diffusion server.

When using automatic fan-out to propagate topics from a primary server to one or more secondary servers, be aware that the order of topic creation on the secondary server can prevent slave topics from being replicated. For example, if a slave topic refers to a topic that does not yet exist because it is in a branch not yet replicated or because it is lower down the link hierarchy.

DEPRECATED: Custom topics

A topic that has its state maintained at the server by a user-written Diffusion class.

By implementing `CustomTopicDataHandler`, you can define the types, formatting, and structure data on a custom topic. By writing a custom topic handler you can define how that data is maintained, compared, and sent to subscribing clients.

These user-written classes must be on the classpath of the Diffusion server.

The topic state is delegated to an instance of the class and is handled on the server side. Updates from a client are passed to the custom topic handler for processing. The custom topic handler can hold the topic state internally or elsewhere.

Why use a custom topic?

Custom topics enable you to use data types and layouts that are not supported directly by Diffusion.

Considerations when using a custom topic

The custom topic handler must be present on the classpath of the Diffusion server. If your solution uses multiple Diffusion servers that all host the same custom topic, the same custom topic handler must be present on all of their classpaths.

Custom topic handlers can only be implemented in Java.

DEPRECATED: Topic notify topics

A functional topic that can be used by clients to receive notifications whenever topics are created, modified, or removed in a part of the topic tree.

A client can subscribe to a topic notify topic to request notifications that contain one of the following levels of notification when a topic is added:

- topic name and type
- topic name, type, and properties
- topic name, type, and metadata
- full topic definition (including name, type, properties, and metadata)

The client uses a handler to handle topic notifications. To receive notifications, the handler must select which parts of the topic tree to receive notifications on. You can use topic names or topic selectors or both to define these parts of the topic tree.

Considerations when using a topic notify topic

In some APIs, a client that subscribes to a topic notify topic must provide the handler that handles the notifications.

DEPRECATED: Child list topics

A functional topic that maintains a list of child topics and notifies subscribed clients when a child topic is added or removed directly.

A child topic is a topic that is directly beneath the child list topic in the topic tree. For example, if the child list topic is at the path `foo`, the topics `foo/bar` and `foo/fred` are its child topics. However, the topic `foo/bar/baz` is not a child topic of `foo`.

Why use a child list topic?

You can use child list topics to group sets of related data items. Each data item is a child topic beneath the child list topic. A client can subscribe to the child list topic and receive a notification when data item is added to a set (that is, when a child topic is added or removed).

Considerations when using a child list topic

Child list topics are supported only by the Classic API.

DEPRECATED: Paged string topics

A topic that maintains server-side state as a number of lines of string data. A client can view the data as pages made up of one or more lines and can page forward and backward through the data.

Note:

Paged topics are deprecated. They will be replaced with an equivalent capability in a future release.

It can be simpler to use a JSON topic with delta updates enabled. Although the whole value is published to a client when it first subscribes, subsequent updates are published incrementally.

Paged topics have state which is stored in a tabular format, as pages of lines. Unlike publishing topics, clients do not publish updates to a paged topic to change the state. Instead clients add, update, or remove lines in the topic.

Paged topics can be ordered or unordered. You can define the ordering of a paged topic by using a user-defined comparator class that is located on the Diffusion server or by declaring the rules that are used for the ordering when you create the topic. Lines that are added to unordered paged topics are added at the end. Lines that are added to ordered paged topics are added at the position defined by the comparator or rules.

A client must subscribe to a paged topic to be able to access the data on it. However, unlike publishing topics clients do not receive updates whenever changes are made to the paged topic state. To access the data, the client must open a view on the paged topic and specify how many lines per page and what page to open the view on. The client can then page through the data. If the state of the client's current page changes, the client is notified and can choose to refresh the page.

Why use a paged string topic?

Unlike most publishing topics, paged topics are not designed to distribute streaming data. Paged topics store tabular data. If your data is of a more tabular form — for example, news items — a paged topic might be most appropriate.

A paged string topic is suitable for simple items of data that can be represented as text.

The state of publishing topics is constantly changing, with no capability to look back at its previous values. With a paged topic you can store all updates on the topic and a client can view the history of previous values, without the client having to store them.

A paged topic can serve different parts of its state to different clients. Any subscribed client can view any page of the data that is required.

If the order of your data is important, a paged record topic ensures that updates are added in the appropriate position. This removes the requirement for a client to have to order the stored data before using or displaying it.

Considerations when using a paged string topic

Viewing paged topics is supported only by the Classic API.

You must be subscribed to a paged topic to open a view on it.

The lines of a paged topic are UTF-8 encoded.

You cannot use topic replication to replicate paged topics between Diffusion servers.

Paged topic data is not supported by the Introspector or the console.

DEPRECATED: Paged record topics

A topic that maintains server-side state as a number of lines of record data. The schema defines the record metadata that defines the lines. A client can view the data as pages made up of one or more lines and can page forward and backward through the data.

Note:

Paged topics are deprecated. They will be replaced with an equivalent capability in a future release.

It can be simpler to use a JSON topic with delta updates enabled. Although the whole value is published to a client when it first subscribes, subsequent updates are published incrementally.

Paged topics have state which is stored in a tabular format, as pages of lines. Unlike publishing topics, clients do not publish updates to a paged topic to change the state. Instead clients add, update, or remove lines in the topic.

Paged topics can be ordered or unordered. You can define the ordering of a paged topic by using a user-defined comparator class that is located on the Diffusion server or by declaring the rules that are used for the ordering when you create the topic. Lines that are added to unordered paged topics are added at the end. Lines that are added to ordered paged topics are added at the position defined by the comparator or rules.

A client must subscribe to a paged topic to be able to access the data on it. However, unlike publishing topics clients do not receive updates whenever changes are made to the paged topic state. To access the data, the client must open a view on the paged topic and specify how many lines per page and what page to open the view on. The client can then page through the data. If the state of the client's current page changes, the client is notified and can choose to refresh the page.

Why use a paged record topic?

Unlike most publishing topics, paged topics are not designed to distribute streaming data. Paged topics store tabular data. If your data is of a more tabular form — for example, news items — a paged topic might be most appropriate.

A paged record topic, enables you to use metadata to define the format of each line of data. If each line contains multiple items of data, paged record topics enables you structure this data, where a paged string topic does not.

The state of publishing topics is constantly changing, with no capability to look back at its previous values. With a paged topic you can store all updates on the topic and a client can view the history of previous values, without the client having to store them.

A paged topic can serve different parts of its state to different clients. Any subscribed client can view any page of the data that is required.

If the order of your data is important, a paged record topic ensures that updates are added in the appropriate position. This removes the requirement for a client to have to order the stored data before using or displaying it.

Considerations when using a paged record topic

Viewing paged topics is supported only by the Classic API.

You must be subscribed to a paged topic to open a view on it.

The lines of a paged topic are UTF-8 encoded.

You cannot use topic replication to replicate paged topics between Diffusion servers.

DEPRECATED: Service topics

A functional topic that implements a request/response type service. The service is implemented as a user-written server-side Diffusion class.

Note: Service topics are deprecated and will be removed in a future release. We recommend that you use the Messaging and MessagingControl features of the Unified API to implement request/response behavior.

Service topic data

Service topic data

Functional topic data that provides an Asynchronous Request/Response service

This is a special form of topic data that provides a request/response service framework.

It is a common requirement for clients to be able to send some form of command through a topic to a publisher, which performs some processing and optionally return some form of response on the same topic.

Service topic data provides the following functionality to support such a service paradigm:

- Adding service topic data to a topic makes that topic a service (or command based) topic where all inbound messages from the client are routed directly to the topic data instance for execution rather than to the publisher.
- A user written service handler is specified to perform the processing for a topic and this are invoked for every command message received on that topic.
- The handler can return a response immediately (synchronous processing) or return no response but delegate the return of the response to some later process (asynchronous processing).
- Subscription actions (sending of topic load) happens automatically, so no handling needed in `Publisher.subscription()` method.
- Automatic timeout mechanism for asynchronous requests.
- Built in error reporting mechanism.

Creating service topic data

Service topic data is created as follows:

```
ServiceTopicData topicData =  
    TopicDataFactory.newServiceData("Service1", new MyServiceHandler());
```

Configuring service topic data

Service topic data does not have state to be initialized as other topics do but there are some options to configure the behavior of the topic data before it is added to the topic:

Service data

Even though service topic data does not have data state there is the option to specify some static data for the topic which is sent to the client upon subscription (in the topic load message). The data is specified in the form of a Message which can contain headers and/or data.

The following example shows some service data being specified:

```
TopicMessage serviceData = createDeltaMessage("Service1");  
serviceData.putFields("A", "B", "C");  
topicData.setServiceData(serviceData);
```

Target topic Name

The `ServiceRequest` object passed to the `ServiceHandler` contains a message containing the data from the request passed from the client. By default, this message has its topic name set to that of the topic that owns the topic data. However, this can be changed if required, for example if the request is to be passed to another process that requires a different topic name. The `setTargetTopicName` method can be used to specify a different topic name.

When only synchronous processing is in use there is probably no point in changing the topic name in the message.

Header Options

The `ServiceRequest` object passed to the `ServiceHandler` contains a message containing the headers from the request passed from the client but might also have other headers automatically included. This is done using the `setHeaderOptions` method which allows a list of header types to be specified. Any headers specified in this list are included in the message within the request before any user headers passed from the client.

Such headers are required only for asynchronous processing as all of the specified information is available in the `ServiceRequest` object anyway when processing synchronously. The purpose of adding such headers is only for the sake of a process to which the request message can be sent.

The following header options are available:

SERVICE_TYPE

The value of the service type specified when the topic data was created.

REQUEST_TYPE

The request type is passed with the request from the client. This is needed to differentiate between different types of requests. However, if there is only ever one type of request, request type is not required.

CLIENT_ID

The client identifier of the client that sent the request.

REQUEST_ID

The request identifier sent from the client with the request which uniquely identifies the request for the client.

By default no additional headers are added.

Asynchronous Request Timeout

When asynchronously processed the topic data only waits for a certain amount of time before timing out the request and automatically sending a timeout error to the Client. By default this value is set to 5 seconds but can be changed using the `setRequestTimeout` method.

Writing a service handler

How to write the handler for a service topic

The key element of a service topic is the user written service handler which performs the actual processing. The service topic data itself is merely a framework within which to execute service handlers.

Service requests are sent from the client and routed through the topic data to an instance of the service handler as specified to the topic data.

A service handler must implement the `ServiceHandler` interface.

Requests are formatted into `ServiceRequest` objects and passed to the Service handler on its `serviceRequest` method. The `ServiceRequest` has information like the client details, a unique request identifier (from the client), a request type and request data in the form of a message. The request data comes from the message received from the client and can comprise user headers and/or data. It is also permissible to have no request data, just a request type.

The service handler can process the request and immediately return a `ServiceResponse` object encapsulating the details of the response to send back to the client. This is synchronous processing. The handler might also choose to delegate the processing to some other process which will asynchronously return the response at some later point using the `serviceResponse` method on the topic data.

For synchronous processing an error can be reported by throwing a `ServiceException` from the handler. This is formatted and returned to the client. For asynchronous processing a callback to the `serviceError` method on the topic data might be used to report a failure in processing.

Client handling of service topic data

How a client handles a service topic

At the client end the client application must be able to handle the service topic Protocol. Most client APIs provide the capability for handling such topics transparently. This section shows how it is handled in the Java client API.

Handling a topic load

A client receives messages on its listener methods and can detect a load message from a service topic by means of the `isServiceLoad()` method. On receipt of such a load message the client application must create a `ServiceTopicHandler` to handle the topic. This handler provides the facility to send requests to the topic and also routes responses and errors from the topic to a specified `ServiceTopicListener`. Such a handler can be created using the `ExternalClientConnection.createServiceTopicHandler` method.

The following code sample shows how to create a suitable topic handler on receipt of a service load message:

```
public void messageFromServer(ServerConnection serverConnection,
    TopicMessage message) {
    if (message.isServiceLoad()) {
        try {
            theHandler=theConnection.createServiceTopicHandler(message, this);
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

The returned handler is of type `ServiceTopicHandler` and the above example assumes the calling class implements `ServiceTopicListener` and process service responses and errors.

Having created such a handler no further messages are received for that topic on the `messageFromServer` method as they are all consumed by the handler.

Service type and service data

After creating the handler the service type and any service data sent from the server can be obtained from it. The service type identifies the service to the server, and can be used by the client to differentiate between different types of service. The server might also have returned service data

which can be used by the service to return any information that might be required by the client. The way in which service type and service data is used is entirely up to the service implementation at the server.

Sending Requests

A service request can be sent to the server through the handler using the request method. A request must specify a request type which must be a request type understood by the service. It can also optionally specify a message containing headers and/or data which is sent with the request. The message can be used to provide parameters to the service request.

The following example shows the simplest case where no parameters are required to the request:

```
String requestId = theHandler.request("GetAccounts", null);
```

The method returns a unique request identifier which can be used to correlate the response (or any error) returned from the service with the request.

The following example shows a message being used to pass a parameter-

```
TopicMessage message = theConnection.createDeltaMessage("XYZ", 50);  
message.put("12435");  
String requestId = theHandler.request("GetAccountDetails", message);
```

The topic name specified for the message is not important as it is replaced by the handler in the actual request that is sent.

Handling Responses

Responses to requests are returned using the `serviceResponse` method on the `ServiceTopicListener` specified when creating the service Handler. The `serviceResponse` method passes a `ServiceTopicResponse` object from which can be obtained the following:

Table 14: Handling Responses

<code>requestId</code>	This is the request identifier that was returned when the request was originally sent.
<code>responseType</code>	This is a response type as sent by the service and is used to allow the client to differentiate between different possible responses.
<code>responseMessage</code>	This is a message containing any headers and/or data returned by the service. This can be an empty message if the service did not return any data.

Handling Errors

If the service request fails in any way at the server, an error is returned through the `serviceError` method on the `ServiceTopicListener` specified when creating the service handler. The `serviceError` method passes a `ServiceTopicError` object from which can be obtained the following:

Table 15: Handling Errors

<code>requestId</code>	This is the request identifier that was returned when the request was originally sent.
------------------------	--

<code>errorType</code>	This is an enum with one of the following possible values:
<code>errorMessage</code>	This is the error message associated with the error. This will always be present.
<code>exceptionMessage</code>	This is an optional exception message which can be returned if the error was due to an exception. This can be null.
<code>additionalDetails</code>	This can return optional additional data associated with an exception. This can be null.

Table 16: Error types

<code>SERVICE</code>	An error has occurred whilst executing the service
<code>INVALID</code>	The service request was invalid
<code>TIMEOUT</code>	The request was executed asynchronously at the server but was timed out before a response was returned.
<code>USER</code>	An error was returned by the user written service handler.
<code>DUPLICATE</code>	A duplicate request identifier was sent. This cannot happen when using the Java API interface but is present for completeness.

Unsubscribing

When the client application unsubscribes from the service topic then the handler will become unusable and any outstanding requests for which responses have not been returned are discarded.

DEPRECATED: Protocol buffer topics

A topic that maintains state at the server in Google protocol buffers format.

Note: Protocol buffer topics are deprecated and will be removed in a future release. We recommend that you stream your protocol buffer data using a binary topic instead and handle serialization and deserialization of the binary data at your clients.

Protocol buffers provide an extensible mechanism for serializing structured data.

For more information about protocol buffers, see <https://developers.google.com/protocol-buffers/>.

Define the format of data on a protocol buffer topic by using a `.proto` file. Compile the `.proto` file into a Java class and ensure that the class is on the Diffusion server classpath.

The schema defines a compiled `.proto` class which must exist at the server and the name of a message definition within the class that defines the topic data layout.

Why use a protocol buffer topic?

Data on protocol buffers topics is simple and small, and can be transmitted and parsed quickly.

A protocol buffer topic benefits from the same delta-processing capability at the server as a record topic.

Serialization and deserialization code can be automatically generated for the data on the topic.

Considerations when using a protocol buffer topic

The compiled `.proto` Java class must be present on the classpath of the Diffusion server. If your solution uses multiple Diffusion servers that all host a protocol buffer topic with the same message definition, the same compiled `.proto` class must be present on all of their classpaths.

Protocol buffer definitions must be compiled into Java.

The supported version of protoc is 2.6.1.

Publication

Having decided on your topic structure and the format of your data, consider how you publish the data through the topics.

Pub-sub is the primary model of data distribution used by Diffusion. Clients subscribe to a topic. When data is published to the topic as an update, the Diffusion server pushes that update out to all of the subscribed clients.

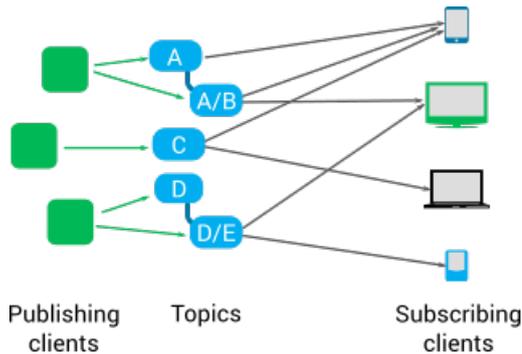


Figure 2: Pub-sub model

A client can both publish to topics and subscribe to topics, depending on the permissions that client has.

Concepts

Update

An update is data published to a topic by a client or publisher that is applied to the topic to change the topic state. The updated data is then pushed out to all subscribing clients.

State

The latest published values of all data items on the topic. The state of a topic is stored on the Diffusion server.

Value

A value is an update that contains the current state of all data on the topic.

Delta

A delta is an update that contains only those items of data that have changed on the topic since the last update was sent.

Topic loading

When a client first subscribes to a topic, it is sent a topic load message. A topic load is a value update that contains the current state of the topic.

Fetch

A request for the current state of all data on the topic. A client can fetch a topic's state without being subscribed to the topic. This request-response mechanism of getting data from a topic is separate from the pub-sub mechanism.

Publishing data

Consider the following information when deciding how to publish data to topics.

Data type

The updates that you publish to a topic must have a data type and format that matches the data type of the topic.

For example, if your topic is a single value topic where the data is of type integer, all updates published to the topic must contain a single piece of integer data.

Similarly, if your topic is a record topic with a metadata structure defined, all updates published to the topic must have the same metadata structure.

Updaters

You can use one of the following types of updater:

Value updater

This is the preferred type of updater to use with JSON and binary topics. When used as exclusive updaters, value updaters cache the values they use to update topics. This enables them to calculate and send deltas, reducing the volume of data sent to the Diffusion server.

Standard updater

This type of updater updates topics that use content to represent their data values. Updaters do not cache values and send all of the data passed to them to the Diffusion server without performing any optimization.

Both updater types can be used exclusively or non-exclusively.

For more information, see [Updaters](#).

Exclusive updating

To update a topic exclusively, a client registers as the update source for that topic. Only one client can be the active update source for a topic and any attempts by other clients to update that topic fail.

Implementing exclusive updating is more complex than non-exclusive updating as it involves the extra step of registering as an update source.

A single client acting as the exclusive updater can be an advantage if you require that a single client has ownership of a topic or branch of the topic tree. This requires less coordination and management than updating a single topic from multiple clients.

If the ordering of the updates is important, use exclusive updating to ensure that a single client has control over what data is published and when.

If you are using high-availability topic replication, clients must update the replicated topics exclusively. Non-exclusive updates are not replicated by high-availability topic replication.

Non-exclusive updating

To update a topic non-exclusively, a client publishes updates to the topic and, if no other client has registered to update the topic exclusively, the update is applied to the topic.

Non-exclusive updating is the simpler way to update a topic.

Clients that update a topic non-exclusively risk their updates being overwritten by updates from other clients or that updates from multiple clients are published in a different order than intended.

If you use a value updater non-exclusively, the updater does not cache the value used to update the topic.

Non-exclusive updating is not supported with topics that are replicated using the high-availability capability.

Dynamically adding topics

A publishing client can create topics dynamically as and when the topics are required. For example, in response to a subscription request from another client for a non-existent topic.

Security

To publish data to a topic, a client must have the `update_topic` permission for that topic.

For more information, see [Permissions](#) on page 134.

Subscribing to topics

Consider the following information when deciding how clients subscribe to topics.

Registering streams

When you subscribe to topics the updates to that topics are received through streams. Register a stream against a set of topics in the topic tree that contains the topic or topics you subscribe to. This ensures that the client receives updates for a subscribed topic.

For more information, see [Streams](#).

Subscribing to multiple topics

A client can subscribe to multiple topics in a single subscribe request. This subscription can be to topics that match a particular regular expression or to topics in a particular branch of the topic tree.

Use topic selectors to define a set of topics.

For more information, see [Topic selectors in the Unified API](#) on page 61 and [Topic selectors in the Classic API \(deprecated\)](#) on page 69

Subscribing to topics that do not exist

A client can subscribe to a topic that does not exist. This is pre-emptive subscription. Diffusion keeps track of all subscribe and unsubscribe requests from a client, including for topics that do not exist. When a new topic is created, if a client has subscribed to that topic path, the client is subscribed to the new topic and receives updates from it.

Forced subscription

A publisher or a client with the required permissions can force other clients to become subscribed to a topic or set of topics, even if the subscribing client did not request the subscription.

Security

To subscribe to a topic, a client must have the `read_topic` permission for that topic.

For more information, see [Permissions](#) on page 134.

Messaging

In addition to publishing data to topics as updates, you can send data as messages to specific clients or to registered message handlers on topic paths.

Messaging and pub-sub are separate capabilities. Messaging uses topic paths to route the messages, but does not use the topics bound to those paths. Sending messages does not change the state of data on the topic, nor does it cause updates to be sent.

There are two ways in which messages are sent through topic paths:

Send messages to a topic path

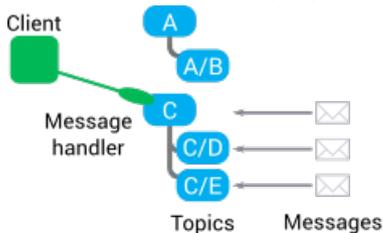


Figure 3: A client registers a handler on part of the topic tree

A client with the `send_to_message_handler` permission can send messages on a topic path. The sending client does not know which client or publisher, if any, receives the message.

A client with the `register_handler` permission can register a handler on a part of the topic tree. This client receives any messages that are sent on topic paths in that part of the topic tree.

Send messages to a client session

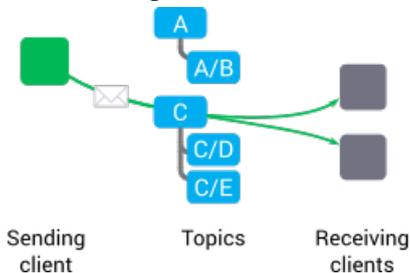


Figure 4: A client can send messages through a topic path to known client sessions

A client with the `send_to_session` and `view_session` permissions that knows the session ID of a client session can send messages through a topic path to the known client session.

The receiving client must have a message stream registered against a topic path to receive messages sent through that topic path. For more information, see [Streams](#).

Considerations when using messaging

- The data type of the value sent in the message is not required to match the data type of the topic, if any, bound to the topic path you use to send the message.
- A message can include headers. These headers are a set of string values that are separate to the message content.
- You can set a priority on messages sent to client sessions. This priority affects the order in which the messages are queued on the Diffusion server for delivery to the client.
- The following APIs require that a topic exists at a topic path to send a message through it and that they are subscribed to a topic path to receive a message through it:

- Android Classic API (deprecated)
- Flash Classic API (deprecated)
- iOS Classic API (deprecated)
- Silverlight Classic API (deprecated)
- C Classic API (deprecated)
- Java Classic API (deprecated)
- JavaScript Classic API (deprecated)
- .NET Classic API (deprecated)

Advanced usage

This section contains information about the advanced capabilities available when interacting with your data.

Conflation

Conflation of messages is the facility to treat two messages as being essentially the same and avoiding sending duplicate information to clients.

Updates and messages for a client are queued on the Diffusion server. Each client has its own outbound queue.

Conflation removes an existing message from the outbound client queue and replaces it with a newer, equivalent message either at the position of the old message or at the end of the client queue. The replacement message can either be a new update or a merge of the new update and an existing update.

Conflation is an optional feature that can be applied to all clients, clients connecting through a specific connector, or can be applied programmatically on a client-by-client basis.

Advantages of message conflation

There are many scenarios in realtime data distribution where data being communicated need not only be current, but must always be consistent. Structural conflation maximizes for concurrency while ensuring consistent delivery.

Concurrency

Both simple and structural conflation maximize for concurrency through avoiding distributing soon to be stale data. The higher the rate of change, the higher the value extracted. On other words, where clients or servers are running near to saturation based on available connectivity Diffusion can automatically adapt to this load by minimizing the data distributed.

In addition to the load-adaptive nature of conflation the effect is fair for clients connected to the same topic. The frequency of distributed changes is evenly amortized across clients.

Conflation favors currency and reduces (but cannot entirely eliminate) the delivery of stale data.

Consistency

Structural conflation synthesizes complex event processing techniques in a highly efficient (lock-free wait-free concurrency) form inside the server. The specific knowledge of data structures and the semantic concerns for distributing data for a given topic in a given system allows consistent views of the data to be delivered in a way that is not possible with messaging technologies that treat messages as opaque.

Considerations when using conflation

- Do not use conflation if there are relationships or dependencies between topics. Conflation alters the order of updates. If a conflated topic is temporally or causally related to another topic, conflation can cause unwanted behavior.
- Do not use conflation if individual updates carry forensic storage or audit trail requirements.
- Delta updates are conflated. Snapshots are not conflated.
- Normal priority updates are conflated. High or low priority updates are not conflated.
- Messages that require acknowledgment are not conflated.

Types of message conflation

The types of conflation are *simple conflation*, where a new update replaces an older update, and *structural conflation*, where the data from two updates is combined in accordance with a user-defined operation to create a new update. Both types of conflation can either append the new update to the end of the queue or replace an existing update on the queue.

No conflation



Figure 5: Message flow without conflation enabled

With no conflation, a stream of messages to a client is delivered to the client in the order that they are published or sent. In example shown in the diagram, two messages for topic A, one message for topic B, and two messages for topic C are ready to send to the client.

This is a scenario common to all messaging platforms.

Simple conflation

When a new message is published or sent to a topic, simple conflation removes any earlier messages for that topic from the queue. The new message can be added to the queue either in the position of the earlier message that was removed (replace) or at the end, preserving the original message order (append).

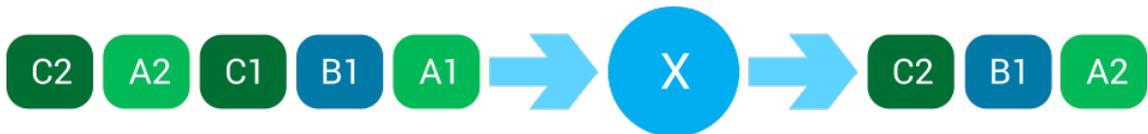


Figure 6: Message flow with simple replace conflation enabled

With simple replace conflation, only the most recent message for a topic is delivered to the client. In the example shown in the diagram, the first message published or sent to topic A is removed and the second message is added to the queue in the position that the first message occupied. The same occurs for the messages sent to topic C.

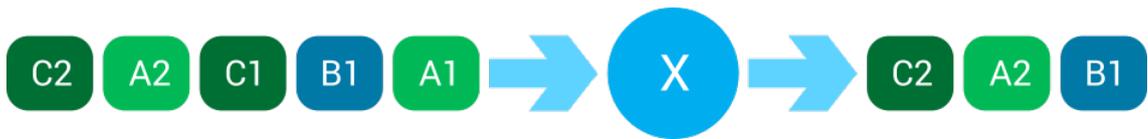


Figure 7: Message flow with simple append conflation enabled

With simple append conflation, only the most recent message for a topic is delivered to the client. Messages are delivered in time order. In the example shown in the diagram, the first message published or sent to topic A is removed and the second message is added to the end of the queue. The same occurs for the messages sent to topic C.

Note: Use this option with care because there is the possibility that messages for a topic can continue to be added to the end of the queue and a value not be delivered for the topic.

In both the append and the replace example, although five messages were ready to send, only three messages were sent. This saves bandwidth and ensures clients receive current data only.

Structural conflation

Structural conflation allows a user-defined operation to be plugged into Diffusion so that rather than refreshing stale data with fresh data, a computation can be performed to merge, aggregate, reverse or combine the effects of multiple changes into a single consistent and current notification to the client.

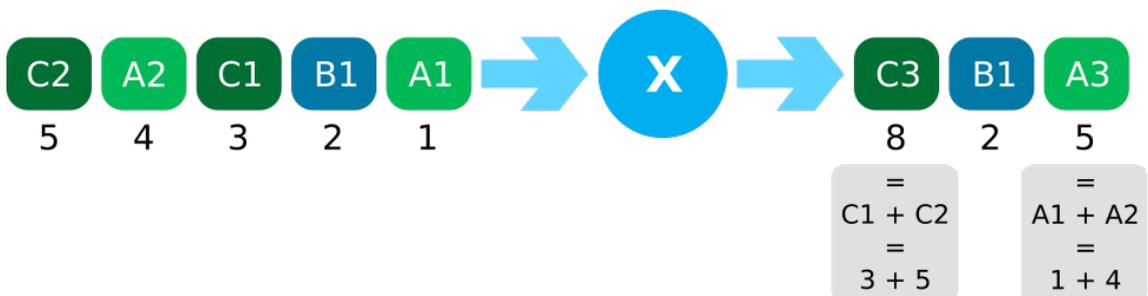


Figure 8: Message flow with merge and replace conflation enabled

In the example shown in the diagram, the operation is the summation of numeric data. The user provides the merge algorithm, that is the summing of the values of two successive messages, and Diffusion sends a single resulting message rather than the individual messages that were combined. The messages A3 and C3 are new messages generated from the merging process.

This is suitable for any scenario where the result is required but individual components that combine to form the result are not required.

The preceding example shows merge and replace. You can merge and append in a similar way as described for simple conflation above.

Various options are available to the user-written merger so that instead of returning a merged message it can indicate that either of the input messages be queued or that the no conflation option be chosen.

Selection of messages for conflation

The preceding examples assume that when a new message is queued for a client it replaces or merges with the last message queued for the message topic. This is the default behavior. When operating conflation in this way there is only ever one message per conflated topic awaiting delivery to the client.

You can specify a user-defined matcher that is used to determine the message that is to be replaced or merged with. This can be used to inspect the content of the messages queued for a topic to select which to conflate. When operating conflation in this way it is likely there can be more than one message per conflated topic awaiting delivery to the client.

How conflation works

Conflation is implemented as a lock-free, wait-free algorithm and can be scaled to meet demands of large numbers of concurrently subscribed clients.

Conflation policies

Conflation policies configure how conflation takes place on topics. One or more conflation policies can be configured, each defining different conflation mechanisms. The policies can define the following behaviors:

- How messages are matched
- Whether replacement is done in place or by appending
- How to merge the two messages

Conflation process

When conflation is enabled for a client, every time a new update is enqueued for the client it is considered for conflation.

1. The Diffusion server checks whether a conflation policy is mapped to the topic that the update is published on.
2. If a policy is mapped to the topic, the matching criteria of the policy is used to scan the updates for that topic that exist on the queue (from newest to oldest) for an existing update that matches the one being queued. If no match is found, the new update is queued at the end of the queue as normal.

Note:

Fast caches and lookup mechanisms are used to find policies and updates in the queue. The whole client queue is not scanned when looking for a match, only updates for the same topic.

If default matching (by topic) is used, no comparison with the existing updates is required. This means that the conflation mechanism is as efficient as possible.

3. If the matching criteria finds a matching update in the queue, the conflation policy defines the following behaviors:
 - Whether the update to queue as a result of conflation is the new update or an update produced by merging the content of the matching update and new update.
 - Whether the update to queue replaces the matching update or the matching update is removed and the new update added to the end of the queue.

Conflation occurs on a client-by-client basis in the multiplexer thread. Results of merge actions are cached to ensure that merges of identical message pairs are not repeated for different clients.

DEPRECATED: Distributing and viewing data as pages

Paged topics act differently to standard publishing topics. Distributing data through paged topics

Concepts

Page

A page is a subset of the tabular data stored on a paged topic. Each page is made up of a certain number of lines.

Line

A line is a single item of data stored on a paged topic. A line can be a string or a record.

Add

Lines are added to a paged topic.

Update

The content of lines can be updated.

Remove

Lines can be removed from a paged topic.

View

A view is a representation of data on a paged topic that can be opened by a subscribing client. The subscribing client can open a view on the paged topic that includes a specified number of lines of the topic data.

Ordering

Lines on a paged topic can be ordered according to simple or user-defined criteria.

Dirty

If a line is added into a paged topic in a position that is open in a subscribing client's view. That view is dirty and must be refreshed to show the latest topic data.

Behavior

Paged topics have state which is stored in a tabular format, as pages of lines. Unlike publishing topics, clients do not publish updates to a paged topic to change the state. Instead clients add, update, or remove lines in the topic.

Paged topics can be ordered or unordered. You can define the ordering of a paged topic by using a user-defined comparator class that is located on the Diffusion server or by declaring the rules that are used for the ordering when you create the topic. Lines that are added to unordered paged topics are added at the end. Lines that are added to ordered paged topics are added at the position defined by the comparator or rules.

A client must subscribe to a paged topic to be able to access the data on it. However, unlike publishing topics clients do not receive updates whenever changes are made to the paged topic state. To access the data, the client must open a view on the paged topic and specify how many lines per page and what page to open the view on. The client can then page through the data. If the state of the client's current page changes, the client is notified and can choose to refresh the page.

Designing your solution

Decide how your solution components interact to most efficiently and securely distribute your data.

There are a number of things to consider when designing your Diffusion solution:

- The number, distribution, and configuration of your Diffusion servers
- How you use clients in your solution
- The additional components to develop
- The third party components you might include in your solution
- Securing your solution

These considerations are not separate. The decisions you make about one aspect of your solution can affect other aspects.

Servers

Consider the quantity, distribution, location and configuration of your Diffusion servers.

How many Diffusion servers?

Consider the following factors when deciding how many Diffusion servers to use in your solution:

Number of client connections

How many client connections do you expect to occur concurrently? For a greater number of concurrent client connections, you might require more Diffusion servers to spread the load between.

Volume of data

At what rate are you publishing updates and sending messages? How large are the updates and messages? If you are distributing a greater volume of data, you might require more Diffusion servers to spread the load between.

Hardware capabilities

The number of concurrent client connections and the volume of data that a single Diffusion server can handle depend on the hardware that the Diffusion server runs on.

In order of importance, the following hardware components have the biggest impact on the server performance:

- Network interface controller (NIC)
- Central processing unit (CPU)
- Random access memory (RAM)

Resilience and failover requirements

Ensure that you have enough Diffusion servers that if one or more becomes unavailable, for example when updating the server or due to a failure of the hosting system, the remaining Diffusion servers can spread the resulting load increase.

You can also use replication between Diffusion servers to increase your solution's resilience. For more information, see [High availability](#) on page 104.

Distribution of servers

How you wish to distribute your servers has an effect on how many servers you require.

For example, if your client base is distributed geographically, you might want to locate your Diffusion servers in different territories. This enables your servers to be more responsive because of their proximity to clients. In this case, the number of territories your client base is spread over affects the number of servers you require.

You can easily scale your solution by adding additional Diffusion servers if your requirements change.

How are your Diffusion servers configured?

Consider the following factors when deciding how to configure the Diffusion servers in your solution:

Ports

What ports do you want to provide access to your Diffusion server on? You can configure the Diffusion server to allow the following kinds of connections on individual ports:

- Unified API clients only
- Classic API clients only

- Both Unified API and Classic API clients
- Policy file requests only
- JMX clients

By default, your Diffusion server supports client connections on port 8080.

Reconnection behavior

Do you want to allow clients that lose their connection to reconnect to the server? How long do you want to keep client sessions available after the client loses connection?

Not all client types can take advantage of reconnection. For more information, see the section of the manual for each specific API.

Replication

Replication enables Diffusion servers to share information about topics and client sessions with each other through a data grid.

For more information, see [High availability](#) on page 104.

Performance

Tuning your Diffusion servers for performance is best done as part of testing your solution before going to production. This enables you to observe the behavior of your solution in action and configure its performance accordingly.

For more information, see [Tuning](#) on page 1052.

For more information, see [Configuring your Diffusion server](#) on page 550.

This manual describes the factors that you must consider when designing your Diffusion solution. However, these factors are too many and too interlinked for this manual to provide specific guidance.

Push Technology provides Consulting Services that can work with you to advise on a solution that best fits your requirements. Email [Sales at Push Technology](#) for more information.

Fan-out

Consider whether to use fan-out to replicate topic information from primary servers on one or more secondary servers.

A fan-out distribution comprises many servers that host the same topic or topics. When the topic is updated on a primary server, the update is fanned out to replica topics on secondary servers.

Why use fan-out?

Having a primary server feed out updates to a number of secondary servers provides a solution where the same topics and data are available from multiple servers. You can use this solution to load balance a large number of client connections across a set of Diffusion servers and provide those clients with the same access to data.

How fan-out works

Fan-out is configured on the secondary server or secondary servers in the solution.

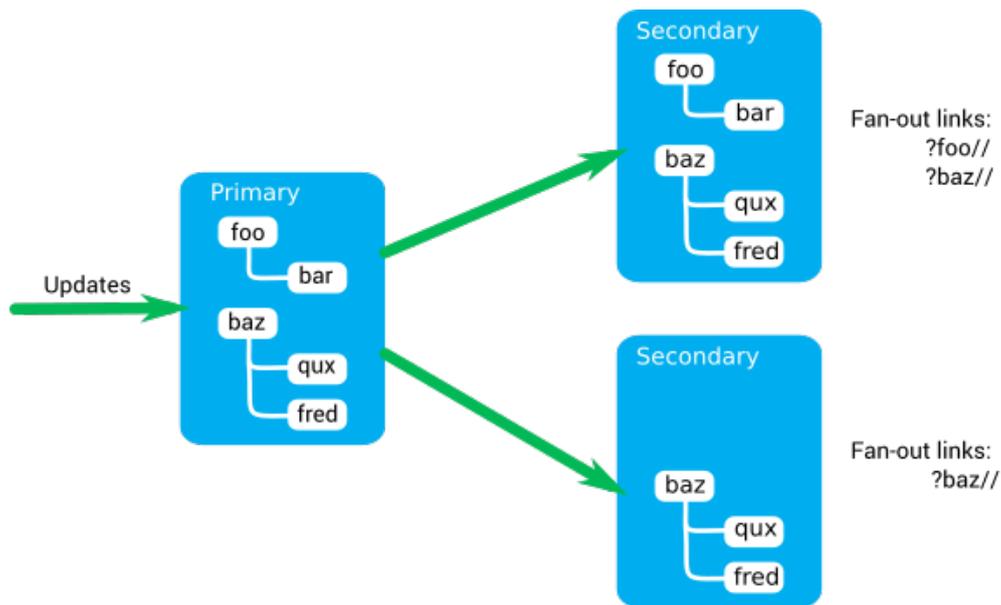


Figure 9: Fan-out

- A secondary server connects to a primary server as a client.
- The secondary server subscribes to a set of topics on the primary server.
 - This set of topics is defined by a selector in the configuration of the secondary server.
- The secondary server replicates the subscribed topics locally.
- When updates are made to the topics on the primary server, the secondary server receives these updates through the standard pub-sub feature in the same way as any other client of the primary server.
- The secondary server applies the updates to its replica topics.
- Any clients subscribed to a replica topic on the secondary server receive the updates through the standard pub-sub feature.
- If a topic is removed at the primary server, the secondary server removes its replica topic.
- If a topic is added at the primary server that matches the set of topics subscribed by the secondary server, the secondary server creates a local replica topic.

A secondary server can connect as a client and subscribe to topics on more than one primary server. However, ensure that the secondary server does not attempt to replicate the same topic from multiple sources as this can cause the data on the topic to be incorrect.

Creating topics on the primary server is an asynchronous action, because of this a client or publisher that creates a topic on the primary server receives a completed callback saying that the topic has been created. However, receiving this callback does not indicate that the topic has been replicated by fan-out and created on a secondary server.

Topic types supported by fan-out

Fan-out supports only the following types of topic:

- JSON
- Binary
- Single value topics
- Record topics
- Stateless topics
- Slave topics

The order of topic creation on the secondary server can prevent slave topics from being replicated. For example, if a slave topic refers to a topic that does not yet exist because it is in a branch not yet replicated or because it is lower down the link hierarchy.

- Routing topics

To use fan-out with routing topics, the routing subscription handlers for a routing topic must be registered at the primary and all secondary servers. The routing logic provided by the handlers on the primary and secondary server must be identical.

- DEPRECATED: Paged string topics
- DEPRECATED: Paged record topics
- DEPRECATED: Custom topics

The custom class for that topic must be available on the classpath of all Diffusion servers replicating that topic.

- DEPRECATED: Protocol buffer topics

The compiled `.proto` file that defines the format of the data on a protocol buffer topic must be available on the classpath of all Diffusion servers replicating that topic.

Topics that match the selector, but are not of one of these topic types, are not replicated by the secondary server.

Topic replication and fan-out

A secondary server cannot replicate the same topic from more than one primary server or multiple times from the same primary server. Validation of the root path part of the selectors is in place to prevent this occurring, but the use of regular expressions in topic selectors can result in an overlap of replication which can cause problems.

Missing topic notifications generated by subscription or fetch requests to a secondary server are propagated to missing topic handlers registered against the primary servers. For more information, see [Using missing topic notifications with fan-out](#) on page 102.

Fan-out and load balancers

If you add a secondary server to a load balancer pool before all topics have propagated from the primary server, it can result in a large number of messages being generated, leading to `MESSAGE_QUEUE_LIMIT_REACHED` errors appearing in the logs.

If you experience this problem, introduce a delay between enabling fan-out and adding any of the secondary servers to a load balancer pool. There is currently no built-in way to determine when propagation is complete, so you will need to experiment to find out how long the delay needs to be for your configuration.

Reconnection and disconnection

You can configure fan-out servers to use the standard reconnect mechanism. If the connection between the secondary server and the primary server is lost, the secondary server can reconnect to the same session. However, if messages are lost between the primary and secondary server, the reconnection is aborted and the session closed. The secondary server must connect again to the primary server with a new session.

If a disconnection between the primary and secondary server results in the session being closed, the secondary server removes all the topics that it has replicated from that primary server. (Only topics explicitly defined by a selector are removed.) Clients subscribing to these topics on the secondary server become unsubscribed. If the secondary server connects again to that primary server with a new session, the secondary server recreates the topics. Clients connecting to the secondary server become resubscribed to the topics.

Related concepts

[Configuring fan-out](#) on page 555

Configure the the Diffusion server to act as a client to one or more other Diffusion servers and replicate topics from those servers.

Using missing topic notifications with fan-out

Missing topic notifications generated by subscription or fetch requests to a secondary server are propagated to missing topic handlers registered against the primary servers.

Control client sessions can use missing topic notifications to monitor the activity of end-user client sessions. In response to subscription or fetch requests to missing topics, the control client session can choose to take an action, such as creating the missing topic.

For more information, see [Handling subscriptions to missing topics](#) on page 317.

How notification propagation works

A missing topic notification is propagated from a secondary server to a missing topic handler registered against a primary server if and only if all of the following conditions are met:

- There is a session between the secondary server and the primary server.
- The selector used for the subscription or fetch request to the secondary server intersects with one or more of the fan-out links to the primary server that are configured at the secondary server.
- On the secondary server, there are no currently replicated topics that match both the fan-out link and selector used in the subscription or fetch request.
- The primary server has no topics that match the selector used in the subscription or fetch request.
- One or more missing topic handlers are registered against the primary server for a path that matches the selector. The following rules are used to select which missing topic handler receives the notification:
 - If multiple handlers are registered for the branch, the handler for the most specific topic path is notified.
 - If there is more than one handler for a path, the Diffusion server notifies a single handler.

The handler can use the supplied callback to respond proceed or cancel. The subscription or fetch operation is delayed until the handler responds, and is abandoned if the response is cancel.

Example flow

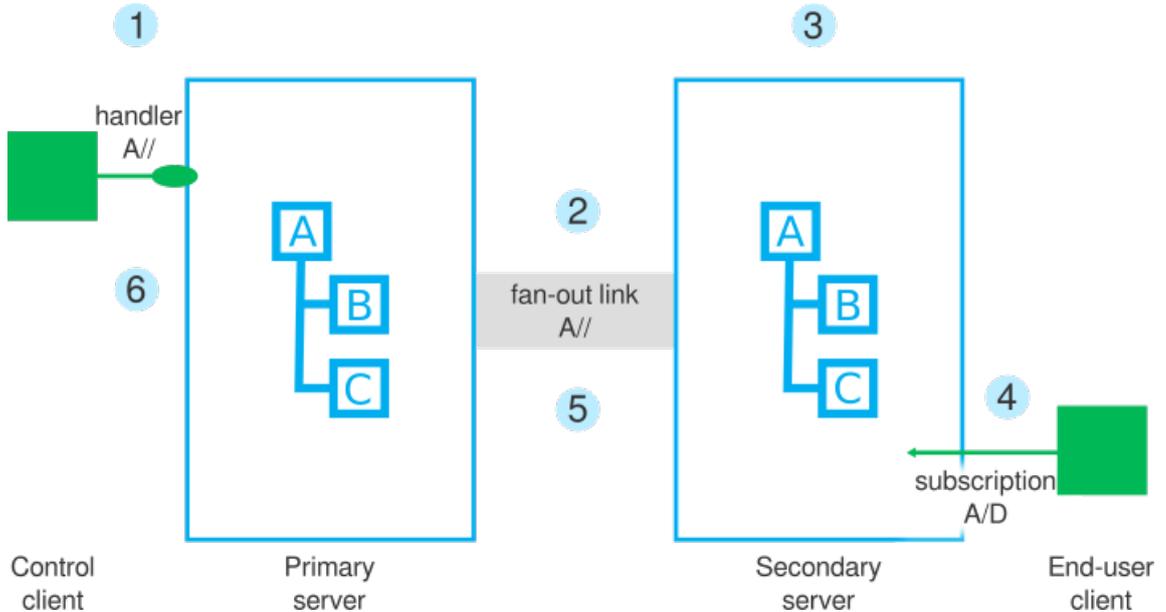


Figure 10: Missing topic notification propagation

1. A control client connects to the primary server and registers a missing topic notification handler against the A branch of the topic tree.
2. A secondary server connects to the primary server and replicates the A branch of the topic tree.
3. On the secondary server the replicated part of the topic tree comprises the following topics: A, A/B and A/C.
4. An end-user client attempts to subscribe to A/D, which does not exist.
5. The topic A/D is in part of the topic tree that is matched by a fan-out link selector, so the secondary server propagates the missing topic notification to the primary server.
6. The topic A/D does not exist on the primary server, so the primary server sends the missing topic notification to the handler registered by the control client.

Missing topic notification handlers at both the primary and secondary servers

A single subscription or fetch can cause a missing topic notification to be sent to a handler registered against the secondary server as well as a handler registered against a primary server.

The decision about whether to notify the handlers registered against a primary server is based on the intersection of the selector used by the subscription or fetch with the selector used to configure the fan-out link. It is possible for a missing topic notification to be sent to the primary server, but not to local handlers because the selector matches other (non-replicated) topics hosted by the secondary.

In particularly complex configurations, multiple primary servers might receive the notification or there can be multiple tiers of fan-out connections.

Where multiple handlers are notified, the subscription or fetch operation is delayed until the all handlers respond, and the operation is abandoned if any response is cancel.

Considerations when using missing topic notifications with fan-out

Missing topic notifications are only propagated if both the primary and secondary server are Diffusion version 5.9.1 or later.

The intersection of the selector used by the subscription or fetch request with a selector used for a fan-out link is calculated based only on the path-prefix of each selector. Complex selectors that use regular expressions can produce false positive results or false negative results. We recommend that you do not use regular expressions in the selectors used to configure fan-out links.

Ensure that the principal that the secondary server uses to make the fan-out connection to the primary server has the `SELECT_TOPIC` permission for the path prefix of the selector that triggered the missing topic notification.

A current session must exist between the secondary server and the primary server to forward notifications. If there is no session or the session fails while the missing topic notification is in-flight, the secondary server logs a warning message and discards the notification. The subscription or fetch operation is completed as if the primary handler had responded proceed.

The robustness of the session between the servers can be improved by configuring reconnection. Fan-out connections can have a large number of messages in flight. It might be necessary to tune the reconnection time-out and increase queue depth and recovery buffer sizes.

Related concepts

[Handling subscriptions to missing topics](#) on page 317

A client can use the `TopicControl` feature of the Unified API to handle subscription or fetch requests for topics that do not exist.

High availability

Consider how to replicate session and topic information between Diffusion servers to increase availability and reliability.

Diffusion uses a datagrid to share session and topic information between Diffusion servers and provide high availability for clients connecting to load-balanced servers.



Figure 11: Information sharing using a datagrid

Diffusion uses Hazelcast™ as its datagrid. Hazelcast is a third-party product that is included in the Diffusion server installation and runs within the Diffusion server process.

The datagrid is responsible for the formation of clusters and the exchange of replicated data. These clusters operate on a peer-to-peer basis and by default there is no hierarchy of servers within the cluster.

Servers reflect session and topic information into the datagrid. If a server becomes unavailable, another server can access the session and topic information that is stored in the datagrid and take over the responsibilities of the first server.

Considerations

Consider the following factors when using replication with Hazelcast:

- By default Hazelcast uses multicast to discover other nodes to replicate data to. This is not secure for production use. In production, configure your Hazelcast nodes to replicate data only with explicitly defined nodes. For more information, see [Configuring your datagrid provider](#) on page 612.
- When Diffusion servers are merged into a cluster, the servers can have inconsistent replicated data. Unresolved inconsistencies can cause unpredictable behavior. If the inconsistencies cannot be resolved, the inconsistent Diffusion server or servers are shutdown and must be restarted.

Diffusion servers in a cluster can become inconsistent in a number of circumstances, for example, if a network partitions and then heals.

Session replication

You can use session replication to ensure that if a client connection fails over from one server to another the state of the client session is maintained.

When a connection from a client through the load balancer to a Diffusion server fails, the load balancer routes the client connection to another Diffusion server. This server has access to the session and client information that is replicated in the datagrid.

Clients that connect to a specific Diffusion server and not through a load balancer cannot use session replication.

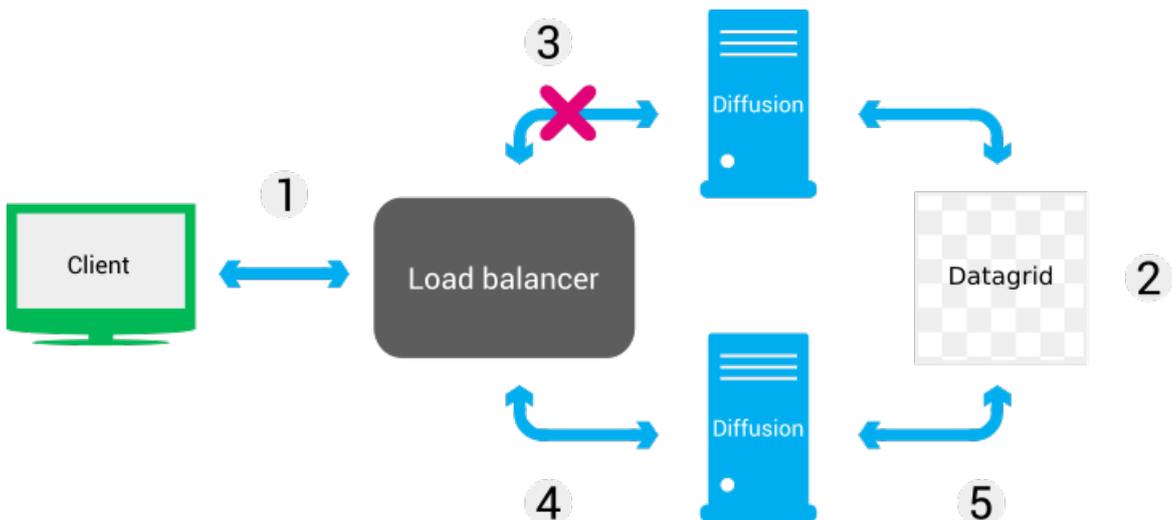


Figure 12: Session replication

1. A client connects to a Diffusion server through a load balancer.

The load balancer is configured to route based on the client's session ID and requests from the client go to the same server until that server becomes unavailable.

2. Information about the client session is reflected into the datagrid.

The following information is replicated:

- session ID
- session principal
- session properties
- list of topic selections

The following information is not replicated and is created anew on the server a client fails over to:

- session start time
 - statistics
 - client queue
3. A client loses connection to the Diffusion server if the server becomes unavailable.
 4. The client can reconnect and the load balancer routes the connection to another Diffusion server.
 5. This Diffusion server has access to all of the client information shared into the datagrid by the first Diffusion server.
 6. The server uses the list of topic selections to recover the set of subscribed topics and subscribes the client to these topics.
 7. Subscribing the client to topics provides full value messages for all topics that contain the current topic state.

The client can reconnect to its session only if it reconnects within the reconnect time specified in the `Connectors.xml` configuration file. If the client does not reconnect within that time, the client session information is removed from the datagrid.

Considerations

Consider the following factors when using session replication:

- Replication of session information into the datagrid is not automatic. It must be configured at the server.
- Messages in transit are not preserved. Use acks to ascertain whether or not messages have been received.
- When a Classic API client session reconnects it must be authenticated again. Ensure that all Diffusion servers in your solution have access to the same authentication methods.
- When a Unified API client session reconnects it does not need to authenticate again. The client uses a session token to reacquire its session. Ensure that this token is secure by using a secure transport to connect, for example, WSS.
- The failover appears to the client as a disconnection and subsequent reconnection. To take advantage of high server availability, clients must implement a reconnect process.
- The Diffusion server that a client reconnection attempt is forwarded to depends on your load balancer configuration. Sticky load balancing can be turned on to take advantage of reconnection or turned off to rely on session replication and failover.

Differences between session reconnection and session failover

When a client loses a load-balanced connection to Diffusion, one of the following things can occur when the client attempts to reconnect through the load balancer:

Session reconnection

The load balancer forwards the client connection to the Diffusion server it was previously connected to, if that server is still available. For more information, see [Reconnect to the Diffusion server](#) on page 252.

Session failover

The load balancer forwards the client connection to a different Diffusion server that shares information about the client's session, if session replication is enabled between the servers.

Prefer session reconnection to session failover wherever possible by ensuring that the load balancer is configured to route all connections from a specific client to the same server if that server is available.

Session reconnection is more efficient as less data must be sent to the client and has less risk of data loss, as sent messages can be recovered, in-flight requests are not lost, and handlers do not need to be registered again.

For more information, see [Routing strategies at your load balancer](#) on page 643.

To a client the process of disconnection and subsequent reconnection has the following differences for session reconnection or session failover.

Session reconnection	Session failover
The client connects to the same Diffusion server it was previously connected to.	The client connects to a Diffusion server different to the one it was previously connected to.
The client sends its last session token to the server.	
The server authenticates the client connection or validates its session token.	
<p>The server uses the session token to resynchronize the streams of messages between the server and client by resending any messages that were lost in transmission from a buffer of sent messages.</p> <p>If lost messages cannot be recovered because they are no longer present in a buffer, the server aborts the reconnection.</p>	The server uses the session token to retrieve the session state and topic selections from the datagrid.
The server sends any messages that have been queued since the session disconnected.	<p>The server uses the state to recover the session, uses the topic selections to match the subscribed topics, and sends the session the current topic value for each subscribed topic.</p> <p>Any in-flight requests made by the client session to the previous server are cancelled and the client session is notified by a callback. All handlers, including authentication handlers and update sources, that the client session had registered with the previous server are closed and receive a callback to notify them of the closure.</p>

Related concepts

[Client reconnection](#) on page 1062

You can configure the client reconnection feature by configuring the connectors at the Diffusion server to keep the client session in a disconnected state for a period before closing the session.

Related tasks

[Configuring the Diffusion server to use replication](#) on page 611

You can configure replication by editing the `etc/Replication.xml` files of your Diffusion servers.

Related reference

[Topic replication](#) on page 108

You can use topic replication to ensure that the structure of the topic tree, topic definitions, and topic data are synchronized between servers.

[Failover of active update sources](#) on page 110

You can use failover of active update sources to ensure that when a server that is the active update source for a section of the topic tree becomes unavailable, an update source on another server is assigned to be the active update source for that section of the topic tree. Failover of active update sources is enabled for any sections of the topic tree that have topic replication enabled.

[Configuring your datagrid provider](#) on page 612

You can configure how the built-in Hazelcast datagrid replicates data within your solution architecture.

[Replication.xml](#) on page 614

This file specifies the schema for the replication properties.

Topic replication

You can use topic replication to ensure that the structure of the topic tree, topic definitions, and topic data are synchronized between servers.

Note: DO NOT use topic replication in production with Diffusion version 5.9. It does not always scale adequately for production use. If you want to use topic replication in production, we advise that you upgrade to Diffusion 6.0 or above which has a completely re-engineered scalable implementation.

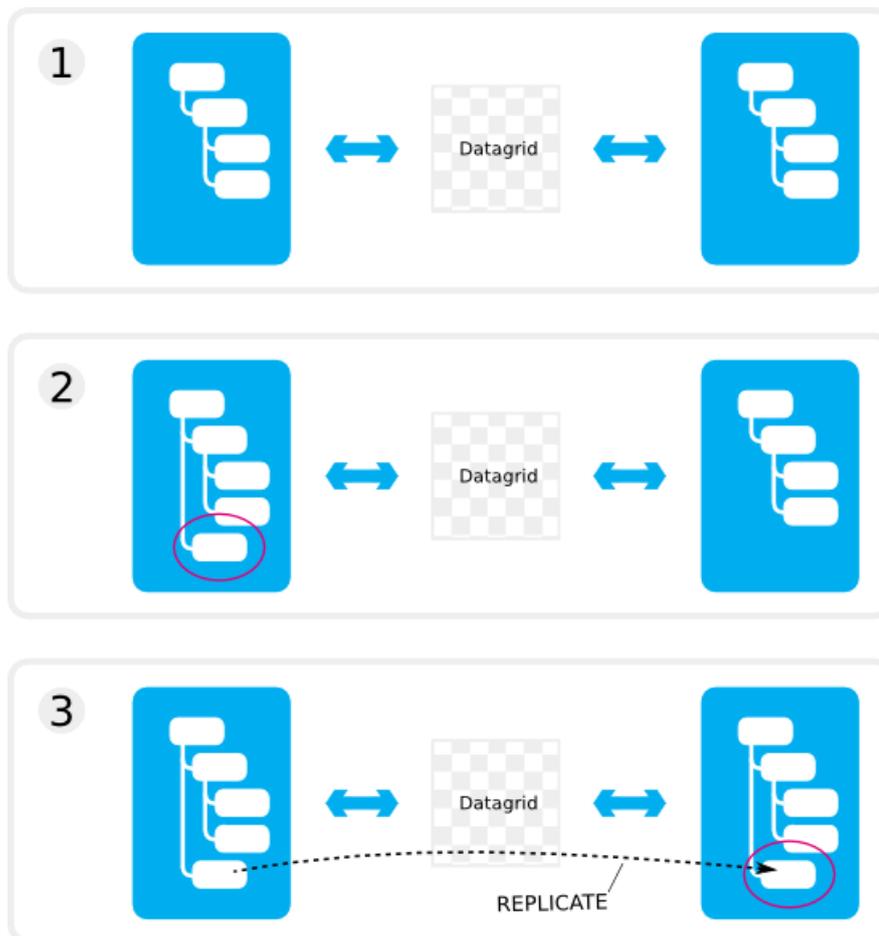


Figure 13: Topic replication

1. Servers with topic replication enabled for a section of the topic tree share information about that section of the topic tree through the datagrid. The topic information and topic data are synchronized on all the servers.
2. A new topic is created on one server in the replicated section of the topic tree.
3. The new topic is replicated on the other servers with identical topic information. When its topic data is updated on the first server, that data is replicated on the other servers.

Considerations

Consider the following factors when using topic replication:

- Replicated topic must be updated by the exclusive update mechanism. Any updating client must register as an update source for the topic.
- Avoid registering a large number of update sources. Do not design your solution so that each topic has its own exclusive update source. This will cause performance problems when starting new servers and joining them to existing clusters, due to the overhead of sharing the update source registrations.
- Only publishing topics can be replicated.
- Replication is supported only for the following types of topic:
 - JSON
 - Binary
 - Single value
 - Record
 - Custom

The custom class for that topic must be available on the classpath of all Diffusion servers replicating that topic.

- Protocol buffer

The compiled `.proto` file that defines the format of the data on a protocol buffer topic must be available on the classpath of all Diffusion servers replicating that topic.

- Stateless
- Slave

A replicated slave topic is linked to a master topic located on the same Diffusion server as the replicated slave topic. This is true whether that master topic is created by replication or directly. If the master topic does not exist on the Diffusion server that a slave topic is replicated to, the slave topic is not created. Instead it is added to a pending set of topics. If the master topic is subsequently created by replication or directly, the slave topic is removed from the pending set and is created on the Diffusion server.

- Replication is not supported for paged topics.
- Any topic that is part of a replicated branch of the topic tree and is not one of the supported types of topic is not replicated. Instead that topic path remains unbound.
- Only topic-wide messages are replicated. Messages sent to a single client or to all clients except one are not replicated.
- Replication of topic information into the datagrid is not automatic. It must be configured at the server. This gives a performance advantage, as you can choose which parts of your topic tree to replicate.
- Replication of topic data can impact performance.
- Do not use topic replication on sections of the topic tree that are owned and updated by publishers. Publishers can make updates to topics that are not replicated or that supersede replicated data. If you use topic replication with topics updated by publishers, this can cause the data on the replicated topics to become unsynchronized.
- Avoid registering requests for topic removal on client session close against replicated topics. When a replicated topic is removed from a server as a result of a client session closing, it is removed from all other servers that replicate that topic. For more information, see [Removing topics with sessions](#) on page 330.

Related tasks

[Configuring the Diffusion server to use replication](#) on page 611

You can configure replication by editing the `etc/Replication.xml` files of your Diffusion servers.

Related reference

[Session replication](#) on page 105

You can use session replication to ensure that if a client connection fails over from one server to another the state of the client session is maintained.

[Failover of active update sources](#) on page 110

You can use failover of active update sources to ensure that when a server that is the active update source for a section of the topic tree becomes unavailable, an update source on another server is assigned to be the active update source for that section of the topic tree. Failover of active update sources is enabled for any sections of the topic tree that have topic replication enabled.

[Configuring your datagrid provider](#) on page 612

You can configure how the built-in Hazelcast datagrid replicates data within your solution architecture.

[Replication.xml](#) on page 614

This file specifies the schema for the replication properties.

Failover of active update sources

You can use failover of active update sources to ensure that when a server that is the active update source for a section of the topic tree becomes unavailable, an update source on another server is assigned to be the active update source for that section of the topic tree. Failover of active update sources is enabled for any sections of the topic tree that have topic replication enabled.

A client must register as an update source to update a replicated topic. Replicated topics cannot be updated non-exclusively. For more information about update sources, see [Updating topics](#) on page 331.

1. A client (CLIENT 1) connects to a Diffusion server (SERVER 1) and registers an update source for a section of the topic tree that has topic replication enabled. This update source is the active update source.
2. Another client (CLIENT 2) connects to another Diffusion server (SERVER 2) and registers an update source for the same section of the topic tree. This update source is a standby update source.
3. The topics on SERVER 2 continue to receive their updates from CLIENT 1 through the datagrid.
4. If SERVER 1 or CLIENT 1 becomes unavailable, the update source registered by CLIENT 2 becomes active. SERVER 2 sends CLIENT 2 a callback to notify it that it is the active update source.

On SERVER 2, the topics in that section of the topic tree receive their updates from CLIENT 2. SERVER 2 reflects this topic data into the datagrid.

Considerations

Consider the following factors when using failover of active update sources:

- If the topic paths that the updating client uses to register an update source do not match the topic paths configured in the `Replication.xml` configuration file of the server, unexpected behavior can occur.
- The mechanism that provides failover of active update sources assumes that all servers have the same configuration and that all control clients implement the same behavior as part of a scalable and highly available deployment. If this is not the case, unexpected behavior can occur.
- Do not use topic replication and failover of active update sources on sections of the topic tree that are owned and updated by publishers. Topic updates sent by publishers are not replicated.

Related concepts

[Updating topics](#) on page 331

A client can use the `TopicUpdateControl` feature to update topics.

Related tasks

[Configuring the Diffusion server to use replication](#) on page 611

You can configure replication by editing the `etc/Replication.xml` files of your Diffusion servers.

Related reference

[Session replication](#) on page 105

You can use session replication to ensure that if a client connection fails over from one server to another the state of the client session is maintained.

[Topic replication](#) on page 108

You can use topic replication to ensure that the structure of the topic tree, topic definitions, and topic data are synchronized between servers.

[Configuring your datagrid provider](#) on page 612

You can configure how the built-in Hazelcast datagrid replicates data within your solution architecture.

[Replication.xml](#) on page 614

This file specifies the schema for the replication properties.

Clients

Consider how you use clients in your solution.

Clients are key to a Diffusion solution. Your solution must include clients as an endpoint to distribute data to. However, clients can also be used for control purposes.

When using clients in your solution, consider the following:

- What types of client you require
- What you use your clients for

Client types

Diffusion provides APIs for many languages and platforms. Some of these APIs have different levels of capability.

A client's type is a combination of the API it uses and the protocol it uses to connect to the Diffusion server.

APIs

JavaScript Unified API

Use this API to develop browser or Node.js clients that can have control capabilities.

Apple Unified API

Use this API to develop mobile clients in Objective-C that do not have control capabilities.

Android Unified API

Use this API to develop mobile clients in Java that can have control capabilities.

Java Unified API

Use this API to develop Java clients that can have control capabilities.

.NET Unified API

Use this API to develop clients in C# that can have control capabilities.

C Unified API

Use this API to develop C clients that can have control capabilities.

Publisher clients

Publisher clients are publishers deployed on a Diffusion server that connect to another Diffusion server as a client. You can use the Java Publisher API to develop a publisher.

DEPRECATED: Android Classic API

Use this API to develop mobile clients in Java that do not have control capabilities.

DEPRECATED: Silverlight Classic API

Use this API to develop browser clients that do not have control capabilities.

DEPRECATED: Flash Classic API

Use this API to develop browser clients in ActionScript® that do not have control capabilities.

DEPRECATED: JavaScript Classic API

You can use this API to develop browser clients that do not have control capabilities. However, we recommend that you use the Unified API instead.

DEPRECATED: Java Classic API

You can use this API to develop Java clients that do not have control capabilities. However, we recommend that you use the Unified API instead.

DEPRECATED: .NET Classic API

You can use this API to develop C# clients that do not have control capabilities. However, we recommend that you use the Unified API instead.

DEPRECATED: iOS Classic API

Use this API to develop mobile clients in Objective-C that do not have control capabilities. However, we recommend that you use the Unified API instead.

DEPRECATED: C Classic API

You can use this API to develop C clients that do not have control capabilities. However, we recommend that you use the Unified API instead.

Protocols

The following protocols, and their secure versions, are available:

WebSocket

The WebSocket implementation provides a browser-based full duplex connection, built on top of WebSocket framing. This complies with the WebSocket standards and is usable with any load balancer or proxy with support for WebSocket.

HTTP Polling

HTTP polling uses HTTP to make a long poll request. Each request remains open until a message is available. More than one message will be returned if available. A separate TCP connection is used to send messages from the client to the server.

DEPRECATED: HTTP Chunked Streaming

HTTP Chunked Streaming provides a streaming connection for messages from the server by using HTTP chunked encoding. A separate TCP connection is used to send messages from the client to the server. This provides two simplex connections, one based on request/response (upstream) and the other streaming data (downstream). This relies on HTTP/1.1 so ensure that network intermediaries such as load balancers are HTTP/1.1 aware.

DEPRECATED: DPT

The DPT protocol (Diffusion protocol over TCP) creates a TCP connection and uses it to send and receive messages in a full duplex way. Load balancers treat these connections as TCP connections.

DEPRECATED: HTTP Full Duplex

HTTP Full Duplex acts like HTTP in the initial connection handshake and acts like DPT for the exchange of messages. HTTP Full Duplex wraps the Diffusion protocol with HTTP request and response headers. Unlike true HTTP, it operates in a full duplex manner. For example, the client can send a response that does not correspond to a request.

This is acceptable to a number of network intermediaries (load balancers and firewalls), and can be a pragmatic way to communicate over a single bi-directional, end-to-end connection via intermediaries that do not accept the WebSocket.

However, we recommend that you use WebSocket instead.

Table 17: Supported protocols by client

Client	WebSocket	HTTP Polling	DEPRECATED: DPT	DEPRECATED: HTTP Full Duplex
JavaScript Unified API	✓	✓		
Apple Unified API	✓			
Android Unified API	✓	✓		
Java Unified API	✓	✓	✓	✓
.NET Unified API	✓		✓	✓
C Unified API	✓		✓	
Publisher client	✓		✓	✓

Using clients

Most clients connect to the Diffusion server only to subscribe to topics and receive message data on those topics. Some clients can also perform control actions such as creating and updating topics or handling events.

Subscribe to topics and receive data

Supported in: JavaScript Unified API, Java Unified API, .NET Unified API, Apple Unified API, Android Unified API, C Unified API, JavaScript Classic API (deprecated), Java Classic API (deprecated), .NET Classic API (deprecated), C Classic API (deprecated), iOS Classic API (deprecated), Android Classic API (deprecated), Flash Classic API (deprecated), Silverlight Classic API (deprecated)

The majority of clients that connect to the Diffusion server, do so to subscribe to topics and receive updates that are published to those topics. These are the clients used by your customers to interact with the data your organization provides.

Control Diffusion, other clients, or the data

Supported in: JavaScript Unified API, Apple Unified API, Java Unified API, .NET Unified API, Android Unified API, C Unified API

You can also develop clients that control aspects of the Diffusion server, other clients, or the data distributed by Diffusion. These are the clients that are used by users or systems inside your organization.

Using clients for control

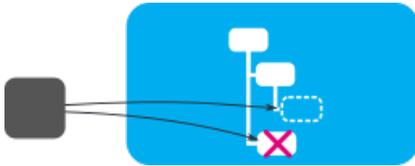
Clients can perform control actions that affect the Diffusion server, other clients, or the data distributed by Diffusion.

Supported in: JavaScript Unified API, Apple Unified API, Java Unified API, .NET Unified API, Android Unified API, C Unified API

Note: Support for these control features can differ slightly between APIs. For more information, see the manual section for the specific API.

When designing your Diffusion solution, decide whether you want to use clients to perform the following actions:

Create and delete topics



Clients can create any type of topic on the Diffusion server. These topics can be created explicitly or dynamically in response to a subscription request from another client.

These topics have the lifespan of the Diffusion server unless the client specifies that the topic be removed when the client session closes.

Clients can also delete topics from the Diffusion server.

You can also use publishers to create and delete topics.

For more information, see [Managing topics](#) on page 295.

Publish updates to topics



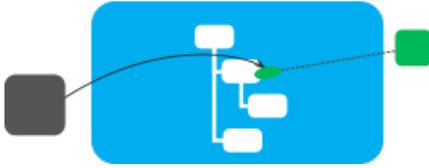
Clients can publish updates to topics that are pushed out to clients subscribed to that topic. These updates can be made exclusively, so that only one client can update a given topic, or non-exclusively, allowing any client to update a given topic.

Note: Do not design your solution to require a large number of update sources (for example, do not give each topic its own exclusive topic updater).

You can also use publishers to publish updates to topics.

For more information, see [Updating topics](#) on page 331.

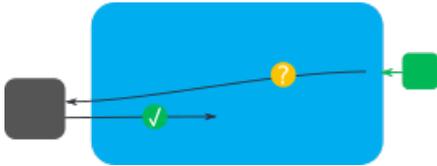
Subscribe other clients to topics



Clients can subscribe other client sessions to topics and also unsubscribe other client session from topics.

For more information, see [Managing subscriptions](#) on page 354.

Authenticate other clients



Clients can provide authentication decisions about whether to allow or deny other client sessions connecting to the Diffusion server. These clients can also assign roles to the connecting client sessions that define the permissions the connecting client has.

You can also use the system authentication handler or an authentication handler located on the Diffusion server to authenticate other clients.

For more information, see [User-written authentication handlers](#) on page 143.

Modify the security information stored on the Diffusion server



Clients can modify the information stored in the security store on the Diffusion server. The security store can be used to specify which permissions are assigned to roles and which roles are assigned to anonymous sessions, and named-principal sessions.

You can also use publishers to modify security information stored on the Diffusion server.

For more information, see [Updating the security store](#) on page 420.

Modify the authentication information stored on the Diffusion server

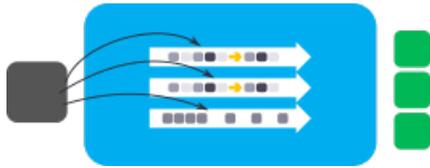


Clients can modify the information stored in the system authentication store on the Diffusion server. The system authentication store can be used to specify which principals a client session can use to connect and what roles are assigned to an authenticated client session.

You can also use publishers to modify authentication information stored on the Diffusion server.

For more information, see [Updating the system authentication store](#) on page 407.

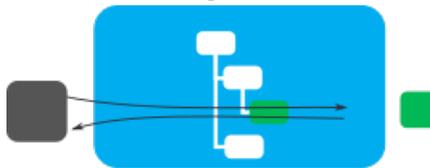
Manage the flow of data to clients



Updates are pushed to subscribing clients through client queues. Clients can receive notifications when client queues reach a certain threshold. These clients can manage the client queues by turning on throttling or conflation for the queue.

For more information, see [Managing clients](#) on page 431.

To handle messages sent to topic paths by clients and send messages to specific clients



Clients can send messages through topic paths to specific clients. Clients can also register to handle messages that are sent to a topic path. Messages sent using topic paths do not update the topic.

You can also use publishers to handle messages on topic paths and send messages to clients.

For more information, see [Messaging to sessions](#) on page 375.

User-written components

Consider which components you must develop to create your solution.

Publishers

Consider whether to develop publishers to distribute data in your solution.

Publishers are written in Java and deployed on the Diffusion server.

You can deploy one or more publishers on a Diffusion server. A publisher can provide the behavior of one or more topics but a topic can belong to only one publisher. The publisher infrastructure is provided by Diffusion and the behavior is provided by the user by writing a publisher.

Why use publishers?

Publishers enable you to manage your topics and updates, and customize their behavior. Unlike clients, publishers are located on the Diffusion server so can communicate more swiftly with the server and do not become disconnected from the server.

Publishers provide the following capabilities:

- Create and delete topics
- Publish updates to topics
- Define topic load data
- Provide topic state to fetch requests
- Send and receive messages to topic paths
- Handle requests for topics that do not exist
- Validate client connections
- Receive notifications of client events

- Subscribe clients to topics

Considerations when using a publisher

Publishers can only be written in Java.

Other user-written components

Diffusion provides many opportunities to create user-written components that define custom behavior. Consider whether to develop any of these components as part of your solution.

Server-related components

All of these components must be created as Java classes and put on the classpath of the Diffusion server.

Authentication handlers

These components handle authentication of clients that connect to the Diffusion server or change the principal that they use to connect to the Diffusion server. If the client connection is allowed, the authentication handler assigns roles to the client session.

You can have zero, one, or many authentication handlers configured on your Diffusion server.

For more information, see [Developing a local authentication handler](#) on page 508 and [Developing a composite authentication handler](#) on page 510.

Note: Local authentication handlers, on the Diffusion server, can be written only in Java. However, control authentication handlers that are part of a client whose API supports Authentication Control can be written in other languages.

Hooks

Startup and shutdown hooks are called by the Diffusion server. The startup hook is instantiated and called as the Diffusion server starts and before publishers are loaded. The shutdown hook is called as the Diffusion server stops.

For example, you can use a shutdown hook to persist some aspect of the state of the Diffusion server to disk.

HTTP service handlers

These components handle HTTP requests as part of an HTTP service in the Diffusion server's built-in web server. Provide a user-written HTTP service handler to enable the Diffusion web server to handle any kind of HTTP request.

Thread pool handlers

These handlers define custom behavior in the Diffusion server related to the inbound thread pool.

You can provide a rejection handler that customizes the behavior when a task cannot be run by the thread pool. By default, if a task cannot be run by the inbound thread pool — for example, if the thread pool is overloaded — the calling thread blocks until there is space on the queue.

You can provide a notification handler that receives notifications when events occur on the inbound thread pool.

DEPRECATED: Authorization handlers

Authorization handlers are deprecated and have been replaced by role-based authorization. For more information, see [Role-based authorization](#) on page 130.

An authorization handler controls authorization and permissions for clients and users. You can have zero or one authorization handler configured on your Diffusion server.

Authorization handlers can be written only in Java.

Topic- and data-related components

All of these components must be created as Java classes and put on the classpath of the Diffusion server.

Message matchers

Message matchers are used to customize conflation behavior. These classes that define how the Diffusion server locates messages on a client's message queue that are to be conflated.

By default, messages for conflation are matched if they are on the same topic.

For more information, see [Conflation](#) on page 93.

Message mergers

Message mergers are used to customize conflation behavior. These classes that define how the Diffusion server conflates matching messages.

By default, the older of the matching messages is removed.

For more information, see [Conflation](#) on page 93.

Comparators and collators

These components define the behavior of an ordered paged topic. Comparators implement `java.util.Comparator` and can customize the ordering of lines in a paged string or paged record topic. Collators implement `java.text.RuleBasedCollator` and can define the ordering of lines in a paged record topic.

For more information, see .

Custom field handlers

These components handle the data in custom fields of your record topics. A custom field handler can define the default value of a custom field, parse incoming data into the format required by the custom field, and compare data in custom fields of the same type for equality.

For more information, see [Metadata](#) on page 75.

Custom topic data handlers

These components handle the behavior of a custom topic. When you create a custom topic, you provide a custom topic handler that defines how the topic data is maintained, compared, and sent to subscribing clients.

For more information, see [DEPRECATED: Custom topics](#) on page 80.

Routing topic handlers

These components handle the behavior of a routing topic. When you create a routing topic, you provide a routing topic handler that, when a subscription to the routing topic is made, maps the routing topic to another topic on the Diffusion server on a client-by-client basis.

For more information, see [Routing topics](#) on page 79.

Service handlers

These components handle the behavior of a service topic. When you create a service topic, you provide a service handler that receives requests from clients on the topic and provides responses through the topic.

For more information, see [DEPRECATED: Service topics](#) on page 84.

Third party components

Diffusion interacts with a number of third-party components. Consider how you use these components as part of your solution.

Load balancers

We recommend that you use load balancers in your Diffusion solution.

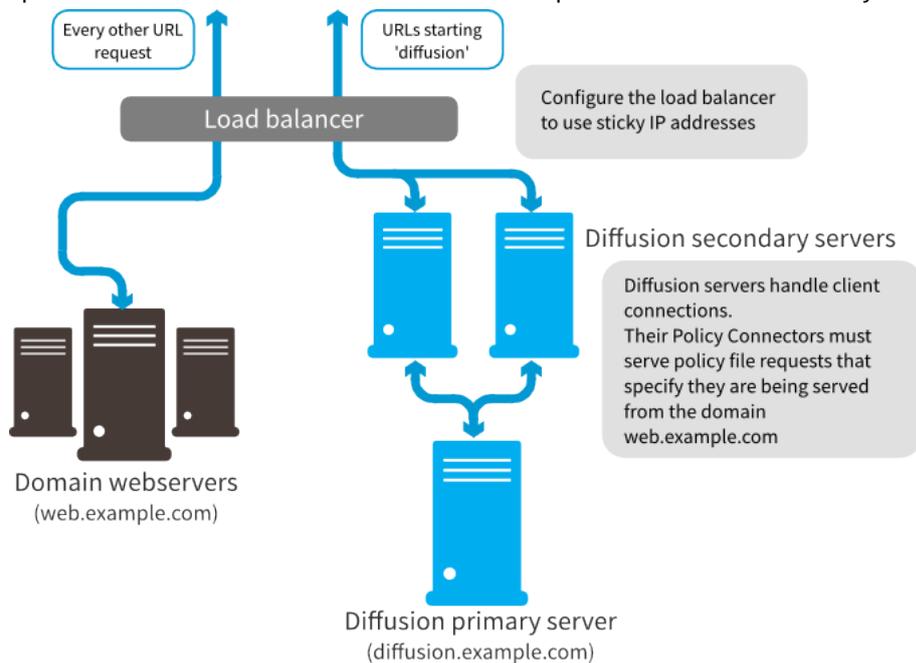
Why use load balancers?

Balancing client traffic across multiple Diffusion servers

Distribute incoming requests from clients fairly over the Diffusion servers in your solution and ensure that all traffic for a specific client is routed to the same Diffusion server.

Compositing URL spaces

If your Diffusion servers are located at a different URL to the Diffusion browser clients hosted by your web servers, you can use a load balancer to composite the URL spaces. This enables Diffusion solutions to interoperate with browser security.



SSL offloading

Diffusion clients can connect to your solution using TLS or SSL. The TLS/SSL can terminate at your load balancer or at your Diffusion server. Terminating the TLS at the load balancer reduces CPU cost on your Diffusion servers.

Considerations when using load balancers

Do not use connection pooling for connections between the load balancer and the Diffusion server. If multiple client connections are multiplexed through a single server-side connection, this can cause client connections to be prematurely closed.

In Diffusion, a client is associated with a single TCP/HTTP connection for the lifetime of that connection. If a Diffusion server closes a client, the connection is also closed. Diffusion makes no distinction between a single client connection and a multiplexed connection, so when a client sharing a multiplexed connection closes, the connection between the load balancer and Diffusion is closed, and subsequently all of the client-side connections multiplexed through that server-side connection are closed.

Multiple users masquerading behind a proxy or access point can have the same IP address, and all requests from clients with that IP address are routed to the same Diffusion instance. Load balancing still occurs, but some hosts might be unfairly loaded.

Web servers

Consider how to use web servers as part of your Diffusion solution.

If you are using Diffusion in conjunction with a web client or web application, this web client or application must be hosted on a web server.

While the Diffusion server includes a web server, this internal web server is intended for the following uses:

- Hosting the Diffusion landing page, demos, and monitoring console
- Providing an endpoint for the HTTP-based transports used by Diffusion clients
- Optionally, hosting a static page you can use to check the status of the Diffusion server

For more information, see [Diffusion web server](#) on page 649.

Do not use the Diffusion web server as the host for your production website. Instead use a third-party web server.

There are two ways you can use Diffusion with a third-party web server:

- As separate, complementary components in your solution.
- With the Diffusion server deployed inside a web application server.

Use a separate web server with the Diffusion server

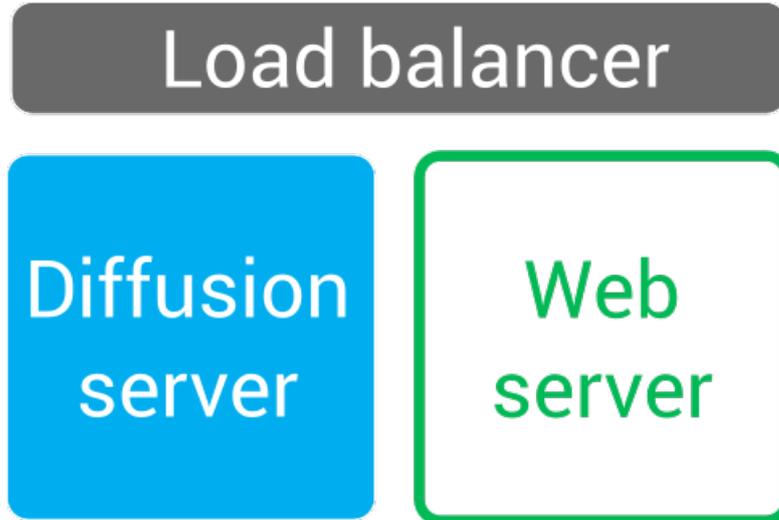


Figure 14: Using a web server with Diffusion

Why use a separate web server with Diffusion?

You can use a third-party web server to host your Diffusion browser clients.

A third-party web server provides the following advantages over the lightweight internal Diffusion web server:

- Greater ability to scale
- More comprehensive security
- Server-side code and dynamic web pages

If your organization already uses a third-party web server, Diffusion augments this component instead of replacing it.

Using a separate web server with the Diffusion server provides the following advantages over deploying the Diffusion server inside a web application server:

- The load balancer set up is simpler
- You can scale the number of Diffusion servers and the number of web servers in your solution independently and more flexibly
- The web server and the Diffusion server do not share a JVM process, which can cause performance advantages
- The web server and the Diffusion server are independent components, which makes them unlikely to be affected by any problems that occur in the other component

For more information, see [Hosting Diffusion web clients in a third-party web server](#) on page 651.

Considerations when using a separate web server with the Diffusion server

If your web server hosts a client that makes requests to a Diffusion server in a different URL space, you can use a load balancer to composite the URL spaces and interoperate with browser security or you can set up cross-domain policy files that allow requests to the different URL space.

When the Diffusion server is separate from the web server, the web server has no access to the Diffusion Publisher API.

Deploy the Diffusion server inside a web application server

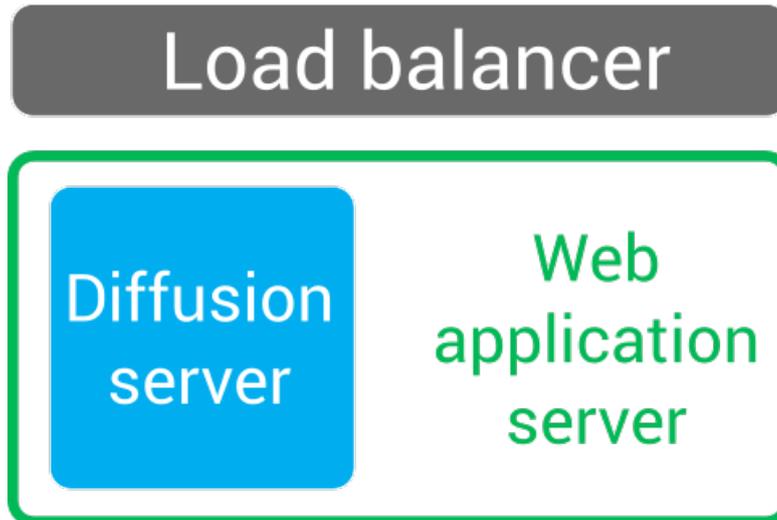


Figure 15: Deploying Diffusion inside a web application server

Why deploy Diffusion inside a web application server?

You can also host your Diffusion server inside a third-party web application server that has the capability to host Java servlets.

This provides the advantage of only setting up a single server and having a single application to manage when hosting your web application.

The web application server has access to the Diffusion Publisher API of the Diffusion server it hosts. This enables your web application to use server-side logic to include Diffusion information in your web pages.

For more information, see [Running the Diffusion server inside of a third-party web application server](#) on page 652.

Considerations when deploying the Diffusion server inside a web server

When running inside a web application server, the Diffusion server still requires its own internal web server to communicate with clients over HTTP-based transports.

Your web application and your Diffusion server, while hosted by the same server, can have different port numbers. This can result in cross-origin security concerns for some browsers. You can use a load balancer to composite the ports or you can set up cross-domain policy files that allow requests to the different ports.

The load balancer configuration can be more complex when deploying the Diffusion server inside a web application server. If you have multiple web application server and Diffusion server pairs, configure your load balancer to ensure that requests from a client always go to a pair and not to the web application server from one pair and the Diffusion server from another pair.

When running the Diffusion server inside of a web application server, the Diffusion server and the web application server share a JVM process. This can lead to large GC pauses. Ensure that you test this configuration and tune the JVM.

Related concepts

[Web servers](#) on page 648

Diffusion incorporates its own basic web server for a limited set of uses. The Diffusion server also interacts with third-party web servers that host Diffusion web clients. The Diffusion server is also capable of being run as a Java servlet inside a web application server.

[Diffusion web server](#) on page 649

Diffusion incorporates its own web server. This web server is required to enable a number of Diffusion capabilities, but we recommend that you do not use it to host your production web applications.

[Configuring the Diffusion web server](#) on page 616

Use the `WebServer.xml` and `Aliases.xml` configuration files to configure the behavior of the Diffusion web server.

[Configuring Diffusion web server security](#) on page 617

When configuring your Diffusion web server, consider the security of your solution.

[Running the Diffusion server inside of a third-party web application server](#) on page 652

Diffusion can run as a Java servlet inside any Java application server.

[Hosting Diffusion web clients in a third-party web server](#) on page 651

Host Diffusion web clients — clients written using the JavaScript, Flash, or Silverlight APIs — on a third-party web server to enable your customers to access them.

Related reference

[WebServer.xml](#) on page 617

This file specifies the schema for the web server properties.

Push notification networks

Consider whether your solution will interact with push notification networks.

Push notification networks can relay data to a client, even when that client is not running.

Diffusion Push Notification Bridge

The Push Notification Bridge is a Diffusion client that subscribes to topics on behalf of other Diffusion clients and uses a push notification network to relay topic updates to the device where the client application is located.

The Push Notification Bridge supports the following push notification networks:

- APNS
- GCM

For more information about how the Push Notification Bridge works, see [Push Notification Bridge](#) on page 660.

Why use the Push Notification Bridge

Diffusion clients on Android or iOS devices might not be running all the time to conserve battery or to enable other processes to run. However, the user might still want to receive realtime updates while the Diffusion client is not running.

By using push notification networks, Diffusion can deliver data to destinations on these devices at any time.

Considerations when using the Push Notification Bridge

- The Push Notification Bridge supports only single value topics.
- Push notification networks identify an app on a device (*a push notification destination*), not an individual user or session.

- If a client requests push notification for a topic and also subscribes to that topic, when the client is connected to Diffusion it receives topic updates once through the Diffusion server and once through the push notification network. The client must handle removing the duplicate messages from the information presented to the user.
- Push notification networks currently limit the size of notifications to 2 KB or less.
- By default, the bridge does not persist the notification subscription requests sent by the clients. If the bridge stops and restarts, this information is lost and notifications are no longer sent.

To ensure that the notification subscriptions are persisted by the bridge, implement a persistence solution. For more information, see [Push Notification Bridge persistence plugin](#) on page 511.

Related concepts

[Push Notification Bridge persistence plugin](#) on page 511

The Push Notification Bridge stores subscription information in memory. To persist this information past the end of the bridge process, implement a persistence plugin.

[Example: Send a request message to the Push Notification Bridge](#) on page 372

The following examples use the Unified API to send a request message on a topic path to communicate with the Push Notification Bridge. The request message is in JSON and can be used to subscribe or unsubscribe from receiving push notifications when specific topics are updated.

[Push Notification Bridge](#) on page 660

The Push Notification Bridge is a Diffusion client that subscribes to topics on behalf of other Diffusion clients and uses a push notification network to relay topic updates to the device where the client application is located.

JMS

Consider whether to incorporate JMS providers into your solution.

If a third-party JMS provider is part of your solution, you can map JMS queues and topics to Diffusion topics by using the Diffusion JMS adapter.

We support integration with JMS providers that conform to version 1.1 or later of the JMS specification.

The following JMS products have been tested and are certified by Push Technology for use with the JMS adapter:

- Apache ActiveMQ v5.11
- IBM MQ v8

Why use a third-party JMS provider

If you are already using a JMS provider to move data in your internal system, you can integrate it with Diffusion to distribute that data to clients and users outside of your organization.

Diffusion JMS adapter

The JMS adapter for Diffusion, enables Diffusion clients to transparently send data to and receive data from destinations (topics and queues) on a JMS server. It is highly configurable and can support the following scenarios:

Pub-sub

Messages on a JMS destination can be published to a Diffusion topic. For more information, see [Publishing using the JMS adapter](#) on page 681.

Messaging

Messages can be sent between JMS destinations and Diffusion clients.

- A message on a JMS destination can be relayed to a Diffusion client through a topic path.
- A Diffusion client can send a message on a topic path that is relayed to a JMS destination.

For more information, see [Sending messages using the JMS adapter](#) on page 682.

Request-response

The JMS provider can integrate with services that interact using an asynchronous request-response pattern. Diffusion exposes these JMS services through its messaging capabilities. For more information, see [Using JMS request-response services with the JMS adapter](#) on page 685.

Data that flows between JMS and Diffusion must be transformed. JMS messages contain headers, properties, and a payload. Diffusion messages contain just content. For more information about how data is transformed between JMS and Diffusion, see [Transforming JMS messages into Diffusion messages or updates](#) on page 678.

Running the JMS adapter in the Diffusion server or as a standalone application

The JMS adapter is provided in the following forms:

Within the Diffusion server

The JMS adapter can be configured to run as part of the Diffusion server process. A JMS adapter running within the Diffusion server cannot become disconnected from the Diffusion server.

As a standalone client

The JMS adapter is a Java application that can be run on any system and acts as a client to the Diffusion server. Topics created by the JMS adapter running as a standalone client are not deleted from the Diffusion server if the JMS adapter becomes disconnected. You can use this capability to design a highly available solution.

For more information, see [JMS adapter](#) on page 677.

Considerations when using the JMS adapter

Note: If you currently use the legacy JMS adapter version 5.1, you must reimplement to use this JMS adapter. No migration path for configuration is available.

Topics defined and created by the JMS adapter when it runs within the Diffusion server are removed when the JMS adapter is stopped.

Topics defined and created by the JMS adapter when it runs as a standalone client are not deleted from the Diffusion server when the JMS adapter client session is closed.

The JMS adapter supports interaction with Diffusion topics that are either stateful (single value) or stateless topics.

Only textual content and JMS TextMessages are supported. Binary content is not currently supported.

You cannot currently publish data to a Diffusion topic and have it sent to a JMS destination.

Data must be transformed between JMS messages and Diffusion content.

If multiple Diffusion servers subscribe to the same JMS queue in a request-response scenario, there is the risk of one server consuming messages intended for another server. Use JMS selectors to ensure that the JMS adapter only receives those messages intended for it.

The creation of temporary queues and topics by the JMS adapter is not currently supported.

Durable subscriptions are not supported.

JMS transactions are not supported.

The only acknowledgment mode that is supported is `AUTO_ACKNOWLEDGE`.

Session properties are not currently supported. The exception is the `$Principal` property.

Related concepts

[Transforming JMS messages into Diffusion messages or updates](#) on page 678

JMS messages are more complex than Diffusion content. A transformation is required between the two formats.

[Sending messages using the JMS adapter](#) on page 682

The JMS adapter can send messages from a Diffusion client to a JMS destination and messages from a JMS destination to a specific Diffusion client.

[Publishing using the JMS adapter](#) on page 681

The JMS adapter can publish data from a JMS destination onto topics in the Diffusion topic tree.

[Using JMS request-response services with the JMS adapter](#) on page 685

You can use the messaging capabilities of the JMS adapter to interact with a JMS service through request-response.

[Configuring the JMS adapter](#) on page 686

Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Related reference

[JMSAdapter.xml](#) on page 695

This file specifies the schema for the configuration required by the JMS adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Example solutions

This section includes some example solutions that you can refer to when designing your own solution.

Example: Simple solution

This solution uses a firewall to restrict incoming traffic and a load balancer to balance the traffic between multiple Diffusion servers.

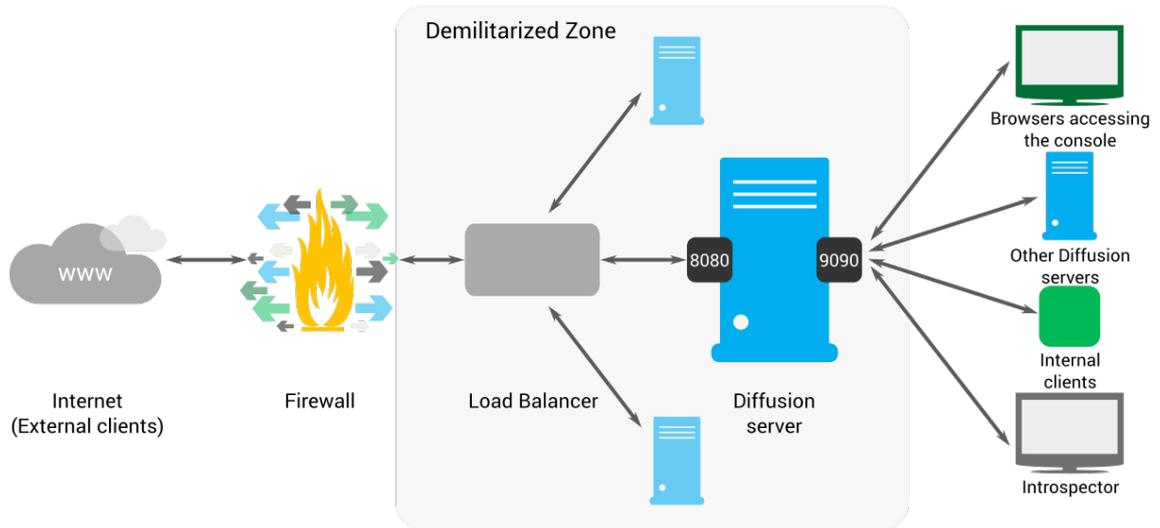


Figure 16: A simple solution

- Client applications can connect to Diffusion from the internet through a firewall.
- The firewall protects the DMZ from unwanted traffic. It allows connections on port 80 and redirects these connections to port 8080.
- The load balancer balances the Diffusion connections between all the Diffusion servers in the DMZ. You can also use the load balancer to filter the URL space and to perform SSL offloading.
- The Diffusion servers receive connections from external clients on port 8080. This port is protected by an authentication handler that performs strict authentication on the incoming connections. Authentication handlers can be local to the server or part of a control client.
- The Diffusion servers receive connections from internal clients on another port, for example 9090. The authentication controls on this port are less strict because these connections come from within your network. Internal connections can come from any of the following components:
 - Browsers accessing the Diffusion console
 - Other Diffusion servers that are not located in the DMZ.
 - Internal clients, such as control clients.
 - The Introspector

Example: A solution using clients

Clients with different uses connect to the Diffusion server in this example solution.

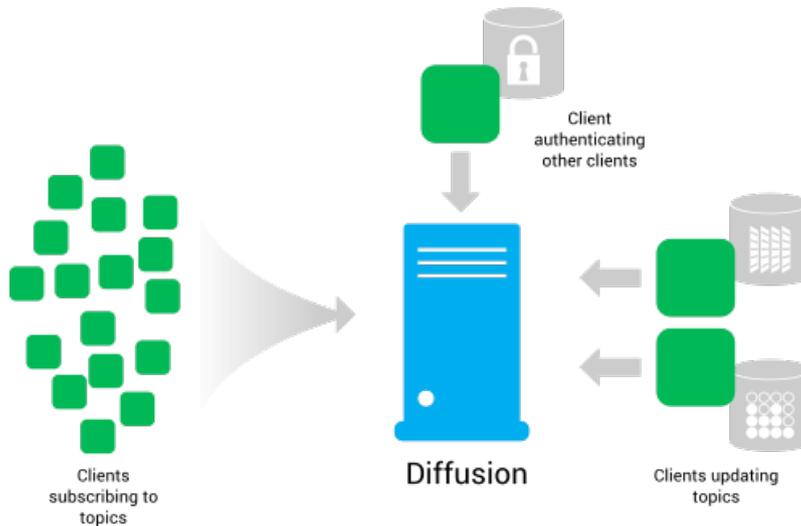


Figure 17: Clients for different purposes

This example solution uses three kinds of client, each for a different purpose:

Clients subscribing to topics

These clients are used by your customers to receive the data you distribute. You can use any of the provided APIs to create these, depending on how your customers want to access your data. For example,

- Use the Apple API to create an iPhone app.
- Use the JavaScript API to create a browser client.

These clients subscribe to the topics that are of interest to your customer, receive updates published on these topics, and display the information to your customers.

Clients creating and updating topics

These clients are used by your organization to distribute your data. You must use an API that provides control features to create these clients. For example, the JavaScript API or the .NET API.

These clients create the topics required to support your data structure and to publish data from your data sources to topics on the Diffusion server.

Clients authenticating other clients

These clients are used by your organization to authenticate connections from other clients. You must use an API that provides control features to create these clients. For example, the Java API.

These clients are called by the Diffusion server to provide an authentication decision when another client connects to the Diffusion server anonymously or with a principal. In addition to deciding whether the other client is allowed to connect, the authenticating client can assign roles to the client session.

The authenticating client can use information stored elsewhere in your system, for example in an LDAP server, to make the authentication decision and assign roles.

Example: Scalable and resilient solution

This solution uses replication to share information between primary servers and make them highly available. The solution also uses fan-out to spread the data from the primary servers behind the firewall to secondary servers in the DMZ.

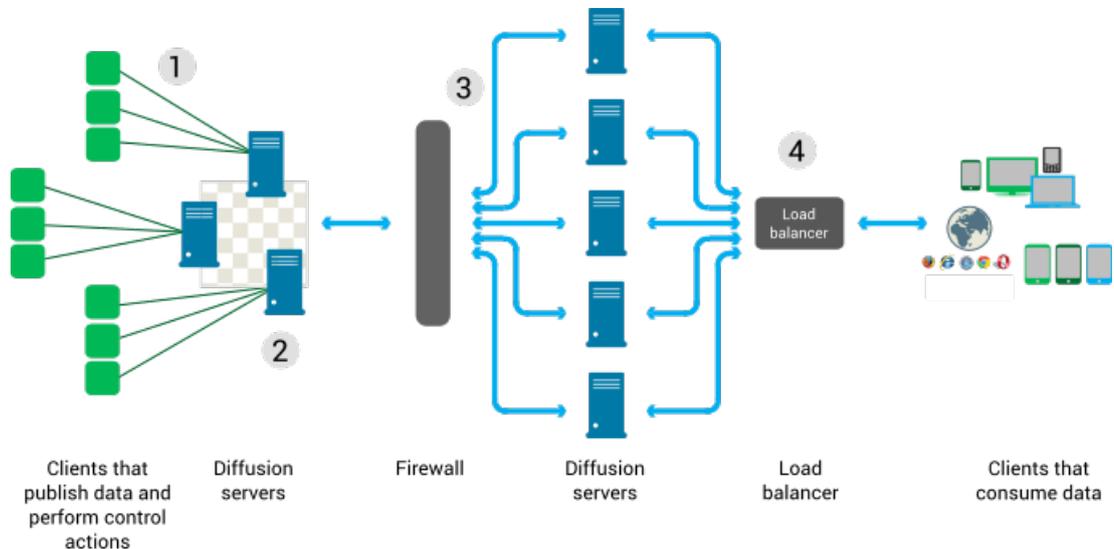


Figure 18: Architecture using replication and fan-out

1. Three clients register handlers with each of the Diffusion servers behind the firewall. These clients can be located on the same system as the server or on remote systems. Each Diffusion server load balances requests between clients that have registered to handle requests of that type. If one of the clients becomes unavailable, the requests can be directed to another client. You can connect more client sessions to deal with higher volumes of requests.
2. The Diffusion servers inside the firewall replicate information into a datagrid. If a Diffusion server that was handling a client session or topic becomes unavailable, the responsibility for that client session or topic can be passed to another Diffusion server that has access to all the information for that session or topic through the datagrid.
3. The Diffusion servers outside of the firewall, in the DMZ, are configured to use automated fan-out to connect to the Diffusion servers inside the firewall. Specified topics on the primary server are fanned out to the secondary servers.
4. You can use a load balancer to spread requests from subscribing clients across many secondary Diffusion servers. If a server becomes unavailable, clients can be directed to another server.

Security

Diffusion secures your data by requiring client sessions to authenticate and using role-based authorization to define the actions that a client can perform.

Concepts

Principal

The principal is a user or system user that has an identity that can be authenticated.

When a principal is authenticated it becomes associated with a session. The default principal that is associated with a session is ANONYMOUS.

Session

A session is a set of communications between the Diffusion server and a client.

Permission

A permission represents the right to perform an action on the Diffusion server or on data.

Role

A role is a named set of permissions and other roles. Principals and sessions can both be assigned roles.

Role hierarchy

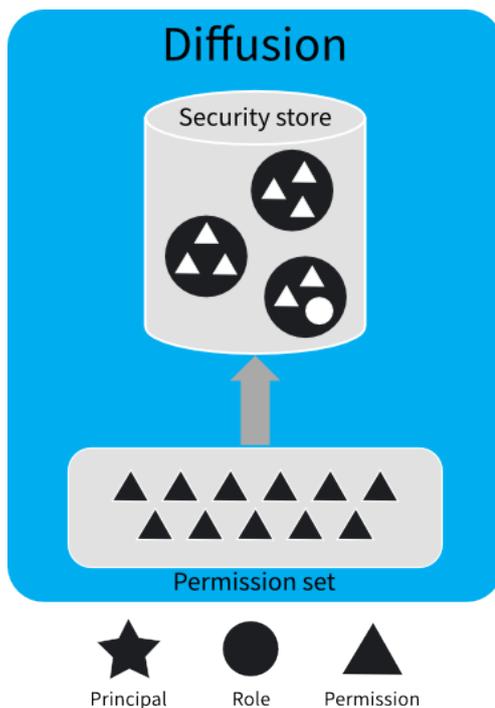
Roles are hierarchical. A role can include other roles and, by doing so, have the permissions assigned to the included roles. A role cannot include itself, either directly or indirectly – through a number of included roles.

Role-based authorization

Diffusion restricts the ability to perform actions to authorized principals. Roles are used to map permissions to principals.

Associating permissions with roles

The association between roles and permissions is defined in the security store.



A fixed set of permissions is defined by the Diffusion server. These permissions are used to control access to actions and data on the Diffusion server.

Roles are used to associate permissions to principals. Permissions are assigned to roles, and roles are assigned to principals.

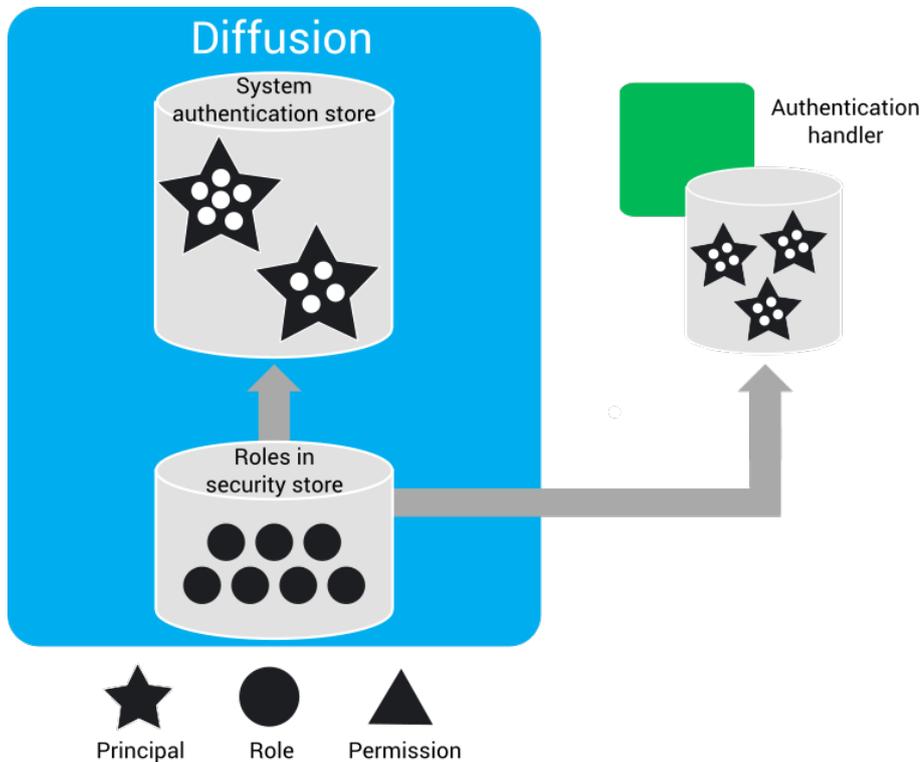
A role can be assigned zero, one, or many permissions. The same permission can be assigned to multiple roles. Roles can also include other roles to form a role hierarchy, and so inherit their permissions. The permissions assigned to a role and the role hierarchy are defined in the security store.

You can update the security store by editing the store file, `installation_directory/etc/Security.store`, and restarting the Diffusion server.

You can update the security store from a client using the SecurityControl feature.

Associating roles with principals

The association between roles and principals is defined in the system authentication store or by user-written authentication handlers.



The association between principals and roles is defined in the following ways:

- In a user-defined store that your user-written authentication handlers refer to. For example, an LDAP server.
- A user-written authentication handler can also hard code the relationship between principals and roles, if that is appropriate.
- In the system authentication store of the Diffusion server

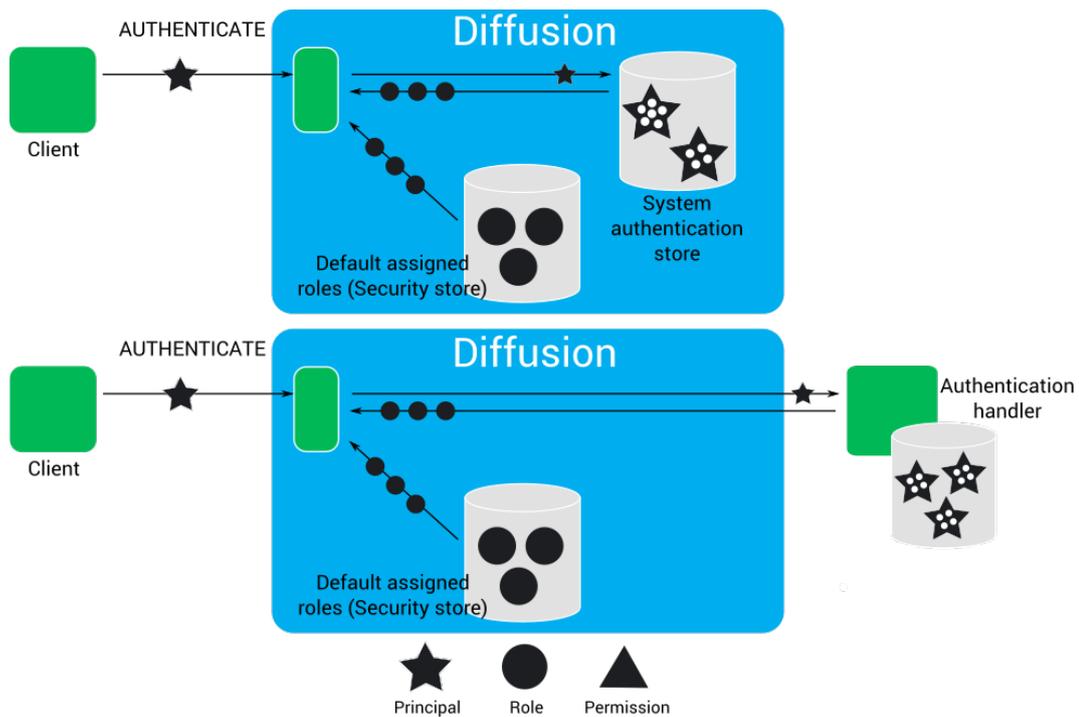
The system authentication store is designed to hold information about Diffusion administration users and system clients. It can manage hundreds or perhaps thousands of principals, but does not provide the administration tools necessary to support millions of principals. We recommend that you delegate such "internet scale" use cases to a third-party identity provider using a custom authentication handler. For example, by using the OAuth or OpenID protocol.

You can update the system authentication store in the following ways:

- From a client using the SystemAuthenticationControl feature.
- By editing the store file, `installation_directory/etc/SystemAuthentication.store`, and restarting the Diffusion server.

Assigning roles to client sessions

Roles are assigned to a new client session after client authentication.



The roles assigned to a client session determine the actions that client session can perform.

A client session is assigned roles in the following ways:

- If the client session connects to the Diffusion server anonymously, the client session is assigned the default assigned roles for the ANONYMOUS principal.

Anonymous authentication can be enabled or disabled in the system authentication store. If enabled, roles can also be specified.

- When a client session authenticates with a principal, the client session can be assigned the following roles:
 - The default assigned roles for a named principal.
 - The set of roles assigned to a principal by the authentication handler that accepts the client session's authentication request. This authentication handler can be one of the following types:
 - The system authentication handler, in which case the roles that are assigned are those associated with that principal in the system authentication store.
 - A user-written authentication handler, in which case the roles that are assigned are those defined by the handler or a user-defined store.

For example: A client session authenticates with the Diffusion server using the principal Armstrong. The first authentication handler that is called is a user-written authentication handler. This authentication handler abstains from the authentication decision, so does not assign roles to the client session. The next authentication handler that is called is the system authentication handler. The system authentication handler does not abstain from the authentication decision. It uses the information in the system authentication store to decide to allow the authentication request. In the system authentication store, the principal Armstrong is associated with the roles ALPHA, BETA, and EPSILON. These roles are assigned to the client session.

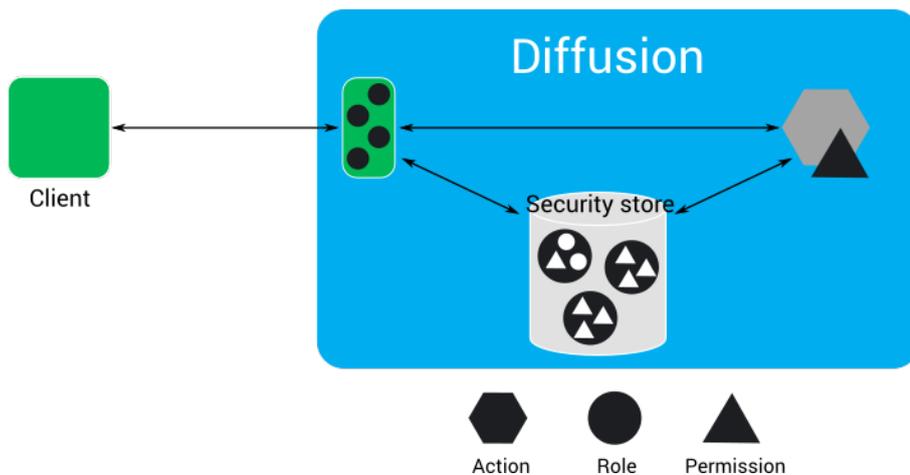
After the authentication request has been allowed, no further authentication handlers are called to make a decision or assign roles. However, the Diffusion server also assigns the default assigned roles for a named principal to the client session. The default assigned roles defined in the security store are GAMMA and RHO.

After authenticating with the principal Armstrong, the client session has the following roles assigned to it:

- ALPHA
- BETA
- EPSILON
- GAMMA
- RHO

Authorizing actions

When a client requests to perform an action or access data that requires a permission, the Diffusion server checks whether the client session is assigned a role that includes the required permission.



The client requests to perform an action. If the action requires that the client session have a permission, the Diffusion server checks what roles the client session is assigned and checks in the security store whether any of these roles have the required permission.

For example: A client requests to subscribe to the topic A/B/C. To subscribe to a topic, a client session must have the `select_topic` permission for that topic. The client session has the ALPHA and BETA roles. In the security store, the ALPHA role does not include the `select_topic` permission, but the BETA role does include the `select_topic` permission for the A/B/C topic. Because the client session is assigned the BETA role, it has the required permission and can subscribe to the topic.

Related concepts

[Authentication](#) on page 140

You can implement and register handlers to authenticate clients when the clients try to perform operations that require authentication.

DEPRECATED: [Authorization handlers](#) on page 147

An authorization handler can control authorization and permissions for clients and users.

[Securing the console](#) on page 150

Configuration is required to enable additional security around connections from the Diffusion console.

Related reference

[Configuring user security](#) on page 588

You can use the `Security.store` and `SystemAuthentication.store` files in the `etc` directory of your Diffusion server to configure the security roles and how they are assigned.

Permissions

The actions a client session can take in Diffusion are controlled by a set of permissions. These permissions are assigned to roles.

Permissions can have one of the following scopes:

Topic

Permissions at topic scope apply to actions on a topic.

Topic-scoped permissions are defined against topic branches. The permissions that apply to a topic are the set of permissions defined at the most specific branch of the topic tree.

Global

Permissions at global scope apply to actions on the Diffusion server.

Topic permissions

The topic-scoped permissions are listed in the following table:

Table 18: List of topic-scoped permissions

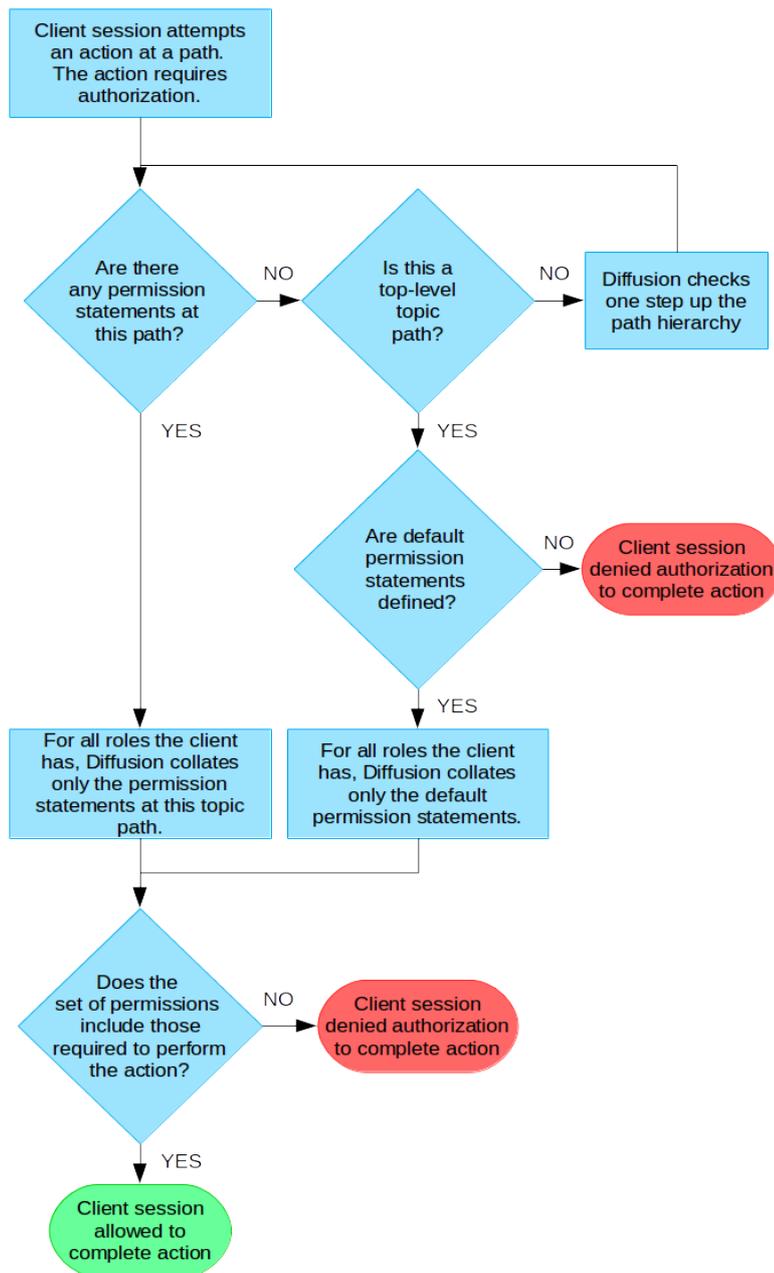
Name	Description
<code>select_topic</code>	Use a topic selector that selects the topic path. A session must have this permission for the path prefix of any topic selector used to subscribe or fetch.
<code>read_topic</code>	Grant read access to the topics. If a session does not have this permission for a topic, that topic does not match subscriptions and is excluded from fetch requests. Also the topics details cannot be retrieved.
<code>update_topic</code>	Update topics at or below a topic branch.
<code>modify_topic</code>	Create or modify topics at or below a topic branch.
<code>send_to_message_handler</code>	Send a topic message to the server for a topic at or below a topic branch.
<code>send_to_session</code>	Send a message to a client session for a topic at or below a topic branch.

Understanding topic-scoped permissions

Topic-scoped permissions are assigned to roles for specific topic paths. The permission assignment applies to all descendant topics, unless there is a more specific assignment.

To evaluate whether a client session has access to a permission for a topic, the Diffusion server starts at that topic and searches up the tree to find the nearest permission assignment. The first assignment is the only one considered, even if the client has roles involved in assignments further up the topic tree.

Default topic-scope assignments can also be defined. These are used if no path assignment matches.



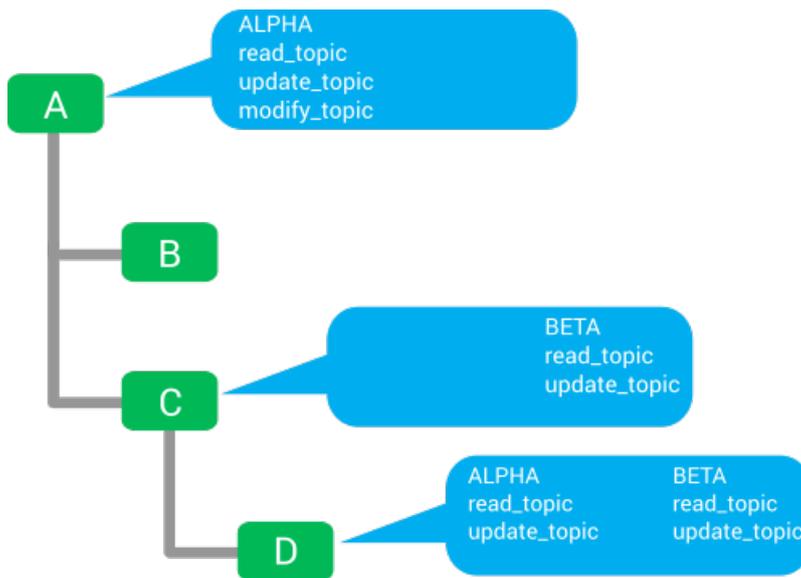


Figure 19: Topic scope example

In this example, client sessions with the role ALPHA have the following permissions on each topic in the topic tree:

A

A permission set is defined for the topic path A.

These permissions give client sessions with the ALPHA role read_topic, update_topic, and modify_topic permissions on the topic A.

A/B

No permission set is defined for the topic path A/B. In this case, the permissions at the most specific scope are those defined for the topic path A

These permissions give client sessions with the ALPHA role read_topic, update_topic, and modify_topic permissions on the topic B.

A/C

A permission set is defined for the topic path A/C. These permissions do not include any permissions for the ALPHA role.

Client sessions with the ALPHA role have no permissions on the topic C. Permissions are defined for the ALPHA role at a less specific scope. However, these permissions are not referred to or inherited if any permissions are defined at a more specific scope. Only the most specific set of permissions is used. In this case, those permissions are only for the BETA role and not the ALPHA role.

A/C/D

A permission set is defined for the topic path A/C/D.

These permissions give client sessions with the ALPHA role read_topic and update_topic permissions on the topic D.

The role ALPHA has only these permissions even though at A/C the role has no permissions defined and at A the role has additional permissions. Only the most specific set of permissions is used.

The BETA role also has permissions defined at this scope. These permissions do not affect the permissions that the ALPHA role has at this scope.

Understanding the select_topic and read_topic permissions

The default configuration grants the select_topic and read_topic permissions to all sessions then protects the topic tree below the Diffusion topic using the OPERATOR role. You can alter this configuration to protect sensitive topics.

A session that does not have the select_topic permission for a particular topic path cannot subscribe directly to topics at that path. However, the session can be independently subscribed to that topic by a control session that has modify_session permission in addition to the select_topic permission for that topic path. The subscribed session requires the read_topic permission for that topic for the subscription to the topic to occur. The control session cannot subscribe a session to a topic if that session does not have the read_topic permission for the topic. When this occurs, the topic is filtered out of the subscription.

Use the select_topic permission with some care because topic selectors can use wild card expressions. For example, with the default configuration, the OPERATOR role is required to use topic selector expressions such as Diffusion or ?Diffusion//", but the CLIENT role is sufficient to use the topic selector expression ?// which selects all of the topics in the topic tree.

In the default configuration, this does not cause a problem as sessions that do not have the OPERATOR role also do not have the read_topic permission for topics below "Diffusion". Any matching topics are filtered from subscription and fetch results for those sessions.

Managing all subscriptions from a separate control session

You can prevent client sessions from subscribing themselves to topics and control all subscriptions from a separate control client session that uses SubscriptionControl feature to subscribe clients to topics.

To restrict subscription capability to control sessions, configure the following permissions:

Control session:

- Grant the modify_session permission
- Grant the select_topic permission

This can either be granted for the default topic scope or more selectively to restrict the topic selectors the control session can use.

Other sessions:

- Grant read_topic to the appropriate topics.
- Deny the select_topic permission by default.

Do not assign the session a role that has the select_topic permission for the default topic scope. This prevents the session from subscribing to all topics using a wildcard selector.

- Optionally, grant the select_topic permission to specific branches of the topic tree to which the session can subscribe freely.

Global permissions

The global permissions are listed in the following table:

Table 19: List of global permissions

Name	Description
view_session	List or listen to client sessions.
modify_session	Alter a client session. This covers a range of actions including the following:

Name	Description
	<ul style="list-style-type: none"> • subscribe a session to a topic • throttle a session • enable conflation for a session • close a session
register_handler	Register any handler with the Diffusion server.
authenticate	Register an authentication handler. The register_handler permission is also required to perform this action.
view_server	Read administrative information about the Diffusion server. For example, through JMX.
control_server	<ul style="list-style-type: none"> • Shut down the Diffusion server. • Start and stop publishers. <p>These actions can be taken only from the console or JMX. Client sessions cannot shut down the Diffusion server or start and stop publishers.</p>
view_security	View the security policy.
modify_security	Change the security policy.

Related reference

[Pre-defined roles](#) on page 138

Diffusion has a pre-defined set of roles with associated permissions.

[Pre-defined users](#) on page 146

Diffusion has a pre-defined set of users with associated password and roles.

Pre-defined roles

Diffusion has a pre-defined set of roles with associated permissions.

Clients can edit this set of roles. For more information, see [Updating the security store](#) on page 420.

	CLIENT	TOPIC_ CONTROL	CLIENT_ CONTROL	AUTHENTICATION _HANDLER	OPERATOR	ADMINISTRATOR and JMX_ ADMINISTRATOR
select topic Default scope	✓	✓	✓	✓	✓	✓
select topic "Diffusion" topic					✓	✓
read topic	✓	✓	✓	✓	✓	✓

	CLIENT	TOPIC_ CONTROL	CLIENT_ CONTROL	AUTHENTICATION _HANDLER	OPERATOR	ADMINISTRATOR and JMX_ ADMINISTRATOR
Default scope						
read topic "Diffusion" topic					✓	✓
modify topic Default scope		✓				✓
modify topic "Diffusion" topic						✓
update topic Default scope		✓				✓
update topic "Diffusion" topic						✓
send to message handler Default scope	✓	✓	✓	✓	✓	✓
send to message handler "Diffusion" topic					✓	✓
send to session Default scope		✓				✓
view session			✓		✓	✓
modify session			✓			✓

	CLIENT	TOPIC_ CONTROL	CLIENT_ CONTROL	AUTHENTICATION _HANDLER	OPERATOR	ADMINISTRATOR and JMX_ ADMINISTRATOR
register handler			✓	✓		✓
authenticate				✓		
view security						✓
modify security						✓
view server					✓	✓
control server						✓

Related reference

[Permissions](#) on page 134

The actions a client session can take in Diffusion are controlled by a set of permissions. These permissions are assigned to roles.

[Pre-defined users](#) on page 146

Diffusion has a pre-defined set of users with associated password and roles.

Authentication

You can implement and register handlers to authenticate clients when the clients try to perform operations that require authentication.

The handlers you can implement and register are the following:

- Any number of local authentication handlers
- Any number of control authentication handlers
- At most one authorization handler

Note: Using authorization handlers for authentication is deprecated.

The server calls the authentication handlers (local and control) in the order that they are defined in the `Server.xml` file. Then, if required, the server calls the authorization handler.

If no handlers are defined, the server allows the client operation by default.

Authentication process

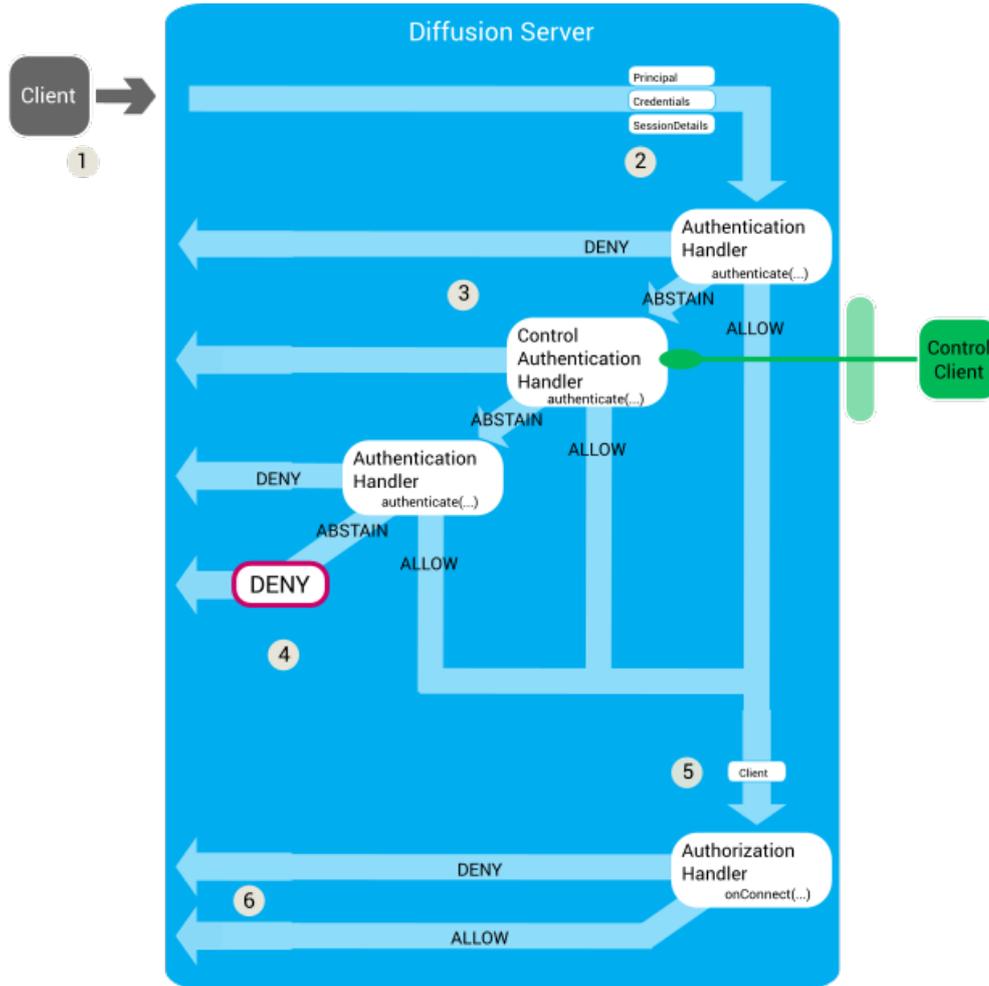


Figure 20: Authentication process for clients

1. A client tries to perform an operation that requires authentication. For more information, see [Client operations that require authentication](#) on page 142.
2. The server calls the authentication handlers one after another in the order that they are listed in the `Server.xml` file. It passes the following parameters to each authentication handler's `authenticate()` method:

Principal

A string that contains the name of the principal or identity that is connecting to the server or performing the action. This can have a value of `Session.ANONYMOUS`.

Credentials

The `Credentials` object contains an array of bytes that holds a piece of information that authenticates the principal. This can be empty or contain a password, a cryptographic key, an image, or any other piece of information. The authentication handler is responsible for interpreting the bytes.

SessionDetails

This contains information about the client. The available details depend on what information the server holds about the client session. Some session information might not be available on initial connection.

This information can be used in the authentication decision. For example, an authentication handler can allow connection only from clients that connect from a specific country.

When it registers with the server, a control authentication handler can specify what details it requires, so only these details are sent by the server (if available). This reduces the amount of data sent across the control client connection.

Callback

A callback that the authentication handler can use to respond to the authentication request by using the callback's `allow()`, `deny()`, or `abstain()` method.

If the authentication handler is a local authentication handler, the authentication logic is done on the server. If the authentication handler is a control authentication handler, the parameters are passed to a control client and the control client handles the authentication logic and returns a response.

3. Each authentication handler can return a response of `ALLOW`, `DENY`, or `ABSTAIN`.
 - If the authentication handler returns `DENY`, the client operation is rejected.
 - If the authentication handler returns `ALLOW`, the decision is passed to the authorization handlers. The authentication handler can also provide a list of roles to assign to the client session.
 - If the authentication handler returns `ABSTAIN`, the decision is passed to the next authentication handler listed in the `Server.xml` configuration file.
4. If all authentication handlers respond with an `ABSTAIN` decision, the response defaults to `DENY`.
5. If an authorization handler is configured in the `Server.xml` file, the server calls it. It passes the following parameter to the authorization handler's `onConnect()` method:

Client

The `Client` object has an associated `Credentials` object. This `Credentials` object is part of the Classic API and is different to the `Credentials` object used by the authentication handlers. The Classic API `Credentials` object contains two strings: username and password. The username string is equivalent to the Principal string used by the Unified API.

6. The authorization handler can return a response of `ALLOW` or `DENY`.
 - If the authorization handler returns `DENY`, the client operation is rejected.
 - If the authorization handler returns `ALLOW`, the client operation is allowed.

Client operations that require authentication

The following client operations require authentication with the server:

Table 20: Client operations that require authentication

Client operation	API version	Behavior if operation is allowed	Behavior if operation is denied
Connect to server	Unified API	The client connection to the server is accepted.	The client connection to the server is rejected and is dropped.
Connect to server	Classic API	The client connection to the server is accepted.	The client connection to the server is rejected and is dropped.

Client operation	API version	Behavior if operation is allowed	Behavior if operation is denied
Change the principal associated with a client session	Unified API	The principal is changed.	The principal is not changed, but the client session is not dropped.
Change the principal associated with a client session	Classic API	The principal is changed. In the Classic API, the principal is the username string inside the <code>Credentials</code> object.	The principal is not changed, but the client session is not dropped.

Related concepts

[Role-based authorization](#) on page 130

Diffusion restricts the ability to perform actions to authorized principals. Roles are used to map permissions to principals.

DEPRECATED: [Authorization handlers](#) on page 147

An authorization handler can control authorization and permissions for clients and users.

[Securing the console](#) on page 150

Configuration is required to enable additional security around connections from the Diffusion console.

User-written authentication handlers

You can implement authentication handlers that authenticate clients that connect to the Diffusion server or perform an action that requires authentication.

The authentication handlers can be implemented either remotely, in a client, or locally, on the server. The authentication handlers can be individual authentication handlers, that perform a single authentication check, or composite authentication handlers, that delegate to one or more individual authentication handlers.

Local authentication handlers

A local authentication handler is an implementation of the `AuthenticationHandler` interface. Local authentication handlers can be implemented only in Java. The class file that contains a local authentication handler must be located on the classpath of the Diffusion server.

For more information, see [Authentication API](#).

Control authentication handlers

A control authentication handler can be implemented in any language where the Diffusion Unified API includes the `AuthenticationControl` feature. A control authentication handler can be registered by any client that has the `authenticate` and `register_handler` permissions.

For more information, see [Authenticating clients](#) on page 395.

Composite authentication handlers

A composite authentication handler delegates the authentication decision to an ordered list of one or more individual authentication handlers and returns a combined decision.

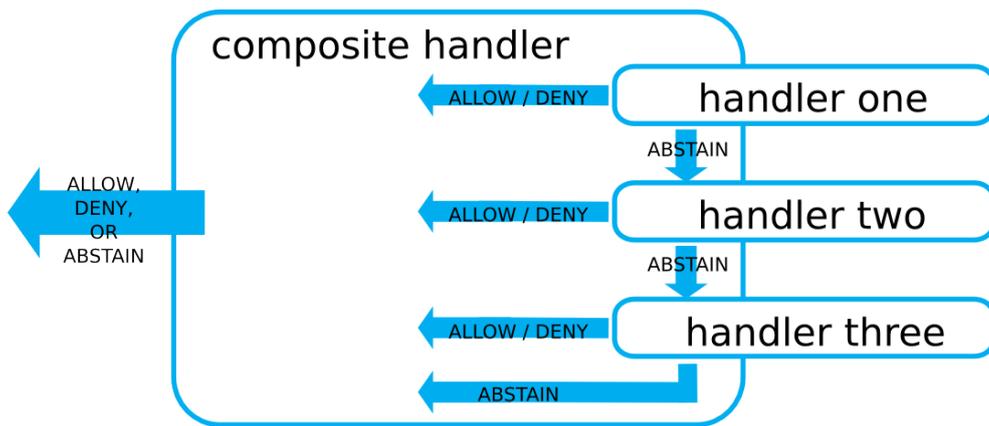


Figure 21: A composite authentication handler

- If an individual handler allows the client action, the composite handler responds with an ALLOW decision.
- If an individual handler denies the client action, the composite handler responds with a DENY decision.
- If an individual authentication handler abstains, the composite handler calls the next individual handler.
- If all individual handlers abstain, the composite handler responds with an ABSTAIN decision.

A composite authentication handler can be either local or control. A local composite authentication handler can delegate the authentication decision to one or more authentication handlers. A composite control authentication handler can delegate the authentication decision to one or more control authentication handlers.

The use of composite authentication handlers is optional. There are two reasons to consider using them:

- Composite authentication handlers enable you to combine authentication handlers together, which reduces the possibility of misconfiguration.
- Composite control authentication handlers improve efficiency by reducing the number of messages sent between the Diffusion server and clients.

The following table matrix shows the four types of authentication handler.

Table 21: Types of authentication handler

	Individual	Composite
Local	Implement the <code>AuthenticationHandler</code> interface. For more information, see Developing a local authentication handler on page 508.	Extend the <code>CompositeAuthenticationHandler</code> class. For more information, see Developing a composite authentication handler on page 510
Control	Implement the <code>ControlAuthenticationHandler</code> interface. For more information, see Developing a control authentication handler on page 402.	Extend the <code>CompositeControlAuthenticationHandler</code> class. For more information, see Developing a composite control authentication handler on page 405

Related concepts

[Configuring authentication handlers](#) on page 561

Authentication handlers and the order that the Diffusion server calls them in are configured in the `Server.xml` configuration file.

Related tasks

[Developing a local authentication handler](#) on page 508

Implement the `AuthenticationHandler` interface to create a local authentication handler.

[Developing a composite authentication handler](#) on page 510

Extend the `CompositeAuthenticationHandler` class to combine the decisions from multiple authentication handlers.

[Developing a control authentication handler](#) on page 402

Implement the `ControlAuthenticationHandler` interface to create a control authentication handler.

[Developing a composite control authentication handler](#) on page 405

Extend the `CompositeControlAuthenticationHandler` class to combine the decisions from multiple control authentication handlers.

[Developing a local authentication handler](#) on page 508

Implement the `AuthenticationHandler` interface to create a local authentication handler.

[Developing a composite authentication handler](#) on page 510

Extend the `CompositeAuthenticationHandler` class to combine the decisions from multiple authentication handlers.

Related reference

[Authentication API](#)

[Server.xml](#) on page 563

This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

System authentication handler

Diffusion provides an authentication handler that uses principal, credential, and roles information stored in the Diffusion server to make its authentication decision.

System authentication store

The principal, credentials, and role information located in the system authentication store is used by the system authentication handler to authenticate users.

The system authentication store is designed to hold information about Diffusion administration users and system clients. It can manage hundreds or perhaps thousands of principals, but does not provide the administration tools necessary to support millions of principals. We recommend that you delegate such "internet scale" use cases to a third-party identity provider using a custom authentication handler. For example, by using the OAuth or OpenID protocol.

By default the following information is set in the system authentication store file, `SystemAuthentication.store` located in the `etc` directory:

```
allow anonymous connections [ "CLIENT" ]

add principal "client" "password" [ "CLIENT" ]
add principal "control" "password" [ "CLIENT_CONTROL" "TOPIC_CONTROL"
  "AUTHENTICATION_HANDLER" ]
add principal "admin" "password" [ "ADMINISTRATOR" ]
add principal "operator" "password" [ "OPERATOR" ]
```

You can edit the usernames and passwords in this file by hand and restart the Diffusion server to reload the file. However, any password you enter in plaintext is hashed by the Diffusion server when it starts and the plaintext value in this file is replaced with the hashed value.

The default hash scheme used is PBKDF-SHA256-1000. You can specify a different hash scheme in the `Server.xml` configuration file. For more information, see [Server.xml](#) on page 563.

Behavior of the system authentication handler

The system authentication handler behaves in the following way:

- If anonymous connections are allowed in the system authentication store and a client session connects anonymously, the system authentication handler returns an ALLOW decision and the list of roles an anonymous client session is assigned.
- If anonymous connections are not allowed in the system authentication store and a client session connects anonymously, the system authentication handler returns a DENY decision.
- If a client session connects with a principal listed in the system authentication store and the correct credentials, the system authentication handler returns an ALLOW decision and the list of roles that client session is assigned.
- If a client session connects with a principal listed in the system authentication store and incorrect credentials, the system authentication handler returns a DENY decision.
- If a client session connects with a principal that is not listed in the system authentication store, the system authentication handler returns an ABSTAIN decision.

Related concepts

[Configuring authentication handlers](#) on page 561

Authentication handlers and the order that the Diffusion server calls them in are configured in the `Server.xml` configuration file.

[Updating the system authentication store](#) on page 407

A client can use the `SystemAuthenticationControl` feature to update the system authentication store. The information in the system authentication store is used by the system authentication handler to authenticate users and assign roles to them.

Pre-defined users

Diffusion has a pre-defined set of users with associated password and roles.

You can use the `SystemAuthenticationControl` feature to edit this set of users.

Note: This set of users and passwords are well known and not secure. Change the passwords or remove the users before putting Diffusion into production.

The users defined in the system authentication store are only authenticated if the system authentication handler is configured. For more information, see [Configuring authentication handlers](#) on page 561.

User	Password	Associated roles
client	password	CLIENT
control	password	CLIENT_CONTROL, TOPIC_CONTROL, AUTHENTICATION_HANDLER
admin	password	ADMINISTRATOR
operator	password	OPERATOR

User	Password	Associated roles
Anonymous connections		CLIENT

Related reference

[Pre-defined roles](#) on page 138

Diffusion has a pre-defined set of roles with associated permissions.

[Permissions](#) on page 134

The actions a client session can take in Diffusion are controlled by a set of permissions. These permissions are assigned to roles.

DEPRECATED: Authorization handlers

An authorization handler can control authorization and permissions for clients and users.

Role-based authorization

Attention: The new role-based security model has superseded authorization handlers. Role-based security enables you to more simply manage permissions and users. We recommend you use role-based authorization instead of authorization handlers. For more information, see [Role-based authorization](#) on page 130.

An authorization handler is a user-written Java class that must implement the `AuthorisationHandler` interface in the Classic API.

Such a handler can be used to restrict access of clients according to any criteria that is appropriate. One capability within Diffusion is for a client to be able to specify Credentials when they connect that can be checked by the authorization handler.

The handler can either be specified in `etc/Server.xml` in which case it is loaded when the server starts or can be set programmatically within a publisher using the `Publishers.setAuthorisationHandler` method.

There can only be one handler and it is system wide across all publishers, although you can have authorization at the publisher level.

If an authorization handler is not specified, credentials sent by a client are assumed to be valid. A publisher has access to the credentials to perform finer-grained authorization, if required.

The authorization handler interface has the following methods:

Table 22: Authorization handler methods

DEPRECATED: <code>canConnect(Client)</code>	This method is called to establish whether the client can connect and is called before any client validation policy is called.
<code>canSubscribe(Client, Topic)</code>	This method is called when a client subscribes to a topic. If topic information is sent with the connection, this method is called after the <code>canConnect</code> method. Note: This is called for every topic being subscribed to, even if subscribed as a result of a topic selector being specified. However (by default), if a topic

	is rejected by this method, it is not called again for any children (or descendants) of the topic.
<code>canSubscribe(Client, TopicSelector)</code>	This method is called when a client attempts to subscribe to a topic selector pattern (as opposed to a simple topic name). If topic information is sent with the connection, this method is called after the <code>canConnect</code> method.
<code>canFetch(Client, Topic)</code>	This method is called when a client sends a fetch request to obtain the current state of a topic. Note: This is called for every topic being fetched, even if fetched as a result of a topic selector being specified. However (by default), if a topic is rejected by this method, it is not be called again for any children (or descendants) of the topic.
<code>canFetch(Client, TopicSelector)</code>	This method is called when a client attempts to fetch topics using a topic selector pattern (as opposed to a simple topic name).
<code>canWrite(Client, Topic)</code>	This method is called when a client sends a message on a given topic, if false is returned the message is ignored, and the publisher will not be notified of the message. When implementing this method, be aware that performance can be impacted if many clients send messages or if a few clients send large messages.
DEPRECATED: <code>credentialsSupplied(Client, Credentials)</code>	This method is called when a client submits credentials after connection. It can be used to validate the credentials and must return true if the credentials are OK. If this returns false, a <code>Credentials Rejected</code> message are sent back to the client.

Authentication

Note: The use of authorization handlers for authentication is deprecated. We recommend that you re-implement your authentication logic using authentication handlers. For more information, see [User-written authentication handlers](#) on page 143.

When a client connects to Diffusion it has the option of supplying user credentials. These credentials are basically tokens called username and password. These tokens can be used for any purpose. When `canConnect` is called, you can get the credentials from the `Client` object.

An example of this is:

```
public boolean canConnect(Client client) {
    Credentials creds = client.getCredentials();

    // No creds supplied, so reject the connection
    if (creds == null) {
        return false;
    }
}
```

```
String username = creds.getUsername().toLowerCase();
```

If the credentials are null, none were supplied, which is different from empty credentials. If you set the username as an empty string (that is, an anonymous user) the password is not stored and you cannot retrieve it with `getCredentials`.

Clients can connect without credentials and submit them later or replace the credentials at any time whilst connected. The authorization handler is notified when new credentials are submitted and can choose to set the new credentials on the client.

The `Credentials` class has username and password attributes, but also allows for an attachment. It is here that a user normally sets any security object required. Returning true will allow the user to connect, returning false will result in the client connection being refused.

Subscription authorization

Subscription authorization is the allowing of a client to subscribe to a topic. In this case the `canSubscribe` is called. Returning true here allows the publisher to have any topic loaders and subscription methods called. Returning false will not notify the client that the subscription was invalid.

```
public boolean canSubscribe(Client client, Topic topic) {  
    // Everyone has access to the top level topic  
    if (topic.getName().equals(CHAT_TOPIC)) {  
        return true;  
    }  
  
    User user = (User) client.getCredentials().attachment();  
  
    return user.isAllowedRoom(topic.getNodeName());  
}
```

Authorization handler

Authorization at the publisher level can also be achieved. This is required if there are many publishers running within the same Diffusion Server and they have different security settings. The following code example works if the publishers all implement `AuthorisationHandler`

```
public boolean canSubscribe(Client client, Topic topic) {  
    AuthorisationHandler handler =  
    (AuthorisationHandler) Publishers.getPublisherForTopic(topic);  
  
    // Call the publisher in question  
    return handler.canSubscribe(client, topic);  
}
```

Permissions

The permissions process governs whether a client is able to send messages to a publisher, or in other words, is the topic read only. This is handled by the `canWrite` method. Again a good pattern might be to look at the credentials attachment object to see if this is permissible.

```
public boolean canWrite(Client client, Topic topic) {  
    User user = (User) client.getClientCredentials().attachment();  
    return user.canWriteMessages(topic);  
}
```

Related concepts

[Role-based authorization](#) on page 130

Diffusion restricts the ability to perform actions to authorized principals. Roles are used to map permissions to principals.

[Authentication](#) on page 140

You can implement and register handlers to authenticate clients when the clients try to perform operations that require authentication.

[Securing the console](#) on page 150

Configuration is required to enable additional security around connections from the Diffusion console.

Securing the console

Configuration is required to enable additional security around connections from the Diffusion console.

Allow the console to connect only on a specific connector

We strongly recommend that you only allow the console to connect to Diffusion through a single connector. The port this connector listens on can be blocked from connections from outside of your organization by your load balancer.

You can configure this in the following way:

1. In your `etc/Connectors.xml` configuration file, wherever the line `<web-server>default</web-server>` appears in a connector that receives external connections, replace it with a web server definition that contains only a `client-service` definition. For example:

```
<web-server name="external">
  <!-- This section enables HTTP-type clients for this Web
  Server -->
  <client-service name="client" debug="true">
    <!-- This parameter is used to re-order out-of-order
    messages received
    over separate HTTP connections opened by client
    browsers. It is rarely
    necessary to set this to more than a few tens of
    seconds.
    If you attempt to set this value to more than one
    hour, a warning is logged
    and a timeout of one hour is used. -->
    <message-sequence-timeout>4s</message-sequence-timeout>
    <!-- This is used to control access from client web
    socket to diffusion.
    This is a REGEX pattern that will match the origin
    of the request (.*) matches
    anything so all requests are allowed -->
    <websocket-origin>.*</websocket-origin>
    <!-- This is used to control cross-origin resource
    sharing client
    connection to Diffusion
    This is a REGEX pattern that will match the origin
    of the request (.*) matches anything -->
    <cors-origin>.*</cors-origin>
    <!-- Enable compression for HTTP responses (Client and
    File). If the response
    is bigger than threshold -->
    <compression-threshold>256</compression-threshold>
  </client-service>
```

```
</web-server>
```

2. Create a new connector in your `etc/Connectors.xml` configuration file that defines a specific port that you use for internal connections to the console.

In this connector, set the value of the `web-server` element to default.

3. In your load balancer, prevent outside traffic from having access to the port specified in the new connector.
4. If required, apply additional connection restrictions.
 - You can use a connection validation policy. For more information, see [ConnectionValidationPolicy.xml](#) on page 624.
 - You can set these restrictions in your load balancer.

Disable console features in the configuration (as required)

The actions that a user can perform using the console are controlled by roles and permissions. The principal that the user uses to log in to the console must have a role with the permissions required to perform an action in the console.

A principal with the ADMINISTRATOR or OPERATOR role can use all of the functions of the Diffusion console.

To restrict users to using a smaller set of console features, ensure they use a principal with a more restrictive set of roles and permissions. For more information, see [Pre-defined roles](#) on page 138.

Related concepts

[Role-based authorization](#) on page 130

Diffusion restricts the ability to perform actions to authorized principals. Roles are used to map permissions to principals.

[Authentication](#) on page 140

You can implement and register handlers to authenticate clients when the clients try to perform operations that require authentication.

DEPRECATED: [Authorization handlers](#) on page 147

An authorization handler can control authorization and permissions for clients and users.

Part IV

Developer Guide

This guide describes how to develop clients, publishers, and server-side components that interact with the Diffusion server.

Note: We recommend that you use develop clients for most use cases. Our client APIs provide access to the majority of Diffusion capabilities. Publishers and server-side components provide a few advanced features that are not available on clients.

In this section:

- [The Diffusion SDKs](#)
- [Feature support in the Diffusion Unified API](#)
- [Best practice for developing clients](#)
- [Getting started](#)
- [Connecting to the Diffusion server](#)
- [Receiving data from topics](#)
- [Managing topics](#)
- [Updating topics](#)
- [Managing subscriptions](#)
- [Messaging to topic paths](#)
- [Messaging to sessions](#)
- [Authenticating clients](#)
- [Updating the system authentication store](#)
- [Updating the security store](#)
- [Managing clients](#)
- [Logging from the client](#)
- [DEPRECATED: Classic API](#)
- [Developing a publisher](#)
- [Developing other components](#)
- [Using Maven to build Java Diffusion applications](#)
- [Testing](#)

The Diffusion SDKs

Push Technology provides client SDKs for JavaScript, Apple, Android, Java, .NET, and C platforms. Use our SDKs to develop client applications that can connect to Diffusion and Diffusion Cloud.

JavaScript

The JavaScript Unified API is provided in the file `diffusion.js` and can be accessed through the web or through NPM.

Use with Node.js:

Install with NPM:

```
npm install --save diffusion
```

Include in your Node.js application:

```
const diffusion = require('diffusion');
```

Get the minified JavaScript:

Download the latest JavaScript file from the following URL:

```
http://download.pushtechnology.com/clients/5.9.24/js/diffusion.js
```

You can also download the JavaScript file as a tarball that can be installed locally by using NPM:

```
http://download.pushtechnology.com/clients/5.9.24/js/diffusion-js-5.9.24.tgz
```

The JavaScript file is also located in your Diffusion server installation:

```
diffusion_directory/clients/js
```

Use TypeScript definitions with the JavaScript client library:

If you got the JavaScript client library using NPM, the TypeScript definitions are included.

You can also download a TypeScript definition file from the following URL:

```
http://download.pushtechnology.com/clients/5.9.24/js/diffusion-5.9.24.d.ts
```

The TypeScript file is also located in your Diffusion server installation:

```
diffusion_directory/clients/js
```

Include the TypeScript definition file in your IDE project to use the TypeScript definitions when developing a JavaScript client for Diffusion.

Capabilities

To see the full list of capabilities supported by the JavaScript API, see [Feature support in the Diffusion Unified API](#) on page 49.

Support

Table 23: Supported platforms and transport protocols for the client libraries

Platform	Minimum supported versions	Supported transport protocols
JavaScript	es6 (TypeScript 1.8)	WebSocket HTTP (Polling XHR)

Resources

- [Examples for the JavaScript API.](#)
- [JavaScript Classic API documentation](#)

Using

Promises

The Diffusion JavaScript Unified API uses the [Promises/A+](#) specification.

Views

The JavaScript Unified API provides a view capability.

Use views to subscribe to multiple topics by using a topic selector and receive all the data from all topics in the selector set as a single structure when any of the topics are updated. If the topic selector matches a topic which is subsequently added or removed, the view is updated.

The following example shows views being used to present data from multiple topics as a single structure:

```
diffusion.connect({
  host      : 'localhost',
  port     : 8080,
  secure    : false,
  principal : 'control',
  credentials : 'password'
}).then(function(session) {

  // Assuming a topic tree:
  //
  // scores
  //   |-- football
  //   |   |-- semi1
  //   |   |-- semi2
  //   |   |-- final
  //   |-- tennis
  //       |-- semi1
  //       |-- semi2
  //       |-- final
```

```

// Use a regular expression to create a view of the
topics tracking the
// scores during the finals for each sport.
var view = session.view('?scores/.*/final');

// Alternatively, we can use a topic set. Note that
the topics do not need
// to be under a common root, they may be anywhere
within the topic tree.
var view2 = session.view('#>scores/football/final////
>scores/tennis/final');

// If any of the topics in the view change, display
which topic changed
// and its new value.
view.on({
  update : function(value) {
    // Get and print the entire view structure.
    console.log('Update: ', JSON.stringify(value,
undefined, 4));

    // Get individual topics. Returns a Buffer,
which is automatically
    // converted to a String during
concatenation, below.
    //
    // Note that the structure may not exist if
the value has not been
    // updated.
    console.log('Football score: ' +
value.scores.football.final);
    console.log('Tennis score : ' +
value.scores.tennis.final);

    // or ...
    // console.log('Football score: ' +
value['scores']['football']['final']);
  }
});

// The structure can also be accessed outside the
update event.
console.log('Football score: ' +
view.get().scores.football.final);
});

```

Regular expressions

The JavaScript client uses a different regular expression engine to the Diffusion server. Some regular expressions in topic selectors are evaluated on the client and others on the Diffusion server. It is possible that topic selectors that include complex or advanced regular expressions can behave differently on the client and on the Diffusion server.

For more information, see [Regular expressions](#) on page 67.

Apple

The Apple SDK is provided for iOS, OS X/macOS, and tvOS. It can be used with Xcode 7 and later.

Get the Apple SDK for iOS:

Download the SDK from the following URL:

```
http://download.pushtechnology.com/clients/5.9.24/apple/diffusion-iphoneos-5.9.24.zip
```

The SDK file is also located in your Diffusion server installation:

```
diffusion_directory/clients/apple/diffusion-iphoneos-5.9.24.zip
```

Get the Apple SDK for OS X/macOS:

Download the SDK from the following URL:

```
http://download.pushtechnology.com/clients/5.9.24/apple/diffusion-macosx-5.9.24.zip
```

The SDK file is also located in your Diffusion server installation:

```
diffusion_directory/clients/apple/diffusion-macosx-5.9.24.zip
```

Get the Apple SDK for tvOS:

Download the SDK from the following URL:

```
http://download.pushtechnology.com/clients/5.9.24/apple/diffusion-appletvos-5.9.24.zip
```

The SDK file is also located in your Diffusion server installation:

```
diffusion_directory/clients/apple/diffusion-appletvos-5.9.24.zip
```

Capabilities

To see the full list of capabilities supported by the Apple API, see [Feature support in the Diffusion Unified API](#) on page 49.

Support

Table 24: Supported platforms and transport protocols for the client libraries

Platform	Minimum supported versions	Supported transport protocols
Apple for iOS	Development environment Xcode 7 (iOS 9.0 SDK) Runtime support	WebSocket

Platform	Minimum supported versions	Supported transport protocols
	<p>Deployment target: iOS 7.0 or later</p> <p>Device architectures: armv7, armv7s, arm64</p> <p>Simulator architectures: i386, x86_64</p>	
Apple for OS X/macOS	<p>Development environment</p> <p>Xcode 7 (OS X 10.11 SDK)</p> <p>Runtime support</p> <p>Deployment target: OS X 10.9 or later</p> <p>Device architectures: x86_64</p>	WebSocket
Apple for tvOS	<p>Development environment</p> <p>Xcode 7 (tvOS 9.0 SDK)</p> <p>Runtime support</p> <p>Deployment target: tvOS 9.0 or later</p> <p>Device architectures: arm64</p> <p>Simulator architectures: x86_64</p>	WebSocket

Resources

- [Objective-C examples for the Apple API.](#)
- [Apple Unified API documentation](#)

Using

Applications in background state

Apple applications can be sent to the background. When this happens your application is notified by the `applicationDidEnterBackground` callback. Applications go into background state before being suspended.

Applications can be sent to the background or suspended at any time. We recommend that your Diffusion app saves its state – in particular, any topic subscriptions – as this state changes.

When your Diffusion app is sent to the background, we recommend the client closes its session with the Diffusion server. When the Diffusion app returns to the foreground, it can open a new client session with the Diffusion server and use the saved state to restore topic subscriptions.

For more information, see [the Apple App Life Cycle documentation](#) and [Strategies for Handling App State Transitions](#).

Consider using push notifications to deliver data to your users when your client application is in background state.

Regular expressions

The Apple client uses a different regular expression engine to the Diffusion server. Some regular expressions in topic selectors are evaluated on the client and others on the Diffusion server. It is possible that topic selectors that include complex or advanced regular expressions can behave differently on the client and on the Diffusion server.

For more information, see [Regular expressions](#) on page 67.

Android

The Android API is bundled in a JAR file and is supported on Android KitKat and later.

Get the Android SDK as a JAR:

Download the JAR from the following URL:

```
http://download.pushtechnology.com/clients/5.9.24/android/diffusion-android-5.9.24.jar
```

The JAR file is also located in your Diffusion server installation:

```
diffusion_directory/clients/android/diffusion-android-5.9.24.jar
```

Capabilities

To see the full list of capabilities supported by the Android API, see [Feature support in the Diffusion Unified API](#) on page 49.

Support

Table 25: Supported platforms and transport protocols for the client libraries

Platform	Minimum supported versions	Supported transport protocols
Android	API 19 / v4.4 / KitKat Note: Push Technology provides only best-effort support for Jelly Bean (API 16-18, v4.1-4.3).	WebSocket HTTP (polling) DEPRECATED: DPT DEPRECATED: HTTP (Full duplex)

Resources

- [Java examples for the Android API.](#)
- [Android Unified API documentation](#)

Using

Considerations and capabilities that are specific to the Android Unified API

Diffusion connections

Ensure that you use the asynchronous `open()` method with a callback. Using the synchronous `open()` method might open a connection on the same thread as the UI and cause a runtime exception. However, the synchronous `open()` method can be used in any thread that is not the UI thread.

Applications in background state

Android applications can be sent to the background and their activity stopped. When this happens your application is notified by the `onStop()` callback of the Android `Activity` class. An application's activity can be stopped when the user switches to another application, starts a new activity from within the application, or receives a phone call.

When your application's activity is stopped, we recommend that it saves its state locally – in particular, any topic subscriptions it has made – and closes its client session with the Diffusion server. When the Diffusion app returns to the foreground, open a new client session with the Diffusion server and use the saved state to restore topic subscriptions.

For more information, see [the Android Activity Lifecycle documentation](#) and [Stopping and Restarting an Activity](#).

Consider using push notifications to deliver data to your users when your client application is in background state. For more information, see [Push notification networks](#) on page 123.

Writing good callbacks

The Android client library invokes callbacks using a thread from Diffusion thread pool. Callbacks for a particular session are called in order, one at a time. Consider the following when writing callbacks:

- Do not sleep or call blocking operations in a callback. If you do so, other pending callbacks for the session are delayed. If you must call a blocking operation, schedule it in a separate application thread.

- You can use the full Diffusion API to make other requests to the server. If you want to make many requests based on a single callback notification, be aware that Diffusion client flow control is managed differently in callback threads. Less throttling is applied and it is easier to overflow the servers by issuing many thousands of requests. If you have a lot of requests to make, it is better to schedule the work in an application thread.

Regular expressions

The Android client uses the same regular expression engine to the Diffusion server. Some regular expressions in topic selectors are evaluated on the client and others on the Diffusion server. There is no difference in how these regular expressions are evaluated in the Android client.

Java

The Java Unified API is provided as a JAR file for Java 7 and later.

Get the Java client libraries using Maven™:

Add the Push Technology public repository to your pom.xml file:

```
<repositories>
  <repository>
    <id>push-repository</id>
    <url>https://download.pushtechnology.com/maven/</url>
  </repository>
</repositories>
```

Declare the following dependency in your pom.xml file:

```
<dependencies>
  <dependency>
    <groupId>com.pushtechnology.diffusion</groupId>
    <artifactId>diffusion-client</artifactId>
    <version>version</version>
  </dependency>
</dependencies>
```

Get the Java client libraries using Gradle:

Add the Push Technology public repository to your build.gradle file:

```
repositories { maven { url "http://download.pushtechnology.com/
maven/" } }
```

Declare the following dependency in your build.gradle file:

```
compile 'com.pushtechnology.diffusion:diffusion-client:5.9.24'
```

Get the Java client libraries:

Download the JAR file from the following URL:

```
http://download.pushtechnology.com/clients/5.9.24/java/diffusion-client.jar
```

The ZIP file is also located in your Diffusion server installation:

```
diffusion_directory/clients/java/diffusion-client-5.9.24.jar
```

Capabilities

To see the full list of capabilities supported by the Java API, see [Feature support in the Diffusion Unified API](#) on page 49.

Support

Table 26: Supported platforms and transport protocols for the client libraries

Platform	Minimum supported versions	Supported transport protocols
Java	8 (recommended), 7 (supported) Note: We recommend that you run your clients on the JDK rather than the JRE.	WebSocket HTTP (Polling) DEPRECATED: DPT DEPRECATED: HTTP (Full duplex)

Resources

- [Examples for the Java API.](#)
- [Java Unified API documentation](#)

Using

Certificates

Diffusion Java clients use certificates to validate the security of their connection to the Diffusion server. The client validates the certificate sent by the Diffusion server against the set of certificates trusted by the .

If the certificate sent by the Diffusion server cannot be validated against any certificates in the set trusted by the JDK, you receive an exception that contains the following message:

```
sun.security.provider.certpath.SunCertPathBuilderException:  
unable to find valid certification path to requested target.
```

Diffusion is authenticated using the certificates provided by your certificate authority for the domain you host the Diffusion server on.

To ensure that the certificate is validated, set up a trust store for the client and add the appropriate certificates to that trust store:

1. Obtain the appropriate intermediate certificate from the certificate authority.
2. Use keytool to create a trust store for your client that includes this certificate.

For more information, see <https://docs.oracle.com/cd/E19509-01/820-3503/ggfka/index.html>

3. Use system properties to add the trust store to your client.

For example:

```
System.setProperty("javax.net.ssl.trustStore",  
"truststore_name");
```

Or at the command line:

```
-Djavax.net.ssl.keyStore=path_to_truststore
```

Writing good callbacks

The Java client library invokes callbacks using a thread from Diffusion thread pool. Callbacks for a particular session are called in order, one at a time. Consider the following when writing callbacks:

- Do not sleep or call blocking operations in a callback. If you do so, other pending callbacks for the session are delayed. If you must call a blocking operation, schedule it in a separate application thread.
- You can use the full Diffusion API to make other requests to the server. If you want to make many requests based on a single callback notification, be aware that Diffusion client flow control is managed differently in callback threads. Less throttling is applied and it is easier to overflow the servers by issuing many thousands of requests. If you have a lot of requests to make, it is better to schedule the work in an application thread.

Regular expressions

The Java client uses the same regular expression engine to the Diffusion server. Some regular expressions in topic selectors are evaluated on the client and others on the Diffusion server. There is no difference in how these regular expressions are evaluated in the Java client.

.NET

The .NET Unified API is provided as a DLL file compatible with .NET Framework 4.5 and above.

Get the .NET SDK from NuGet:

Use the following command in the NuGet Package Manager Console:

```
PM> Install-Package PushTechnology.UnifiedClientInterface
```

Get the .NET SDK:

Download the DLL file from the following URL:

```
http://download.pushtechnology.com/clients/5.9.24/dotnet/  
PushTechnology.ClientInterface.dll
```

The .NET SDK also requires NLog:

```
http://download.pushtechnology.com/clients/5.9.24/dotnet/NLog.dll
```

You can download these XML files to get IntelliSense documentation:

```
http://download.pushtechnology.com/clients/5.9.24/dotnet/  
PushTechnology.ClientInterface.XML
```

```
http://download.pushtechnology.com/clients/5.9.24/dotnet/NLog.xml
```

These files are also located in your Diffusion server installation:

```
diffusion_directory/clients/dotnet
```

Capabilities

To see the full list of capabilities supported by the .NET API, see [Feature support in the Diffusion Unified API](#) on page 49.

Support

Table 27: Supported platforms and transport protocols for the client libraries

Platform	Minimum supported versions	Supported transport protocols
.NET	4.5	WebSocket DEPRECATED: DPT DEPRECATED: HTTP (Full duplex)

Resources

- [Examples for the .NET API.](#)
- [.NET Unified API documentation](#)

Using

Certificates

Diffusion .NET clients use certificates to validate the security of their connection to the Diffusion server. The client validates the certificate sent by the Diffusion server against the set of certificates trusted by the .NET Framework.

If the certificate sent by the Diffusion server cannot be validated against any certificates in the set trusted by the .NET Framework, you must set up a trust store for the client and add the appropriate certificates to that trust store.

Diffusion is authenticated using the certificates provided by your certificate authority for the domain you host the Diffusion server on.

1. Obtain the appropriate intermediate certificate from the certificate authority.
2. Use the **Microsoft Management Console** to import the certificate into the **Trusted Root Certification Authorities** folder. For more information, see [https://msdn.microsoft.com/en-us/library/aa738659\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/aa738659(v=vs.110).aspx)

Writing good callbacks

The .NET client library invokes callbacks using a single inbound thread. Callbacks for a particular session are called in order, one at a time. Consider the following when writing callbacks:

- Do not sleep or call blocking operations in a callback. If you do so, other pending callbacks for the session are delayed. If you must call a blocking operation, schedule it in a separate application thread.
- You can use the full Diffusion API to make other requests to the server. If you want to make many requests based on a single callback notification, be aware that Diffusion client flow control is managed differently in callback threads. Less throttling is applied and it is easier to overflow the servers by issuing many thousands of requests. If you have a lot of requests to make, it is better to schedule the work in an application thread.

Regular expressions

The .NET client uses a different regular expression engine to the Diffusion server. Some regular expressions in topic selectors are evaluated on the client and others on the Diffusion server. It is possible that topic selectors that include complex or advanced regular expressions can behave differently on the client and on the Diffusion server.

For more information, see [Regular expressions](#) on page 67.

C

The C client libraries are provided for Linux, Windows, and OS X/macOS.

Get the C client libraries for Linux:

Download the ZIP file from the following URL:

```
http://download.pushtechnology.com/clients/5.9.24/c/diffusion-c-5.9.24.zip
```

The ZIP file is also located in your Diffusion server installation:

```
diffusion_directory/clients/c/diffusion-c-5.9.24.zip
```

Get the C client libraries for Windows:

Download the ZIP file from the following URL:

```
http://download.pushtechnology.com/clients/5.9.24/c/diffusion-c-windows-5.9.24.zip
```

The ZIP file is also located in your Diffusion server installation:

```
diffusion_directory/clients/c/diffusion-c-windows-5.9.24.zip
```

Get the C client libraries for OS X/macOS:

Download the ZIP file from the following URL:

```
http://download.pushtechnology.com/clients/5.9.24/c/diffusion-c-osx-5.9.24.zip
```

The ZIP file is also located in your Diffusion server installation:

```
diffusion_directory/clients/c/diffusion-c-osx-5.9.24.zip
```

Capabilities

To see the full list of capabilities supported by the C API, see [Feature support in the Diffusion Unified API](#) on page 49.

Support

Table 28: Supported platforms and transport protocols for the client libraries

Platform	Minimum supported versions	Supported transport protocols
C for Linux	Red Hat and CentOS, version 7.2 and later Ensure that you use a C99-capable compiler.	WebSocket DEPRECATED: DPT
C for Windows	Visual C Compiler 2013 or later, Windows 7 or later	WebSocket DEPRECATED: DPT
C for OS X/macOS	For building using GCC, use Xcode 7.1 or later	WebSocket DEPRECATED: DPT

If you require libraries compiled on a different platform, this can be provided as an additional service by our Consulting Services team. Contact support@pushtechnology.com to discuss your requirements.

Resources

- [Examples for the C API.](#)
- [C Unified API documentation](#)

Using

On Linux

The C libraries are provided compiled for 64-bit Linux in the file `diffusion-c-version.zip`. A dynamic library, `libdiffusion.so`, and a static library, `libdiffusion.a`, are available. These libraries are supported on Red Hat and CentOS version 6.5 and later.

To use the C Unified API on Linux ensure that the following dependencies are available on your development system:

- Perl Compatible Regular Expressions (PCRE) library, version 8.3 or later
For more information, see <http://pcre.org>
- OpenSSL library, version 1.0.2a or later
For more information, see <https://www.openssl.org>

You can download these dependencies through your operating system's package manager.

The C client library statically links to APR version 1.5 with APR-util. Ensure that you set `APR_DECLARE_STATIC` and `APU_DECLARE_STATIC` before you use any APR includes. You can set these values in the following ways:

- By including `diffusion.h` before any APR includes. The `diffusion.h` file sets these values.
- As command-line flags

For more information, see <http://apr.apache.org>

On Windows

The C library is provided as a static library compiled for 32-bit and 64-bit Windows in the file `diffusion-c-windows-version.zip`. This static library, `uci.lib`, is compiled with Visual C Compiler 2013 (version 120), which is shipped by default with Microsoft Visual Studio 2013. You must use this version of the Visual C Compiler and use Windows 7 or later.

If you are using Visual Studio 2015, you must configure it to use the 2013 compiler. Using Visual Studio 2017 is not supported. See [Compile the C client libraries with Visual Studio 2015](#) for details.

Other Windows compilers, such as Clang and GCC, are not supported.

When compiling with Visual C Compiler 2013, define `/D WIN32` in the compiler settings.

To use the C Unified API on Windows ensure that the following dependencies are available on your development system:

- Perl Compatible Regular Expressions (PCRE) library, version 8.3 or later
For more information, see <http://pcre.org>
- OpenSSL library, version 1.0.2a or later
For more information, see <https://www.openssl.org>

We provide these dependencies in the `diffusion-c-windows-version.zip` file.

The C client library statically links to APR version 1.5 with APR-util. Ensure that you set `APR_DECLARE_STATIC` and `APU_DECLARE_STATIC` before you use any APR includes. You can set these values in the following ways:

- By including `diffusion.h` before any APR includes. The `diffusion.h` file sets these values.
- As command-line flags

For more information, see <http://apr.apache.org>

On OS X/macOS

The C library is provided as a static library, `libdiffusion.a`, compiled for 64-bit OS X/macOS in the file `diffusion-c-osx-version.zip`.

To use the C Unified API on OS X/macOS ensure that the following dependencies are available on your development system:

- Perl Compatible Regular Expressions (PCRE) library, version 8.3 or later
For more information, see <http://pcre.org>

You can download this dependencies using `brew`.

The C client library statically links to APR version 1.5 with APR-util. Ensure that you set `APR_DECLARE_STATIC` and `APU_DECLARE_STATIC` before you use any APR includes. You can set these values in the following ways:

- By including `diffusion.h` before any APR includes. The `diffusion.h` file sets these values.
- As command-line flags

For more information, see <http://apr.apache.org>

For building using GCC, use Xcode 7.1 or later, which includes Apple LLVM.

Defining the structure of record topic data using XML

Data on record topics can be structured using metadata. Other Diffusion APIs provide builder methods you can use to define the metadata structure. The C Unified API uses XML to define the structure of a record topic's metadata.

The following schema describes the structure of that XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://
www.w3.org/2001/XMLSchema">

  <xs:element name="field" type="field"/>

  <xs:element name="message" type="message"/>

  <xs:element name="record" type="record"/>

  <xs:complexType name="record">
    <xs:complexContent>
      <xs:extension base="node">
        <xs:sequence>
          <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="record"/>
            <xs:element ref="field"/>
          </xs:choice>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="node">
    <xs:sequence/>
    <xs:attribute name="name" type="xs:string"
use="required"/>
    <xs:attribute name="multiplicity" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="field">
    <xs:complexContent>
      <xs:extension base="node">
        <xs:sequence/>
        <xs:attribute name="type" type="dataType"
use="required"/>
        <xs:attribute name="default" type="xs:string"/>
        <xs:attribute name="scale" type="xs:integer"/>
        <xs:attribute name="allowsEmpty"
type="xs:boolean"/>
        <xs:attribute name="customFieldHandlerClassName"
type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

```

</xs:complexContent>
</xs:complexType>

<xs:complexType name="message">
  <xs:complexContent>
    <xs:extension base="record">
      <xs:sequence/>
      <xs:attribute name="topicDataType"
type="topicDataType"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:simpleType name="dataType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="integerString"/>
    <xs:enumeration value="string"/>
    <xs:enumeration value="customString"/>
    <xs:enumeration value="decimalString"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="topicDataType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="record"/>
  </xs:restriction>
</xs:simpleType>
</xs:schema>

```

Threading model

The C Unified API is not thread-safe. Session and their derived artifacts must belong to a single thread or only be acted upon by a single thread at any time.

Internally, the C client creates threads for managing the connection to the Diffusion server. All callbacks into user-defined code are synchronous and it usually the case that these must execute as quickly as possible. If this code runs for a non-trivial amount of time, ensure that it hands off work to your own threads.

It is safe to send messages while processing callbacks, as outbound messages are queued and are sent as soon as possible.

Ensure that callbacks do not alter the session as this can lead to undefined behavior. This includes calling functions such as `session_close()` from the session state change callback.

Always call `session_close()` and `session_free()` from the same thread that created the session with `session_create()` or `session_create_async()`. This allows the threads to be joined and reaped correctly, and is a requirement of the APR library which the C Unified API relies on.

Regular expressions

The C client uses a different regular expression engine to the Diffusion server. Some regular expressions in topic selectors are evaluated on the client and others on the Diffusion server. It is possible that topic selectors that include complex or advanced regular expressions can behave differently on the client and on the Diffusion server.

For more information, see [Regular expressions](#) on page 67.

Feature support in the Diffusion Unified API

Review this information when designing your clients to determine which APIs provide the functionality you require.

Features are sets of capabilities provided by the Diffusion Unified API. Some features are not supported or not fully supported in some APIs.

The Diffusion libraries also provide capabilities that are not exposed through their APIs. Some of these capabilities can be configured.

Table 29: Capabilities provided by the Diffusion client libraries

Capability	JavaScript	Apple	Android	Java	.NET	C
Connecting						
Connect to the Diffusion server	✓	✓	✓	✓	✓	✓
Cascade connection through multiple transports	✓	✗	✓	✓	✗	✗
Connect asynchronously	✓	✓	✓	✓	✓	✓
Connect synchronously	✗	✗	✓	✓	✓	✓
Connect using a URL-style string as a parameter	✗	✓	✓	✓	✓	✓
Connect using individual parameters	✓	✗	✓	✓	✗	✗
Connect securely	✓	✓	✓	✓	✓	✓
Configure SSL context or behavior	✓	✓	✓	✓	✓	✗
Connect through an HTTP proxy	✗	✓	✓	✓	✓	✗
Connect through a load balancer	✓	✓	✓	✓	✓	✓
Pass a request path to a load balancer	✓	✗	✓	✓	✗	✗
Reconnecting						
Reconnect to the Diffusion server	✓	✓	✓	✓	✓	✓
Failover to a replicated session	✓	✓	✓	✓	✓	✓

Capability	JavaScript	Apple	Android	Java	.NET	C
on a different Diffusion server						
Configure a reconnection timeout	✓	✓	✓	✓	✓	✗
Define a custom reconnection strategy	✓	✓	✓	✓	✓	✓
Resynchronize message streams on reconnect	✓	✓	✓	✓	✓	✗
Abort reconnect if resynchronization fails	✓	✓	✓	✓	✓	✗
Maintain a recovery buffer of messages on the client to resend to the Diffusion server on reconnect	✓	✗	✓	✓	✗	✓
Configure the client-side recovery buffer	✗	✗	✓	✓	✗	✗
Detect disconnections by monitoring activity	✓	✓	✓	✓	✓	✗
Detect disconnections by using TCP state	✓	✓	✓	✓	✓	✓
Ping the Diffusion server	✗	✓	✓	✓	✓	✓
Change the principal used by the connected client session	✓	✓	✓	✓	✓	✓
Receiving data from topics						
Subscribe to a topic or set of topics	✓	✓	✓	✓	✓	✓
Receive data as a value stream (JSON, binary, and single value topics)	✓	✓	✓	✓	✓	✗
Receive data as content (all topic types)	✓	✓	✓	✓	✓	✓

Capability	JavaScript	Apple	Android	Java	.NET	C
Fetch the state of a topic	✓	✓	✓	✓	✓	✓
Managing topics						
Create a JSON or binary topic	✓	✓	✓	✓	✓	✗
Create a topic (not including JSON or binary topics)	✓	✓	✓	✓	✓	✓
Create a topic from an initial value	✓	✗	✓	✓	✓	✗
Create a topic with metadata	✓	✓	✓	✓	✓	✓
Listen for topic events (including topic has subscribers and topic has zero subscribers)	✗	✓	✓	✓	✓	✗
Delete a topic	✓	✓	✓	✓	✓	✓
Delete a branch of the topic tree	✓	✓	✓	✓	✓	✓
Mark a branch of the topic tree for deletion when this client session is closed	✓	✓	✓	✓	✓	✗
Updating topics						
Update a JSON or binary topic	✓	✗	✓	✓	✓	✗
Update a topic (not including JSON and binary topics)	✓	✓	✓	✓	✓	✓
Perform exclusive updates	✓	✓	✓	✓	✓	✓
Perform non-exclusive updates	✓	✓	✓	✓	✓	✗
Managing subscriptions						
Subscribe or unsubscribe another client to a topic	✓	✗	✓	✓	✓	✓
Subscribe or unsubscribe another client to a topic	✓	✗	✓	✓	✓	✗

Capability	JavaScript	Apple	Android	Java	.NET	C
based on session properties						
Handling subscriptions to routing topics	✗	✗	✓	✓	✓	✗
Handling subscriptions to missing topics	✓	✓	✓	✓	✓	✓
Messaging						
Send a message to a path	✓	✓	✓	✓	✓	✓
Send a message directly to a client	✓	✗	✓	✓	✓	✓
Send a message directly to a client based on session properties	✓	✗	✓	✓	✓	✓
Receive direct messages	✓	✓	✓	✓	✓	✓
Handle messages sent to a topic path	✓	✓	✓	✓	✓	✓
Managing security						
Authenticate client sessions and assign roles to client sessions	✗	✗	✓	✓	✓	✓
Configure how the Diffusion server authenticates client sessions and assign roles to client sessions	✓	✗	✓	✓	✓	✓
Configure the roles assigned to anonymous sessions and named sessions	✓	✗	✓	✓	✓	✓
Configure the permissions associated with roles assigned to client sessions	✓	✗	✓	✓	✓	✓
Managing other clients						
Receive notifications about client session events	✓	✗	✓	✓	✓	✓

Capability	JavaScript	Apple	Android	Java	.NET	C
including session properties						
Get the properties of a specific client session	✓	✗	✓	✓	✓	✓
Receive notifications about client queue events	✗	✗	✓	✓	✓	✗
Conflate and throttle clients	✗	✗	✓	✓	✓	✗
Close a client session	✗	✗	✓	✓	✓	✗
Push notifications (The Push Notification Bridge must be enabled)						
Receive push notifications	✗	✓	✓	✗	✗	✗
Request that push notifications be sent from a topic to a client	✓	✓	✓	✓	✓	✓
Publish an update to a topic that sends push notifications	✓	✓	✓	✓	✓	✓
Other capabilities						
Flow control	✗	✗	✓	✓	✓	✓

Best practice for developing clients

Follow these best practices to develop resilient and well performing clients.

Use an asynchronous programming model

All calls in the Unified API are asynchronous. Ensure that you code your client using asynchronous models to gain the advantages this provides.

Asynchronous calls remove the possibility of your client becoming blocked on a call. The Unified API also provides context-specific callbacks, enabling you to pass contextual information with a callback, and a wide range of event notifications.

Write good callbacks

The Unified API invokes callbacks using a thread from Diffusion thread pool. Callbacks for a particular session are called in order, one at a time. Consider the following when writing callbacks:

- Do not sleep or call blocking operations in a callback. If you do so, other pending callbacks for the session are delayed. If you must call a blocking operation, schedule it in a separate application thread.

- You can use the full Diffusion API to make other requests to the Diffusion server. If you want to make many requests based on a single callback notification, be aware that Diffusion client flow control is managed differently in callback threads. Less throttling is applied and it is easier to overflow the Diffusion server by issuing many thousands of requests. If you have a lot of requests to make, it is better to schedule the work in an application thread.

Use a modular design

The Diffusion Unified API provides interfaces on a feature-by-feature basis. There is a clear delineation between features. At runtime, the client starts only those services that it uses.

You can take advantage of the modular design of the Unified API by designing multiple smaller and more modular control clients. Smaller modules are easier to design, maintain and keep running. Develop separate clients for different control responsibilities. For example, have a client or set of clients responsible for authentication and a different client or set of clients responsible for creating topics.

Also consider separating the responsibility for different parts of the topic tree between clients. For example, have a client or set of clients responsible for updating the Tennis branch of the topic tree and a different client or set of clients responsible for updating the Rugby branch of the topic tree.

Make your client resilient and defensive

If the Diffusion server restarts, all topic information — tree structure and topic state — is removed, all subscription information is removed, and all clients are disconnected. Security and authentication information is persisted.

If your client disconnects and cannot reconnect to the same session, all of its subscriptions and any handlers it has registered are lost.

Ensure that you program your clients to handle and respond to these possibilities.

Getting started

Get started developing Diffusion clients by downloading one of our SDKs, discovering its capabilities, and starting to stream realtime data through the Diffusion server.

Start subscribing with JavaScript

Create a JavaScript browser client within minutes that connects to the Diffusion server. This example creates a web page that automatically updates and displays the value of a topic.

Before you begin

To complete this example, you need a Diffusion server and a web server where you can host your client application. Get the `diffusion.js` file from the `clients/js` directory of your Diffusion

You also require either a named user that has a role with the `select_topic` and `read_topic` permissions or that anonymous client connections are assigned a role with the `select_topic` and `read_topic` permissions. For example, the “CLIENT” role. For more information about roles and permissions, see [Role-based authorization](#) on page 130.

About this task

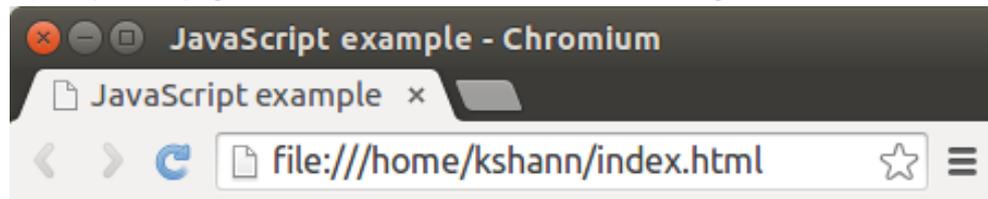
This example steps through the lines of code required to subscribe to a topic. There are several different [topic types](#) which provide data in different formats. This example shows you how to subscribe to a JSON topic. The [full code example](#) is provided after the steps.

Procedure

1. Create a template HTML page which displays the information.
For example, create the following `index.html` file.

```
<html>
  <head>
    <title>JavaScript example</title>
  </head>
  <body>
    <span>The value of foo/counter is: </span>
    <span id="update">Unknown</span>
  </body>
</html>
```

If you open the page in a web browser, it looks like the following screenshot:



The value of foo/counter is: Unknown

2. Include the Diffusion JavaScript library in the `<head>` section of your `index.html` file.

```
<head>
  <title>JavaScript example</title>
  <script type="text/javascript" src="path_to_library/
diffusion.js"></script>
</head>
```

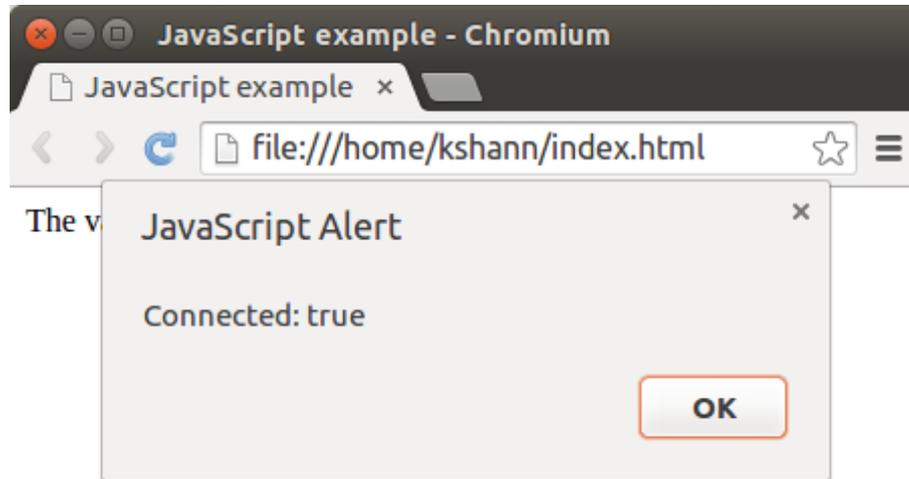
3. Create a connection from the page to the Diffusion server. Add a `script` element to the body element.

```
<body>
  <span>The value of foo/counter is: </span>
  <span id="update">Unknown</span>
  <script type="text/javascript">
    diffusion.connect({
      // Edit these lines to include the host and port of your
Diffusion server
      host : 'hostname',
      port : 'port',
      // To connect anonymously you can leave out the following
parameters
      principal : 'user',
      credentials : 'password'
    }).then(function(session) {
      alert('Connected: ' + session.isConnected());
```

```
    }  
  );  
</script>  
</body>
```

Where *hostname* is the name of the system hosting your Diffusion server, *user* is the name of a user with the permissions required to subscribe to a topic, and *password* is the user's password.

If you open the page in a web browser it looks like the following screenshot:



4. Subscribe to a topic and receive data from it.

Add the following function before the `diffusion.connect()` call:

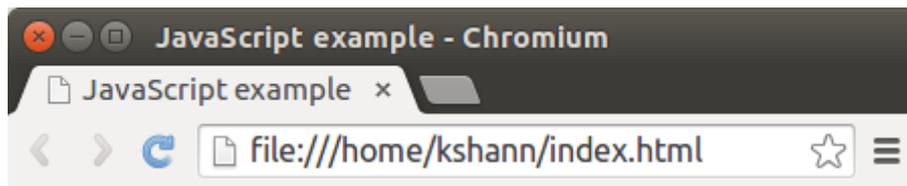
```
function subscribeToJsonTopic(session) {  
  session.subscribe('foo/counter');  
  session.stream('foo/  
counter').asType(diffusion.datatypes.json()).on('value',  
  function(topic, specification, newValue, oldValue) {  
    console.log("Update for " + topic,  
  newValue.get());  
    document.getElementById('display').innerHTML =  
  JSON.stringify(newValue.get());  
  });  
}
```

The `subscribe()` method of the `session` object takes the name of the topic to subscribe to and emits an update event. The attached function takes the data from the topic and updates the `update` element of the web page with the topic data.

5. Change the function that is called on connection to the `subscribeToJsonTopic` function you just created.

```
.then(subscribeToJsonTopic);
```

If you open the page in a web browser it looks like the following screenshot:



The value of foo/counter is: 27

Results

The web page is updated every time the value of the foo/counter topic is updated. You can update the value of the foo/counter topic by creating a publishing client to update the topic. To create and publish to the foo/counter topic, you require a user with the modify_topic and update_topic permissions. For more information, see [Start publishing with JavaScript](#) on page 202.

Example

The completed index.html file contains the following code:

```
<html>
  <head>
    <title>JavaScript example</title>

    <script type="text/javascript" src="path_to_library/
diffusion.js"></script>
  </head>
  <body>
    <div>
      <span>The value of foo/counter is: </span>
      <span id="update">Unknown</span>
    </div>

    <script type="text/javascript">
      function subscribeToJsonTopic(session) {
        session.subscribe('foo/counter');
        session.stream('foo/
counter').asType(diffusion.datatypes.json()).on('value',
function(topic, specification, newValue, oldValue) {
          console.log("Update for " + topic,
newValue.get());
          document.getElementById('display').innerHTML =
JSON.stringify(newValue.get());
        });
      }

      diffusion.connect({

        // Edit these lines to include the host and port of your
Diffusion server
        host : 'hostname',
        port : 'port',
```

```
// To connect anonymously you can leave out the
following parameters
    principal : 'user',
    credentials : 'password'
}).then(subscribeToJsonTopic);
</script>
</body>
</html>
```

Start subscribing with iOS

Create an Objective-C iOS client within minutes that connects to the Diffusion server. This example creates a client that prints the value of a topic to the console when the topic is updated.

Before you begin

To complete this example, you need Apple's Xcode installed on your development system and a Diffusion server.

You also require either a named user that has a role with the `select_topic` and `read_topic` permissions or that anonymous client connections are assigned a role with the `select_topic` and `read_topic` permissions. For example, the “CLIENT” role. For more information about roles and permissions, see [Role-based authorization](#) on page 130.

About this task

This example steps through the lines of code required to subscribe to a topic and was created using Xcode version 7.1 and the Diffusion dynamically linked framework targeted at iOS 8.

[Skip to the full example.](#)

Procedure

1. Get the Diffusion Apple SDK for iOS.

The `diffusion-iphoneos-version.zip` file is located in the `clients` directory of your Diffusion installation.

This example uses the iOS framework provided in `diffusion-iphoneos-version.zip`. Frameworks are also available for OS X/macOS-targeted development in `diffusion-macosx-version.zip` and tvOS-targeted development in `diffusion-tvos-version.zip`.

2. Extract the contents of the `diffusion-iphoneos-version.zip` file to your preferred location for third-party SDKs for use within Xcode.

For example, `~/Documents/code/SDKs/diffusion-iphoneos-version`.

3. Create a new project in Xcode.

- a) From the **File** menu, select **New > Project...**

The **Choose a template for your new project** wizard opens.

- b) Select **iOS > Application** on the left.

- c) Select **Single View Application** on the right and click **Next**.

Xcode prompts you to **Choose options for your new project**.

- d) Configure your project appropriately for your requirements.

Select **Objective-C** as the **Language**.

For example, use the following values:

- **Product Name:** TestClient
- **Language:** Objective-C
- **Devices:** Universal

For this example, it is not necessary to select **Use Core Data, Include Unit Tests, or Include UI Tests**.

- Click the **Next** button.
Xcode prompts you to select a destination directory for your new project.
- Select a target directory.
For example: `~/Documents/code/`
- Click the **Create** button.
Xcode creates a new project that contains the required files.

4. Import the Diffusion framework, `Diffusion.framework`, into Xcode.

- From the **View** menu, select **Navigators > Show Project Navigator**
- Select the root node of the project in the **Project Navigator** in the left sidebar.
The project editor opens in the main panel.
- In the project editor, click on the project name at the top left and select **Targets > TestClient**.
The target editor opens in the main panel.
- In the target editor, select the **General** tab.
- Expand the **Embedded Binaries** section and click the **+** icon (Add Items).
The **Choose items to add** wizard opens.
- Click the **Add Other...** button.
- Navigate to the location of the expanded `Diffusion.framework` SDK for iOS8 and click **Open**.
Xcode prompts you to choose options for adding these files.
- Select the options you require for adding the files and click **Finish**

Unless you require different options, use the following values:

- Do not select **Copy items if needed**.
- Select **Create groups**.

5. Make the framework visible to your code.

Despite the framework now appearing under both **Embedded Binaries** and **Linked Frameworks and Libraries**, it is still necessary to tell Xcode where the header files for the framework can be found.

- Select the **Build Settings** tab in the target editor.
You can also do this at project level if you prefer and depending on your requirements.
- Click on **All** to display All Build Settings.
- Expand the **Search Paths** section.
- Expand the **Framework Search Paths** section and click the **+** icon next to **Release**.
Add the absolute path to the `iOS8` directory that contains `Diffusion.framework`.

Note: Ensure that you use the path to the directory, not to the `Diffusion.framework` file. Linking to the file causes a silent failure.

6. Import the Diffusion module into the `ViewController.h` file.

```
@import Diffusion;
```

7. In the `ViewController.m` file, create a connection to the Diffusion server.

- Define a long-lived session property.

Add the session instance to the class extension to maintain a strong reference to the session instance:

```
@interface ViewController ()
@property PTDiffusionSession* session;
@end
```

The strong reference ensures that once the session instance has been opened, it remains open.

- b) Open a session connection to the Diffusion server.

This example opens the session when the view controller loads.

```
- (void)viewDidLoad {
    [super viewDidLoad];

    NSLog(@"Connecting...");

    // Connect anonymously.
    // Replace 'hostname' with that of your server.
    [PTDiffusionSession openWithURL:[NSURL
URLWithString:@"ws://hostname" ]
    completionHandler:^(PTDiffusionSession
    *session, NSError *error)
    {
        if (!session) {
            NSLog(@"Failed to open session: %@", error);
            return;
        }

        // At this point we now have a connected session.
        NSLog(@"Connected.");

        // Maintain strong reference to session instance.
        self.session = session;
    }];
}
```

Replace *hostname* with the host name of your Diffusion server.

8. Subscribe to a topic.

- a) In the `ViewController.h` file, conform to the topic stream delegate protocol.

In this example, the single view controller class handles topic stream update messages.

```
@interface ViewController : UIViewController
<PTDiffusionTopicStreamDelegate>
```

- b) In the `ViewController.h` file, create a `UILabel` control.

Add a label to your user interface and bind it to an outlet in the view controller:

```
@property(n nonatomic) IBOutlet UILabel *counterLabel;
```

This label is used to display the value of the topic as it updates.

- c) In the `ViewController.m` file, implement the topic stream update method.

This is a required method in the topic stream delegate protocol.

```
-(void)diffusionStream:(PTDiffusionStream *)stream
    didUpdateTopicPath:(NSString *)topicPath
    content:(PTDiffusionContent *)content
    context:(PTDiffusionUpdateContext *)context {
```

```

        // Interpret the received content's data as a string.
        NSString *counterString = [[NSString alloc]
initWithData:content.data
encoding:NSUTF8StringEncoding];

        // Diffusion sends messages to delegates on the main
dispatch queue so it's
        // safe to update the user interface here.
        self.counterLabel.text = counterString;
    }

```

- d) In the `ViewController.h` file, within the session's `completionHandler` callback, register with the Topics feature as the fallback topic stream handler.

```
[session.topics addFallbackTopicStreamWithDelegate:self];
```

- e) Next, request that the Diffusion server subscribe your session to the `foo/counter` topic.

```

[session.topics subscribeWithTopicSelectorExpression:@"foo/
counter"

completionHandler:^(NSError *error)
{
    if (error) {
        NSLog(@"Subscribe request failed. Error: %@",
error);
    } else {
        NSLog(@"Subscribe request succeeded.");
    }
}];

```

9. Build and Run.

Results

The client app updates the label every time the value of the `foo/counter` topic is updated. You can update the value of the `foo/counter` topic by creating a publishing client to update the topic. To create and publish to the `foo/counter` topic, you require a user with the `modify_topic` and `update_topic` permissions. For more information, see [Start publishing with OS X/macOS](#) on page 203.

Full example

The completed view controller implementation for the subscribing client contains the following code.

`ViewController.h`:

```

#import UIKit;

@interface ViewController : UIViewController
<PTDiffusionTopicStreamDelegate>
@property(nonatomic) IBOutlet UILabel *counterLabel;
@end

```

`ViewController.m`:

```

#import "ViewController.h"

@interface ViewController ()
@property PTDiffusionSession* session;
@end

```

```

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    NSLog(@"Connecting...");

    // Connect anonymously.
    // Replace 'hostname' with that of your server.
    [PTDiffusionSession openWithURL:[NSURL
URLWithString:@"ws://hostname"]
    completionHandler:^(PTDiffusionSession *session,
NSError *error)
    {
        if (!session) {
            NSLog(@"Failed to open session: %@", error);
            return;
        }

        // At this point we now have a connected session.
        NSLog(@"Connected.");

        // Maintain strong reference to session instance.
        self.session = session;

        // Register self as the fallback handler for topic
updates.
        [session.topics addFallbackTopicStreamWithDelegate:self];

        NSLog(@"Subscribing...");
        [session.topics
subscribeWithTopicSelectorExpression:@"foo/counter"

completionHandler:^(NSError *error)
        {
            if (error) {
                NSLog(@"Subscribe request failed. Error: %@",
error);
            } else {
                NSLog(@"Subscribe request succeeded.");
            }
        }
    ]];
    }

-(void)diffusionStream:(PTDiffusionStream *)stream
didUpdateTopicPath:(NSString *)topicPath
content:(PTDiffusionContent *)content
context:(PTDiffusionUpdateContext *)context {
    // Interpret the received content's data as a string.
    NSString *counterString = [[NSString alloc]
initWithData:content.data

encoding:NSUTF8StringEncoding];

    // Diffusion sends messages to delegates on the main dispatch
queue so it's
    // safe to update the user interface here.
    self.counterLabel.text = counterString;
}

```

```
@end
```

Start subscribing with Android

Create an Android client application within minutes that connects to the Diffusion server. This example creates a client that prints the value of a JSON topic to the console when the topic is updated.

Before you begin

To complete this example, you need Android Studio installed on your development system and a Diffusion server.

This example was tested in Android Studio 2.3.3. If you are using a different version of Android Studio, the details of some steps may vary slightly.

You also require that anonymous client connections are assigned a role with the `select_topic` and `read_topic` permissions. For example, the “CLIENT” role. For more information about roles and permissions, see [Role-based authorization](#) on page 130.

About this task

This example steps through the lines of code required to subscribe to a topic. The full code example is provided after the steps.

Procedure

1. Set up a project in Android Studio that uses the Diffusion Unified API.
 - a) Create a new project using API Level 21 or later.
 - b) Copy the `diffusion-android-x.x.x.jar` file into the `app/libs` folder of your project.
 - c) In Android Studio, right-click on the `libs` folder in the left-hand panel (the **Project Tool Window**), then select **Add as Library**.
If the `libs` folder is not shown in the left-hand panel, use the pull-down menu at the top of the panel to select **Project** view.
2. In your project's `AndroidManifest.xml` file set the `INTERNET` permission.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Insert the element between the opening `<manifest>` tag and the opening `<application>` tag. This permission is required to use the Diffusion API.

3. Open your project's `MainActivity.java` file.
This file is where you develop the code to interact with the Diffusion server.

The empty `MainActivity.java` file contains the following boilerplate code:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

```
}
```

4. Import the following packages and classes:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.callbacks.ErrorReason;
import com.pushtechology.diffusion.client.features.Topics;
import
    com.pushtechology.diffusion.client.features.Topics.TopicStream;
import com.pushtechology.diffusion.client.session.Session;
import com.pushtechology.diffusion.client.session.SessionFactory;
import
    com.pushtechology.diffusion.client.topics.details.TopicSpecification;
import com.pushtechology.diffusion.datatype.json.JSON;
import com.pushtechology.diffusion.datatype.json.JSONDataType;

public class MainActivity extends AppCompatActivity {

}
```

5. Create a `SessionHandler` inner class that implements `SessionFactory.OpenCallback`. This inner class will contain the code that interacts with the Diffusion server.

```
private class SessionHandler implements SessionFactory.OpenCallback
{
    private Session session = null;

    @Override
    public void onOpened(Session session) {
        this.session = session;
    }

    @Override
    public void onError(ErrorReason errorReason) {

    }

    public void close() {
        if ( session != null ) {
            session.close();
        }
    }
}
```

6. In the `onOpened` method, create the code required to subscribe to the `foo/counter` topic.
 - a) Get the `Topics` feature.

```
// Get the Topics feature to subscribe to topics
final Topics topics = session.feature( Topics.class );
```

- b) Add an instance of `Topics.ValueStream.Default<JSON>` as the topic stream for the `foo/counter` topic, and subscribe to the topic.

```
topics.addStream("foo/counter", JSON.class, new
Topics.ValueStream.Default<JSON>() {
    @Override
    public void onSubscription(String topicPath,
TopicSpecification specification) {
```

```

        Log.i("diffusion", "Subscribed to: " + topicPath);
    }

```

- c) Override the `onValue` method to print the value of the topic to the log when it changes.

```

@Override
public void onValue(
    String topicPath,
    TopicSpecification specification,
    JSON oldValue,
    JSON newValue) {

    Log.i("diffusion", topicPath + ": " +
        newValue.toJsonString());
}

```

7. In the `MainActivity` class, declare an instance of session handler.

```
private SessionHandler sessionHandler = null;
```

8. Override the `onCreate` method of the `MainActivity` class to open the session with the Diffusion server.

```

private SessionHandler sessionHandler = null;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    if (sessionHandler == null) {
        sessionHandler = new SessionHandler();

        Diffusion.sessions()
            .principal("username")
            .password("password")
            .open("ws://host:port", sessionHandler);
    }
}

```

You can connect securely, using:

```
Diffusion.sessions().open("wss://host:port",
    sessionHandler);
```

Or you can connect with a principal and credentials if that principal is assigned a role with the `select_topic` and `read_topic` permissions:

```
Diffusion.sessions().principal("username").password("password").open("wss://host:port",
    sessionHandler);
```

Replace the *host*, *port*, *principal*, and *password* values with your own information.

9. Override the `onDestroy` method of the `MainActivity` class to close the session with the Diffusion server.

```

if ( sessionHandler != null ) {
    sessionHandler.close();
    sessionHandler = null;
}

```

```
super.onDestroy();
```

10. Compile and run your client.

Results

The client outputs the value to the log console every time the value of the foo/counter JSON topic is updated. You can update the value of the foo/counter topic by creating a publishing client to update the topic. To create and publish to the foo/counter topic, you require a user with the modify_topic and update_topic permissions. For more information, see [Start publishing with Android](#) on page 210.

Full example

The completed MainActivity class contains the following code:

```
package com.pushtechology.demo.subscribe;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.callbacks.ErrorReason;
import com.pushtechology.diffusion.client.features.Topics;
import
    com.pushtechology.diffusion.client.features.Topics.TopicStream;
import com.pushtechology.diffusion.client.session.Session;
import com.pushtechology.diffusion.client.session.SessionFactory;
import
    com.pushtechology.diffusion.client.topics.details.TopicSpecification;
import com.pushtechology.diffusion.datatype.json.JSON;
import com.pushtechology.diffusion.datatype.json.JSONDataType;

public class MainActivity extends AppCompatActivity {
    /**
     * A session handler that maintains the diffusion session.
     */
    private class SessionHandler implements
        SessionFactory.OpenCallback {
        private Session session = null;

        @Override
        public void onOpened(Session session) {
            this.session = session;

            // Get the Topics feature to subscribe to topics
            final Topics topics = session.feature( Topics.class );

            // Subscribe to the "counter" topic and establish a
            JSON value stream
            topics.addStream("foo/counter", JSON.class, new
                Topics.ValueStream.Default<JSON>() {
                    @Override
                    public void onSubscription(String topicPath,
                        TopicSpecification specification) {
                        Log.i("diffusion", "Subscribed to: " +
                            topicPath);
                    }
                }

            @Override
            public void onValue(
                String topicPath,
```

```

        TopicSpecification specification,
        JSON oldValue,
        JSON newValue) {

            Log.i("diffusion", topicPath + ": " +
newValue.toJsonString());
        }
    });

    topics.subscribe("foo/counter", new
Topics.CompletionCallback.Default());

}

@Override
public void onError(ErrorReason errorReason) {
    Log.e( "Diffusion", "Failed to open session because: "
+ errorReason.toString() );
    session = null;
}

public void close() {
    if (session != null) {
        session.close();
    }
}

}

private SessionHandler sessionHandler = null;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    if (sessionHandler == null) {
        sessionHandler = new SessionHandler();

        Diffusion.sessions()
            .principal("username")
            .password("password")
            .open("wss://host:port", sessionHandler);
    }
}

@Override
protected void onDestroy() {
    if (sessionHandler != null ) {
        sessionHandler.close();
        sessionHandler = null;
    }

    super.onDestroy();
}
}
}

```

Related information

<http://developer.android.com/training/index.html>

Start subscribing with Java

Create a Java client within minutes that connects to the Diffusion server. This example creates a client that prints the value of a JSON topic to the console when the topic is updated.

Before you begin

To complete this example, you need a Diffusion server.

You also require either a named user that has a role with the `select_topic` and `read_topic` permissions or that anonymous client connections are assigned a role with the `select_topic` and `read_topic` permissions. For example, the “CLIENT” role. For more information about roles and permissions, see [Role-based authorization](#) on page 130.

About this task

This example steps through the lines of code required to subscribe to a topic. There are several different [topic types](#) which provide data in different formats. This example shows you how to subscribe to a JSON topic. The full code example is provided after the steps.

Procedure

1. Include the client jar file on the build classpath of your Java client. You can use one of the following methods:
 - You can use Maven to declare the dependency. First add the Push Technology public repository to your `pom.xml` file:

```
<repositories>
  <repository>
    <id>push-repository</id>
    <url>https://download.pushtechnology.com/maven/</url>
  </repository>
</repositories>
```

Next declare the following dependency in your `pom.xml` file:

```
<dependencies>
  <dependency>
    <groupId>com.pushtechnology.diffusion</groupId>
    <artifactId>diffusion-client</artifactId>
    <version>version</version>
  </dependency>
</dependencies>
```

Where *version* is the Diffusion version, for example 5.9.24.

- If you are not using Maven, you can include the `diffusion-client-version.jar` file that is located in the `clients/java` directory of your Diffusion server installation.

A `diffusion-api-5.9.0.jar` is also provided. This file contains only the development interfaces without any client library capabilities and can be used for developing and compiling your Java clients. However, to run your Diffusion Java client you must use the `diffusion-client-version.jar` file.

2. Create a client class that imports the following packages and classes:

```
import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.content.Content;
import com.pushtechology.diffusion.client.features.Topics;
import
    com.pushtechology.diffusion.client.features.Topics.ValueStream;
import com.pushtechology.diffusion.client.session.Session;
import
    com.pushtechology.diffusion.client.topics.details.TopicSpecification;
import com.pushtechology.diffusion.datatype.json.JSON;

public class SubscribingClient {

}
```

3. Create a main method for the client.

```
public class SubscribingClient {
    public static void main(String... arguments) throws
        Exception {

}
```

4. In the main method, connect to the Diffusion server.

```
        // Connect anonymously
        // Replace 'host' with your hostname
        final Session session =
            Diffusion.sessions().open("ws://host:port");
```

Or you can connect securely, using :

```
        final Session session =
            Diffusion.sessions().open("ws://host:port");
```

Or you can connect with a principal and credentials if that principal is assigned a role with the `read_topic` permission:

```
        final Session session =
            Diffusion.sessions().principal("principal")
                .password("password").open("ws://host:port");
```

Replace the *host*, *port*, *principal*, and *password* values with your own information.

5. Next, in the main method, get the Topics feature.

```
        // Get the Topics feature to subscribe to topics
        final Topics topics = session.feature(Topics.class);
```

The Topics feature enables a client to subscribe to a topic or fetch its state. For more information, see .

6. Within the `SubscribingClient` class, create an inner class that extends `ValueStream.Default<JSON>` and overrides the `onValue` method.

This inner class defines the behavior that occurs when a topic that the client subscribes to is updated. In this example, the value stream prints the topic name and the value of the update to the console.

```
private static class ValueStreamPrintLn extends
ValueStream.Default<Content> {
    @Override
    public void onValue(
        String topicPath,
        TopicSpecification specification,
        Content oldValue,
        Content newValue) {
        System.out.println(topicPath + ": " +
newValue.asString());
    }
}
```

7. Back in the main method of the `SubscribingClient` class, use the `addStream` method to associate an instance of the value stream that you created with the JSON topic you want to subscribe to.

```
// Add a new stream for 'foo/counter'
topics.addStream("foo/counter", JSON.class, new
Topics.ValueStream.Default<JSON>() {
    @Override
    public void onSubscription(String topicPath,
TopicSpecification specification) {
        System.out.println("Subscribed to: " + topicPath);
    }

    @Override
    public void onValue(String topicPath,
TopicSpecification specification, JSON oldValue, JSON newValue) {
        System.out.println(topicPath + " : " +
newValue.toJsonString());
    }
});
```

8. Next, use the `subscribe` method to subscribe to the topic `foo/counter`.

```
// Subscribe to the topic 'foo/counter'
topics.subscribe("foo/counter", new
Topics.CompletionCallback.Default());
```

9. Use a `Thread.sleep()` to hold the client open for a minute while the updates are received and output.

```
// Wait for a minute while the stream prints updates
Thread.sleep(60000);
```

10. Compile and run your client.

Ensure that the `diffusion-client-version.jar` file is included in your compiled client or on its classpath.

We recommend that you run your client using the JDK rather than the JRE. The JDK includes additional diagnostic capabilities that might be useful.

Results

The client outputs the value to the console every time the value of the foo/counter topic is updated. You can update the value of the foo/counter topic by creating a publishing client to update the topic. To create and publish to the foo/counter topic, you require a user with the modify_topic and update_topic permissions. For more information, see [Start publishing with Java](#) on page 215.

Full example

The completed `SubscribingClient` class contains the following code:

```
import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.content.Content;
import com.pushtechology.diffusion.client.features.Topics;
import
    com.pushtechology.diffusion.client.features.Topics.ValueStream;
import com.pushtechology.diffusion.client.session.Session;
import
    com.pushtechology.diffusion.client.topics.details.TopicSpecification;
import com.pushtechology.diffusion.datatype.json.JSON;

/**
 * A client that subscribes to the topic 'foo/counter.
 *
 * @author Push Technology Limited
 * @since 5.9
 */
public class SubscribingClient {

    /**
     * Main.
     */
    public static void main(String... arguments) throws Exception
    {

        // Connect anonymously
        // Replace 'host' with your hostname
        final Session session =
            Diffusion.sessions().open("ws://host:port");

        // Get the Topics feature to subscribe to topics
        final Topics topics = session.feature(Topics.class);

        // Add a new stream for 'foo/counter'
        topics.addStream("foo/counter", JSON.class, new
            Topics.ValueStream.Default<JSON>() {
                @Override
                public void onSubscription(String topicPath,
                    TopicSpecification specification) {
                    System.out.println("Subscribed to: " + topicPath);
                }

                @Override
                public void onValue(String topicPath,
                    TopicSpecification specification, JSON oldValue, JSON newValue) {
                    System.out.println(topicPath + " : " +
                        newValue.toJsonString());
                }
            });

        // Subscribe to the topic 'foo/counter'
```

```

        topics.subscribe("foo/counter", new
Topics.CompletionCallback.Default());

        // Wait for a minute while the stream prints updates
        Thread.sleep(60000);
    }

    /**
     * A topic stream that prints updates to the console.
     */
    private static class ValueStreamPrintLn extends
ValueStream.Default<Content> {
        @Override
        public void onValue(
            String topicPath,
            TopicSpecification specification,
            Content oldValue,
            Content newValue) {
            System.out.println(topicPath + ": " +
newValue.asString());
        }
    }
}

```

Start subscribing with .NET

Create a .NET client within minutes that connects to the Diffusion server. This example creates a client that prints the value of a JSON topic to the console when the topic is updated.

Before you begin

To complete this example, you need a Diffusion server.

You also require either a named user that has a role with the `select_topic` and `read_topic` permissions or that anonymous client connections are assigned a role with the `select_topic` and `read_topic` permissions. For example, the "CLIENT" role. For more information about roles and permissions, see [Role-based authorization](#) on page 130.

About this task

This example steps through the lines of code required to subscribe to a topic. The full code example is provided after the steps.

Procedure

1. Create a .NET project that references the following DLL file located in the `clients/dotnet` directory of your .NET installation:

PushTechnology.ClientInterface.dll

The assembly contains the interface classes and interfaces that you use when creating a control client.

The .NET assembly is also available through NuGet.

Use the following command in the NuGet Package Manager Console:

```
PM> Install-Package PushTechnology.UnifiedClientInterface
```

2. In your project, create a C# file that uses the following packages:

```
using System;
using System.Threading;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Features.Topics;
using PushTechnology.ClientInterface.Client.Topics.Details;
using PushTechnology.ClientInterface.Data.JSON;

namespace PushTechnology.ClientInterface.GettingStarted {
    public sealed class SubscribingClient {
    }
}
```

3. Create a Main method.

```
public sealed class SubscribingClient
{
    public static void Main( string[] args ) {
    }
}
```

4. In the Main method, connect to the Diffusion server.

```
public static void Main( string[] args ) {
    // Connect anonymously
    // Replace 'host' with your hostname
    var session =
Diffusion.Sessions.Open("ws://host:port");
}
```

Or you can connect securely, using :

```
Session session =
Diffusion.sessions().Open("wss://host:port");
```

Or you can connect with a principal and credentials if that principal is assigned a role with the read_topic permission:

```
Session session =
Diffusion.Sessions().Principal("principal")
.Password("password").Open("wss://host:port");
```

Replace the *host*, *port*, *principal*, and *password* values with your own information.

5. Next, in the Main method, get the Topics feature.

```
// Get the Topics feature to subscribe to topics
var topics = session.GetTopicsFeature();
```

The Topics feature enables a client to subscribe to a topic or fetch its state.

6. Within the subscribing client class, create an inner class that implements IValueStream for a JSON topic.

This inner class defines the behavior that occurs when a topic that the client subscribes to is updated. In this example, the topic stream prints the topic name and the content of the update to the console.

```

    /// <summary>
    /// Basic implementation of the IValueStream<TValue> for
JSON topics.
    /// </summary>
    internal sealed class JSONStream : IValueStream<IJSON> {

        /// <summary>
        /// Notification of stream being closed normally.
        /// </summary>
        public void OnClose() {
            Console.WriteLine( "The subscription stream is now
closed." );
        }

        /// <summary>
        /// Notification of a contextual error related to this
callback.
        /// </summary>
        /// <remarks>
        /// Situations in which <code>OnError</code> is called
include the session being closed, a communication
        /// timeout, or a problem with the provided parameters.
        No further calls will be made to this callback.
        /// </remarks>
        /// <param name="errorReason">Error reason.</param>
        public void OnError( ErrorReason errorReason ) {
            Console.WriteLine( "An error has occurred : {0}",
errorReason );
        }

        /// <summary>
        /// Notification of a successful subscription.
        /// </summary>
        /// <param name="topicPath">Topic path.</param>
        /// <param name="specification">Topic specification.</
param>
        public void OnSubscription( string topicPath,
ITopicSpecification specification ) {
            Console.WriteLine( "Client subscribed to {0} ",
topicPath );
        }

        /// <summary>
        /// Notification of a successful unsubscription.
        /// </summary>
        /// <param name="topicPath">Topic path.</param>
        /// <param name="specification">Topic specification.</
param>
        /// <param name="reason">Error reason.</param>
        public void OnUnsubscription( string topicPath,
ITopicSpecification specification, TopicUnsubscribeReason reason )
        {
            Console.WriteLine( "Client unsubscribed from {0} :
{1}", topicPath, reason );
        }

        /// <summary>
        /// Topic update received.

```

```

        /// </summary>
        /// <param name="topicPath">Topic path.</param>
        /// <param name="specification">Topic specification.</
param>
        /// <param name="oldValue">Value prior to update.</param>
        /// <param name="newValue">Value after update.</param>
        public void OnValue( string topicPath,
ITopicSpecification specification, IJSON oldValue, IJSON
newValue ) {
            Console.WriteLine( "New value of {0} is {1}",
topicPath, newValue.ToJSONString() );
        }
    }
}

```

7. Back in the `Main` method of the subscribing client class, use the `AddStream` method to associate an instance of the topic stream that you created with the topic you want to subscribe to. In this example, the topic path is `"foo/counter"`.

```

        var topic = ">foo/counter";
        // Add a topic stream for 'foo/counter' and request
subscription
        var jsonStream = new JSONStream();
        topics.AddStream( topic, jsonStream );
        topics.Subscribe( topic, new
TopicsCompletionCallbackDefault() );
    }
}

```

8. Use a `Thread.sleep()` to hold the client open for 10 minutes while the updates are received and output.

```

        //Stay connected for 10 minutes
        Thread.Sleep( TimeSpan.FromMinutes( 10 ) );

        session.Close();
    }
}

```

9. Compile and run your client.

Results

The client outputs the value to the console every time the value of the `foo/counter` topic is updated. You can update the value of the `foo/counter` topic by creating a publishing client to update the topic. To create and publish to the `foo/counter` topic, you require a user with the `modify_topic` and `update_topic` permissions. For more information, see [Start publishing with .NET](#) on page 220.

Full example

The completed subscribing client class contains the following code:

```

using System;
using System.Threading;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Features.Topics;
using PushTechnology.ClientInterface.Client.Topics.Details;
using PushTechnology.ClientInterface.Data.JSON;

namespace PushTechnology.ClientInterface.GettingStarted {
    /// <summary>
    /// A client that subscribes to the topic 'foo/counter'.
    /// </summary>
    public sealed class SubscribingClient {

```

```

        public static void Main( string[] args ) {
            // Connect anonymously
            var session = Diffusion.Sessions.Open( "ws://
localhost:8080" );

            // Get the Topics feature to subscribe to topics
            var topics = session.GetTopicsFeature();

            var topic = ">foo/counter";
            // Add a topic stream for 'foo/counter' and request
subscription
            var jsonStream = new JSONStream();
            topics.AddStream( topic, jsonStream );
            topics.Subscribe( topic, new
TopicsCompletionCallbackDefault() );

            //Stay connected for 10 minutes
            Thread.Sleep( TimeSpan.FromMinutes( 10 ) );

            session.Close();
        }
    }

    /// <summary>
    /// Basic implementation of the IValueStream<TValue> for JSON
topics.
    /// </summary>
    internal sealed class JSONStream : IValueStream<IJSON> {

        /// <summary>
        /// Notification of stream being closed normally.
        /// </summary>
        public void OnClose() {
            Console.WriteLine( "The subscription stream is now
closed." );
        }

        /// <summary>
        /// Notification of a contextual error related to this
callback.
        /// </summary>
        /// <remarks>
        /// Situations in which <code>OnError</code> is called
include the session being closed, a communication
        /// timeout, or a problem with the provided parameters. No
further calls will be made to this callback.
        /// </remarks>
        /// <param name="errorReason">Error reason.</param>
        public void OnError( ErrorReason errorReason ) {
            Console.WriteLine( "An error has occured : {0}",
errorReason );
        }

        /// <summary>
        /// Notification of a successful subscription.
        /// </summary>
        /// <param name="topicPath">Topic path.</param>
        /// <param name="specification">Topic specification.</
param>
        public void OnSubscription( string topicPath,
ITopicSpecification specification ) {
            Console.WriteLine( "Client subscribed to {0} ",
topicPath );
        }
    }
}

```

```

    }

    /// <summary>
    /// Notification of a successful unsubscription.
    /// </summary>
    /// <param name="topicPath">Topic path.</param>
    /// <param name="specification">Topic specification.</
param>
    /// <param name="reason">Error reason.</param>
    public void OnUnsubscription( string topicPath,
ITopicSpecification specification, TopicUnsubscribeReason
reason ) {
        Console.WriteLine( "Client unsubscribed from {0} :
{1}", topicPath, reason );
    }

    /// <summary>
    /// Topic update received.
    /// </summary>
    /// <param name="topicPath">Topic path.</param>
    /// <param name="specification">Topic specification.</
param>
    /// <param name="oldValue">Value prior to update.</param>
    /// <param name="newValue">Value after update.</param>
    public void OnValue( string topicPath, ITopicSpecification
specification, IJSON oldValue, IJSON newValue ) {
        Console.WriteLine( "New value of {0} is {1}",
topicPath, newValue.ToJSONString() );
    }
}
}
}

```

Start subscribing with C

Create a C client within minutes that connects to the Diffusion server. This example creates a client that prints the value of a topic to the console when the topic is updated.

Before you begin

The C client libraries rely on a number of dependencies. Depending on which platform you are using the C client libraries for, these dependencies might be included in the client library. If they are not included in the client library, ensure that the dependencies are available on your development system.

For more information about dependencies on each supported platform, see [C](#) on page 164.

The C client library statically links to APR version 1.5 with APR-util. Ensure that you set `APR_DECLARE_STATIC` and `APU_DECLARE_STATIC` before you use any APR includes. You can set these values in the following ways:

- By including `diffusion.h` before any APR includes. The `diffusion.h` file sets these values.
- As command-line flags

For more information, see <http://apr.apache.org>

To complete this example, you need a Diffusion server.

You also require either a named user that has a role with the `select_topic` and `read_topic` permissions or that anonymous client connections are assigned a role with the `select_topic` and `read_topic` permissions. For example, the “CLIENT” role. For more information about roles and permissions, see [Role-based authorization](#) on page 130.

About this task

This example steps through the lines of code required to subscribe to a topic. The [full code example](#) is provided after the steps.

Procedure

1. Get the Diffusion C client library for your platform and extract the ZIP file.
The C client library is available in the `clients/c` directory of the Diffusion installation.
2. Create a C file called `getting-started.c`.
 - a) Include the following libraries:

```
#include <stdio.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"
```

- b) Create a main method:

```
int
main(int argc, char **argv)
{
}
```

3. Create a connection to the Diffusion server.
Inside the main method add the following lines:

```
/*
 * Create a session
 */
DIFFUSION_ERROR_T error = { 0 };
SESSION_T *session = NULL;
// Edit this line to include the host and port of your
Diffusion server
session = session_create("ws://hostname:port", "user",
credentials_create_password("password"), NULL, NULL, &error);
if(session == NULL) {
    fprintf(stderr, "TEST: Failed to create session
\n");
    fprintf(stderr, "ERR : %s\n", error.message);
    return EXIT_FAILURE;
} else {
    fprintf(stdout, "Connected\n");
}
```

Where *hostname* is the name of the system hosting your Diffusion server, *hostname* is the name the Diffusion server accepts client connections on, *user* is the name of a user with the permissions required to subscribe to a topic, and *password* is the user's password.

The client logs the string "Connected" to the console if the connection is a success.

4. Subscribe to the topic `foo/counter`.
 - a) Above the main method, create a callback for when subscription occurs.

```
static int
on_subscribe(SESSION_T *session, void *context_data)
{
    printf("Subscribed to topic\n");
    return HANDLER_SUCCESS;
}
```

```
}
```

This callback prints a message to the console when the client subscribes to the `foo/counter` topic.

- b) Above the `main` method, create a callback for when an update is received.

```
static int
on_topic_message(SESSION_T *session, const TOPIC_MESSAGE_T *msg)
{
    printf("%.s\n", (int)msg->payload->len, msg->payload->data);
    return HANDLER_SUCCESS;
}
```

This callback prints the value of the topic to the console every time the `foo/counter` topic is updated.

- c) In the `main` method, call the `subscribe` method.

```
subscribe(session, (SUBSCRIPTION_PARAMS_T)
{ .topic_selector = ">foo/counter", .on_topic_message =
on_topic_message, .on_subscribe = on_subscribe });
```

5. Set the client to wait for 5 minutes before closing.

```
/*
 * Receive messages for 5 minutes.
 */
sleep(300);

session_close(session, NULL);
session_free(session);

return EXIT_SUCCESS;
```

6. Build your C client.

- a) Create a `Makefile` in the same directory as your C file.

An [example Makefile](#) is provided after the steps.

- b) Ensure that your `Makefile` links to the `include` and `lib` directory of the Diffusion C library.

```
DIFFUSION_C_CLIENT_INCDIR = ../path-to-client/include
DIFFUSION_C_CLIENT_LIBDIR = ../path-to-client/lib
```

- c) Run the `make` command to build the example.

The `getting-started` binary is created in the `target/bin` directory.

7. Run your C client from the command line.

Results

The client prints the value to the console every time the value of the `foo/counter` topic is updated. You can update the value of the `foo/counter` topic by creating a publishing client to update the topic. To create and publish to the `foo/counter` topic, you require a user with the `modify_topic` and `update_topic` permissions. For more information, see [Start publishing with C](#) on page 227.

Example

The completed `getting-started.c` file contains the following code:

```
#include <stdio.h>
#include <unistd.h>
```

```

#include "diffusion.h"
#include "args.h"

/*
 * When a subscribed message is received, this callback is
 * invoked.
 */
static int
on_topic_message(SESSION_T *session, const TOPIC_MESSAGE_T *msg)
{
    printf("%.s\n", (int)msg->payload->len, msg->payload->
>data);
    return HANDLER_SUCCESS;
}

/*
 * This callback is fired when Diffusion responds to say that a
 * topic
 * subscription request has been received and processed.
 */
static int
on_subscribe(SESSION_T *session, void *context_data)
{
    printf("Subscribed to topic\n");
    return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
    /*
     * Create a session
     */
    DIFFUSION_ERROR_T error = { 0 };
    SESSION_T *session = NULL;

    // Edit this line to include the host and port of your
    Diffusion server
    session = session_create("ws://hostname:port", "user",
credentials_create_password("password"), NULL, NULL, &error);
    if(session == NULL) {
        fprintf(stderr, "TEST: Failed to create session
\n");
        fprintf(stderr, "ERR : %s\n", error.message);
        return EXIT_FAILURE;
    } else {
        fprintf(stdout, "Connected\n");
    }

    subscribe(session, (SUBSCRIPTION_PARAMS_T)
{ .topic_selector = ">foo/counter", .on_topic_message =
on_topic_message, .on_subscribe = on_subscribe });

    /*
     * Receive messages for 5 minutes.
     */
    sleep(300);

    session_close(session, NULL);
    session_free(session);
}

```

```
        return EXIT_SUCCESS;
    }
}
```

The Makefile contains the following code:

```
# The following two variables must be set.
#
# Directory containing the C client include files.
DIFFUSION_C_CLIENT_INCDIR = ../path-to-client/include
#
# Directory containing libdiffusion.a
DIFFUSION_C_CLIENT_LIBDIR = ../path-to-client/lib

ifndef DIFFUSION_C_CLIENT_INCDIR
$(error DIFFUSION_C_CLIENT_INCDIR is not set)
endif

ifndef DIFFUSION_C_CLIENT_LIBDIR
$(error DIFFUSION_C_CLIENT_LIBDIR is not set)
endif

CC = gcc

# Extra definitions from parent directory, if they exist.
-include ../makefile.defs

CFLAGS += -g -Wall -Werror -std=c99 -D_POSIX_C_SOURCE=200112L -
D_XOPEN_SOURCE=700 -c -I$(DIFFUSION_C_CLIENT_INCDIR)
LDFLAGS += $(LIBS) $(DIFFUSION_C_CLIENT_LIBDIR)/libdiffusion.a -
lpthread -lpcrc -lssl -lcrypto

ARFLAGS +=
SOURCES = getting-started.c

TARGETDIR = target
OBJDIR = $(TARGETDIR)/objs
BINDIR = $(TARGETDIR)/bin
OBJECTS = $(SOURCES:.c=.o)
TARGETS = getting-started

all: prepare $(TARGETS)
.PHONY: all

prepare:
    mkdir -p $(OBJDIR) $(BINDIR)

$(OBJDIR)/%.o: %.c
    $(CC) $(CFLAGS) -o $@ $<

getting-started: $(OBJDIR)/getting-started.o
    $(CC) $< $(LDFLAGS) -o $(BINDIR)/$@

clean:
    rm -rf $(TARGETS) $(OBJECTS) $(TARGETDIR) core a.out
```

Start publishing with JavaScript

Create a Node.js client that publishes data through topics on the Diffusion server.

Before you begin

To complete this example, you need a Diffusion server and a development system with [Node.js](#) and [npm](#) installed on it.

You also require either a named user that has a role with the `modify_topic` and `update_topic` permissions. For example, the “ADMINISTRATOR” role. For more information about roles and permissions, see [Role-based authorization](#) on page 130.

About this task

This example steps through the lines of code required to publish to a topic. There are several different [topic types](#) which provide data in different formats. This example shows you how to publish to a JSON topic. The full code example is provided after the steps.

Procedure

1. Install the Diffusion JavaScript library on your development system.

```
npm install --save diffusion
```

2. Create the JavaScript file that will be your publishing client.

For example, `publishing.js`

- a) Require the Diffusion library.

```
const diffusion = require('diffusion');
```

- b) Connect to the Diffusion server.

```
diffusion.connect({
  host : 'host-name',
  principal : 'control-user',
  credentials : 'password'
}).then(function(session) {
  console.log('Connected!');
});
```

Where *host-name* is the name of the system that hosts your Diffusion server, *control-user* is the name of a user with the permissions required to create and update topics, and *password* is the user's password.

- c) Create a JSON topic called `foo/counter`.

```
session.topics.add("foo/counter",
  diffusion.topics.TopicType.JSON);
```

- d) Every second update the value of the topic with the value of the counter.

```
setInterval(function() {
  session.topics.update('foo/counter', { count : i++ });
}, 1000);
```

3. Use Node.js to run your publishing client from the command line.

```
node publishing.js
```

Results

The publisher updates the value of the foo/counter topic every second. You can watch the topic value being updated by subscribing to the topic.

- You can use the example subscribing client from [Start subscribing with JavaScript](#) on page 174 to subscribe to foo/counter and output the value on a web page.

Example

The completed publishing.js file contains the following code:

```
const diffusion = require('diffusion');

diffusion.connect({
  host : 'hostname',
  principal : 'control-user',
  credentials : 'password'
}).then(function(session) {
  console.log('Connected!');

  var i = 0;

  // Create a JSON topic
  session.topics.add("foo/counter",
diffusion.topics.TopicType.JSON);

  // Start updating the topic every second
  setInterval(function() {

    session.topics.update("foo/counter", { count : i++ });

  }, 1000);
});
```

What to do next

Now that you have the outline of a publisher, you can use it to publish your own data instead of a counter.

Start publishing with OS X/macOS

Create an OS X/macOS client that publishes data through topics on the Diffusion server.

Before you begin

To complete this example, you need Apple's Xcode installed on your development system and a Diffusion server.

You also require a named user that has a role with the modify_topic and update_topic permissions. For example, the "TOPIC_CONTROL" role. For more information about roles and permissions, see [Role-based authorization](#) on page 130.

About this task

This example steps through the lines of code required to create and publish to a topic and was created using Xcode version 7.1 and the Diffusion dynamically linked framework targeted at OS X/macOS.

[Skip to the full example.](#)

Procedure

1. Get the Diffusion Apple SDK for OS X/macOS.

The `diffusion-macosx-version.zip` file is located in the `clients` directory of your Diffusion installation.

This example uses the OS X/macOS framework provided in `diffusion-macosx-version.zip`. Frameworks are also available for iOS-targeted development in `diffusion-iphones-version.zip` and tvOS-targeted development in `diffusion-tvos-version.zip`.

2. Extract the contents of the `diffusion-macosx-version.zip` file to your preferred location for third-party SDKs for use within Xcode.

For example, `~/Documents/code/SDKs/diffusion-macosx-version`.

3. Create a new project in Xcode.

- a) From the **File** menu, select **New > Project...**

The **Choose a template for your new project** wizard opens.

- b) Select **OS X > Application** on the left.

- c) Select **Command Line Tool** on the right and click **Next**.

Xcode prompts you to **Choose options for your new project**.

- d) Configure your project appropriately for your requirements.

Select **Objective-C** as the **Language**.

For example, use the following values:

- **Product Name:** CounterPublisher
- **Language:** Objective-C

- e) Click the **Next** button.

Xcode prompts you to select a destination directory for your new project.

- f) Select a target directory.

For example: `~/Documents/code/`

- g) Click the **Create** button.

Xcode creates a new project that contains the required files.

4. Link to the Diffusion framework.

For more information, see the Xcode documentation <http://help.apple.com/xcode/mac/8.1/#/dev51a648b07>.

5. Create a `CounterPublisher.h` file.

- a) Bring up the context menu for the root node of the project in the **Project Navigator** in the left sidebar.

Select **New File**

- b) In the dialog that opens, select **Header File**

- c) Name the header file `CounterPublisher.h` and click **Create**.

6. Import the `Foundation` module into the `CounterPublisher.h` file.

```
@import Foundation;
```

7. Define the `CounterPublisher` interface in the `CounterPublisher.h` file.

```
@interface CounterPublisher : NSObject  
  
-(void)startWithURL:(NSURL *)url;  
  
@end
```

8. Create a `CounterPublisher.m` file.
 - a) Bring up the context menu for the root node of the project in the **Project Navigator** in the left sidebar.
 - Select **New File**
 - b) In the dialog that opens, select **Objective-C File**
 - c) Name the file `CounterPublisher.m` and click **Create**.

9. In the `CounterPublisher.m` file, import the required modules and set up properties and variables.

- a) Import `Diffusion` and `CounterPublisher.h`

```
#import "CounterPublisher.h"
#import Diffusion;
```

- b) Define a long-lived session property.

Add the session instance to the class extension to maintain a strong reference to the session instance:

```
@interface CounterPublisher ()
@property(nonatomic) PTDiffusionSession* session;
@end
```

The strong reference ensures that once the session instance has been opened, it remains open.

- c) Declare an integer `_counter` to hold the value to publish to the topic.

```
@implementation CounterPublisher {
    NSInteger _counter;
}
```

- d) Define the topic path of the topic to create and publish to.

```
static NSString *const _TopicPath = @"foo/counter";
```

10. In the `CounterPublisher.m` file, create a method that starts the session with the Diffusion server.

- a) Call the method `startWithURL` and give it a signature that matches that defined in `CounterPublisher.h`

```
-(void)startWithURL:(NSURL *)url {
}

```

- b) Inside the method, define the security principal and credentials that the client uses to connect.

```
PTDiffusionCredentials *credentials =
    [[PTDiffusionCredentials alloc]
     initWithPassword:@"password"];
PTDiffusionSessionConfiguration *sessionConfiguration =
    [[PTDiffusionSessionConfiguration alloc]
     initWithPrincipal:@"principal"
     credentials:credentials];
```

Replace *principal* and *password* with the username and password to connect to the Diffusion server with. This user must have sufficient permissions to create and update the topic, for example, by being assigned the "TOPIC_CONTROL" role.

c) Open a session on the Diffusion server.

```
[PTDiffusionSession openWithURL:url
                      configuration:sessionConfiguration
                      completionHandler:^(PTDiffusionSession
*session, NSError *error)
{
    if (!session) {
        NSLog(@"Failed to open session: %@", error);
        return;
    }

    // At this point we now have a connected session.
    NSLog(@"Connected.");

    // Maintain strong reference to session instance.
    self.session = session;

    // Next step

}];
```

11. Create a topic.

After connecting a session and creating a strong reference to it, use `session.topicControl` to create a topic:

```
// Send request to add topic for publishing to.
[session.topicControl addWithTopicPath:_TopicPath

type:PTDiffusionTopicType_SingleValue
                                value:nil
                                completionHandler:^(NSError *error)
{
    if (error) {
        NSLog(@"Failed to add topic: %@", error);
        return;
    }

    // Next step

}];
```

12. Update the topic.

a) After successfully creating the topic, call the `updateCounter` method:

```
[self updateCounter];
```

b) At the top-level of the `CounterPublisher.m` file, define the `updateCounter` method:

```
-(void)updateCounter {
    // Get the updater to be used for non-exclusive topic
    updates.
    PTDiffusionTopicUpdater *updater =
    self.session.topicUpdateControl.updater;

    // Format string content for the update.
    NSString *string = [NSString stringWithFormat:@"%lu",
(unsigned long)_counter++];
    NSData *data = [string
dataUsingEncoding:NSUTF8StringEncoding];
```

```

PTDiffusionContent *content = [[PTDiffusionContent alloc]
initWithData:data];

// Send request to update topic.
NSLog(@"Updating: %@", string);
[updater updateWithTopicPath:_TopicPath
value:content
completionHandler:^(NSError *error)
{
    if (error) {
        NSLog(@"Failed to update topic: %@", error);
    }
}];

// Schedule another update in one second's time.
__weak CounterPublisher *const weakSelf = self;
dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(1.0
* NSEC_PER_SEC)),
dispatch_get_main_queue(), ^
{
    [weakSelf updateCounter];
});
}

```

This method recursively calls itself via a weak reference to self.

13. In the `main.m`, add the code needed to run your publishing client from the command line:

- a) Import the Foundation module and the `CounterPublisher.h` file.

```

#import Foundation;
#import "CounterPublisher.h"

```

- b) In a main method, create a `CounterPublisher` and call its `startWithURL` method:

```

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        CounterPublisher *const publisher = [CounterPublisher
new];
        NSURL *const url = [NSURL
URLWithString:@"wss://hostname"];
        [publisher startWithURL:url];

        // Run in an infinite loop.
        [[NSRunLoop currentRunLoop] run];
    }
    return 0;
}

```

Replace `hostname` with the host name of your Diffusion server.

14. Build and Run.

Results

The client publishes a value to the `foo/counter` topic every second. You can subscribe to the `foo/counter` topic by creating a client to subscribe to the topic. For more information, see [Start subscribing with iOS](#) on page 178.

Full example

The completed implementation of the publishing client files contain the following code:

main.m:

```
@import Foundation;
#import "CounterPublisher.h"

/**
 Wrapper around the counter publisher example class demonstrating
 how it can
 be launched as a command line tool.
 */
int main(int argc, const char * argv[]) {
    @autoreleasepool {
        CounterPublisher *const publisher = [CounterPublisher
new];
        NSURL *const url = [NSURL
URLWithString:@"wss://hostname"];
        [publisher startWithURL:url];

        // Run, Infinite Loop.
        [[NSRunLoop currentRunLoop] run];
    }
    return 0;
}
```

CounterPublisher.h:

```
@import Foundation;

@interface CounterPublisher : NSObject

-(void)startWithURL:(NSURL *)url;

@end
```

CounterPublisher.m:

```
@import Diffusion;

@interface CounterPublisher ()
@property(n nonatomic) PTDiffusionSession* session;
@end

@implementation CounterPublisher {
    NSUInteger _counter;
}

@synthesize session = _session;

static NSString *const _TopicPath = @"foo/counter";

-(void)startWithURL:(NSURL *)url {
    NSLog(@"Connecting...");

    // Connect with control client credentials.
    PTDiffusionCredentials *credentials =
        [[PTDiffusionCredentials alloc]
initWithPassword:@"password"];
    PTDiffusionSessionConfiguration *sessionConfiguration =
        [[PTDiffusionSessionConfiguration alloc]
initWithPrincipal:@"principal"]
```

```

credentials:credentials];

    [PTDiffusionSession openWithURL:url
                        configuration:sessionConfiguration
                        completionHandler:^(PTDiffusionSession *session,
NSError *error)
    {
        if (!session) {
            NSLog(@"Failed to open session: %@", error);
            return;
        }

        // At this point we now have a connected session.
        NSLog(@"Connected.");

        // Maintain strong reference to session instance.
        self.session = session;

        // Send request to add topic for publishing to.
        [session.topicControl addWithTopicPath:_TopicPath

type:PTDiffusionTopicType_SingleValue
                                value:nil
                                completionHandler:^(NSError *error)
        {
            if (error) {
                NSLog(@"Failed to add topic: %@", error);
                return;
            }

            // At this point we now have a topic.
            [self updateCounter];
        }
    ]];
}

-(void)updateCounter {
    // Get the updater to be used for non-exclusive topic updates.
    PTDiffusionTopicUpdater *updater =
self.session.topicUpdateControl.updater;

    // Format string content for the update.
    NSString *string = [NSString stringWithFormat:@"%lu",
(unsigned long)_counter++];
    NSData *data = [string
dataUsingEncoding:NSUTF8StringEncoding];
    PTDiffusionContent *content = [[PTDiffusionContent alloc]
initWithData:data];

    // Send request to update topic.
    NSLog(@"Updating: %@", string);
    [updater updateWithTopicPath:_TopicPath
                        value:content
                        completionHandler:^(NSError *error)
    {
        if (error) {
            NSLog(@"Failed to update topic: %@", error);
        }
    }
    ]];

    // Schedule another update in one second's time.
    __weak CounterPublisher *const weakSelf = self;

```

```

        dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(1.0 *
NSEC_PER_SEC)),
                        dispatch_get_main_queue(), ^
        {
            [weakSelf updateCounter];
        });
    }
@end

```

Start publishing with Android

Create an Android client that publishes data through topics on the Diffusion server.

Before you begin

To complete this example, you need Android Studio installed on your development system and a Diffusion server.

This example was tested in Android Studio 2.3.3. If you are using a different version of Android Studio, the details of some steps may vary slightly.

You also require a named user that has a role with the `modify_topic` and `update_topic` permissions. For example, the “ADMINISTRATOR” role. For more information about roles and permissions, see [Role-based authorization](#) on page 130.

About this task

This example steps through the lines of code required to publish to a JSON topic. The full code example is provided after the steps.

Procedure

1. Set up a project in Android Studio that uses the Diffusion Unified API.
 - a) Create a new project using API Level 21 or later.
 - b) Copy the `diffusion-android-x.x.x.jar` into the `app/libs` folder of your project.
 - c) In Android Studio, right-click on the `libs` folder in the left-hand panel (the **Project Tool Window**), then select **Add as Library**.
If the `libs` folder is not shown in the left-hand panel, use the pull-down menu at the top of the panel to select **Project** view.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Insert the element between the opening `<manifest>` tag and the opening `<application>` tag. This permission is required to use the Diffusion API.

3. Open your project's `MainActivity.java` file.

This file is where you develop the code to interact with the Diffusion server.

The empty `MainActivity.java` file contains the following boilerplate code:

```

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

```

```

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
        }
    }
}

```

4. Import the following packages and classes:

```

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.callbacks.ErrorReason;
import com.pushtechology.diffusion.client.features.Topics;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicControl;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicUpdateControl;
import com.pushtechology.diffusion.client.session.Session;
import com.pushtechology.diffusion.client.session.SessionFactory;
import
    com.pushtechology.diffusion.client.topics.details.TopicType;
import com.pushtechology.diffusion.datatype.json.JSON;
import com.pushtechology.diffusion.datatype.json.JSONDataType;

import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicInteger;

public class MainActivity extends AppCompatActivity {
}

```

The `com.pushtechology.diffusion.client` packages contain the classes to use to interact with the Diffusion server.

5. Create a `SessionHandler` inner class that implements `SessionFactory.OpenCallback`. This inner class will contain the code that interacts with the Diffusion server.

```

    private class SessionHandler implements
        SessionFactory.OpenCallback {
        private Session session = null;

        @Override
        public void onOpened(Session session) {
            this.session = session;
        }

        @Override
        public void onError(ErrorReason errorReason) {

        }

        public void close() {
            if ( session != null ) {
                session.close();
            }
        }
    }
}

```

6. In the `onOpened` method, add the code required to create the `foo/counter` topic and update it with an incrementing value.

- a) Use the `TopicControl` feature to create a JSON topic.

```
// Create a JSON topic 'foo/counter'
session.feature(TopicControl.class).addTopic(
    "foo/counter",
    TopicType.JSON,
    new TopicControl.AddCallback.Default());
```

- b) Get the `TopicUpdateControl` feature and JSON data type.

```
// Get the TopicUpdateControl feature and JSON data
type
final JSONDataType jsonDataType =
Diffusion.dataTypes().json();
final TopicUpdateControl updateControl = session
.feature(TopicUpdateControl.class);
```

- c) Loop once a second updating the `foo/counter` topic with an incrementing count from 0 to 1000. Use the non-exclusive `updateControl.updater().update()` method to update the topic while still allowing other clients to update the topic.

```
final AtomicInteger i = new AtomicInteger(0);

Executors.newSingleThreadScheduledExecutor().scheduleAtFixedRate(new
Runnable() {
    @Override
    public void run() {
        // Create the json value
        final JSON value = jsonDataType.fromJsonString(
            String.format("{\"count\" : %d }",
i.getAndIncrement()));

        // Update the topic
        updateControl.updater().update(
            "counter",
            value,
            new
            TopicUpdateControl.Updater.UpdateCallback.Default());
    }
}, 1000, 1000, TimeUnit.MILLISECONDS);
```

7. In the `MainActivity` class, declare an instance of session handler.

```
private SessionHandler sessionHandler = null;
```

8. Override the `onCreate` method of the `MainActivity` class to open the session with the Diffusion server.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    if (sessionHandler == null) {
```

```

        sessionHandler = new SessionHandler();

        Diffusion.sessions()
            .principal("username")
            .password("password")
            .open("ws://host:port", sessionHandler);
    }
}

```

Or you can connect securely, using :

```

Diffusion.sessions().principal("principal").password("password").open("wss://
sessionHandler);

```

Replace the *host*, *port*, *principal*, and *password* values with your own information.

9. Override the `onDestroy` method of the `MainActivity` class to close the session with the Diffusion server.

```

        if ( sessionHandler != null ) {
            sessionHandler.close();
            sessionHandler = null;
        }
        super.onDestroy();

```

10. Compile and run your client.

Results

The client publishes a JSON value to the `foo/counter` topic every second. You can subscribe to the `foo/counter` topic by creating a client to subscribe to the topic. For more information, see [Start subscribing with Android](#) on page 183.

Full example

The completed `MainActivity` class contains the following code:

```

package com.pushtechology.demo.update;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.callbacks.ErrorReason;
import com.pushtechology.diffusion.client.features.Topics;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicControl;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicUpdateControl;
import com.pushtechology.diffusion.client.session.Session;
import com.pushtechology.diffusion.client.session.SessionFactory;
import
    com.pushtechology.diffusion.client.topics.details.TopicType;
import com.pushtechology.diffusion.datatype.json.JSON;
import com.pushtechology.diffusion.datatype.json.JSONDataType;

import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicInteger;

```

```

public class MainActivity extends AppCompatActivity {

    /**
     * A session handler that maintains the diffusion session.
     */
    private class SessionHandler implements
    SessionFactory.OpenCallback {
        private Session session = null;

        @Override
        public void onOpened(Session session) {
            this.session = session;

            // Create a JSON topic 'foo/counter'
            session.feature(TopicControl.class).addTopic(
                "foo/counter",
                TopicType.JSON,
                new TopicControl.AddCallback.Default());

            // Get the TopicUpdateControl feature and JSON data
            type
            final JSONDataType jsonDataType =
            Diffusion.dataTypes().json();
            final TopicUpdateControl updateControl = session
                .feature(TopicUpdateControl.class);

            final AtomicInteger i = new AtomicInteger(0);

            // Schedule a recurring task that increments the
            counter and updates the "counter" topic with a json value
            // every second

            Executors.newSingleThreadScheduledExecutor().scheduleAtFixedRate(new
            Runnable() {
                @Override
                public void run() {
                    // Create the json value
                    final JSON value =
                    jsonDataType.fromJsonString(
                        String.format("{\"count\" : %d }",
                    i.getAndIncrement()));

                    // Update the topic
                    updateControl.updater().update(
                        "counter",
                        value,
                        new
                    TopicUpdateControl.Updater.UpdateCallback.Default());
                }
            }, 1000, 1000, TimeUnit.MILLISECONDS);
        }

        @Override
        public void onError(ErrorReason errorReason) {
            Log.e("Diffusion", "Failed to open session because: "
            + errorReason.toString());
            session = null;
        }

        public void close() {

```

```

        if (session != null) {
            session.close();
        }
    }

    private SessionHandler sessionHandler = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        if (sessionHandler == null) {
            sessionHandler = new SessionHandler();

            Diffusion.sessions()
                .principal("username")
                .password("password")
                .open("ws://host:port", sessionHandler);
        }
    }

    @Override
    protected void onDestroy() {
        if (sessionHandler != null) {
            sessionHandler.close();
            sessionHandler = null;
        }

        super.onDestroy();
    }
}

```

Related information

<http://developer.android.com/training/index.html>

Start publishing with Java

Create a Java client that publishes data through topics on the Diffusion server.

Before you begin

To complete this example, you need a Diffusion server and a development system with Java installed on it.

You also require a principal that has a role with the `modify_topic` and `update_topic` permissions. For example, the “ADMINISTRATOR” role. For more information about roles and permissions, see [Role-based authorization](#) on page 130.

About this task

This example steps through the lines of code required to publish a value to a JSON topic. The full code example is provided after the steps.

Procedure

1. Include the client jar file on the build classpath of your Java client. You can use one of the following methods:

- You can use Maven to declare the dependency. First add the Push Technology public repository to your `pom.xml` file:

```
<repositories>
  <repository>
    <id>push-repository</id>
    <url>https://download.pushtechnology.com/maven/</url>
  </repository>
</repositories>
```

Next declare the following dependency in your `pom.xml` file:

```
<dependencies>
  <dependency>
    <groupId>com.pushtechnology.diffusion</groupId>
    <artifactId>diffusion-client</artifactId>
    <version>version</version>
  </dependency>
</dependencies>
```

Where *version* is the Diffusion version, for example 5.9.24.

- If you are not using Maven, you can include the `diffusion-client.jar` file that is located in the `clients/java` directory of your Diffusion server installation.
2. Create a `PublishingClient` class that imports the following packages and classes:

```
import com.pushtechnology.diffusion.client.Diffusion;
import
  com.pushtechnology.diffusion.client.features.control.topics.TopicControl;
import
  com.pushtechnology.diffusion.client.features.control.topics.TopicControl.AddC
import
  com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateControl;
import
  com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateControl;
import com.pushtechnology.diffusion.client.session.Session;
import
  com.pushtechnology.diffusion.client.topics.details.TopicType;
import com.pushtechnology.diffusion.datatype.json.JSON;
import com.pushtechnology.diffusion.datatype.json.JSONDataType;

import java.util.concurrent.CountDownLatch;

public final class PublishingClient {
}
}
```

The `com.pushtechnology.diffusion.client` packages contain the classes to use to interact with the Diffusion server. The `java.util.concurrent.CountDownLatch` class is used to simplify this example by making it more synchronous. However, the Diffusion API is designed to be most powerful when used asynchronously.

3. Create a main method.

```
public final class PublishingClient {
```

```

        public static void main(String... arguments) throws
        InterruptedException {
            }
        }
    }
}

```

4. Inside the `main` method, connect to the Diffusion server.

Make sure to edit the `host` value in the example to match your Diffusion server's hostname or IP address.

The principal you use to connect must have `modify_topic` and `update_topic` permissions. In this example, we use the default `control` principal.

```

        final Session session =
        Diffusion.sessions().principal("control")
        .password("password").open("ws://host:8080");
    }
}

```

Or you can connect securely to the Diffusion server using :

```

        .open("wss://host:port");
    }
}

```

Replace the `host`, `port`, `principal`, and `password` values with your own information.

You can choose to connect anonymously if anonymous sessions are assigned the `modify_topic` and `update_topic` permissions. However, we do not recommend that anonymous sessions are given write access to data on the Diffusion server.

5. Next, in the `main` method, get the `TopicControl` and `TopicUpdateControl` features.

```

        // Get the TopicControl and TopicUpdateControl feature
        TopicControl topicControl =
        session.feature(TopicControl.class);

        TopicUpdateControl updateControl = session
        .feature(TopicUpdateControl.class);
    }
}

```

The `TopicControl` feature enables a client to create and delete topics. For more information, see [Managing topics](#) on page 295.

The `TopicUpdateControl` feature enables a client to publish updates to a topic. For more information, see [Updating topics](#) on page 331.

6. Next, create an instance of `JSONDataType` to store the value of the JSON topic.

```

        final JSONDataType jsonDataType =
        Diffusion.dataTypes().json();
    }
}

```

7. Next, in the `main` method, use the `TopicControl` feature to create the `foo/counter` topic.

```

        final CountdownLatch waitForStart = new CountdownLatch(1);

        // Create a JSON topic 'foo/counter'
        topicControl.addTopic(
            "foo/counter",
            TopicType.JSON,
            new AddCallback.Default() {
                @Override
                public void onTopicAdded(String topicPath) {
                    waitForStart.countDown();
                }
            }
        );
    }
}

```

```

    });

    // Wait for the onTopicAdded() callback.
    waitForStart.await();

```

This example uses a `CountDownLatch` to wait until the topic is successfully added. This approach is used to simplify the example and is not recommended for production clients.

8. Next, in the `main` method, loop once a second updating the `foo/counter` topic with an incrementing count from 0 to 1000.

Use the `updateControl.updater().update()` method to update a topic without locking that topic.

```

        // Update the topic
        final UpdateCallback updateCallback = new
UpdateCallback.Default();
        for (int i = 0; i < 1000; ++i) {
            final JSON value =
jsonDataType.fromJsonString(String.format("{\"count\" : %d }",
i));
            // Use the non-exclusive updater to update the topic
without locking it
            updateControl.updater().update(
                "foo/counter",
                Integer.toString(i),
                updateCallback);

            Thread.sleep(1000);
        }

```

9. Compile and run your client.

We recommend that you run your client using the JDK rather than the JRE. The JDK includes additional diagnostic capabilities that might be useful.

Results

The client publishes a value to the `foo/counter` topic every second. You can subscribe to the `foo/counter` topic by creating a client to subscribe to the topic. For more information, see [Start subscribing with Java](#) on page 188.

Full example

The completed `PublishingClient` class contains the following code:

```

import com.pushtechology.diffusion.client.Diffusion;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicControl;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicControl.AddC
import
    com.pushtechology.diffusion.client.features.control.topics.TopicUpdateContro
import
    com.pushtechology.diffusion.client.features.control.topics.TopicUpdateContro
import com.pushtechology.diffusion.client.session.Session;
import
    com.pushtechology.diffusion.client.topics.details.TopicType;
import com.pushtechology.diffusion.datatype.json.JSON;
import com.pushtechology.diffusion.datatype.json.JSONDataType;

import java.util.concurrent.CountDownLatch;

```

```

/**
 * A client that publishes an incrementing count to the topic
 * 'foo/counter'.
 *
 * @author Push Technology Limited
 * @since 5.9
 */
public final class PublishingClient {

    /**
     * Main.
     */
    public static void main(String... arguments) throws
    InterruptedException {

        // Connect using a principal with 'modify_topic' and
        'update_topic'
        // permissions
        // Change 'host' to the hostname/address of your Diffusion
        server
        final Session session =
        Diffusion.sessions().principal("control")
            .password("password").open("ws://host:port");

        // Get the TopicControl and TopicUpdateControl feature
        final TopicControl topicControl =
        session.feature(TopicControl.class);

        final TopicUpdateControl updateControl =
        session.feature(TopicUpdateControl.class);

        final JSONDataType jsonDataType =
        Diffusion.dataTypes().json();

        final CountdownLatch waitForStart = new CountdownLatch(1);

        // Create a JSON topic 'foo/counter'
        topicControl.addTopic(
            "foo/counter",
            TopicType.JSON,
            new AddCallback.Default() {
                @Override
                public void onTopicAdded(String topicPath) {
                    waitForStart.countDown();
                }
            }
        ));

        // Wait for the onTopicAdded() callback.
        waitForStart.await();

        // Update the topic
        final UpdateCallback updateCallback = new
        UpdateCallback.Default();
        for (int i = 0; i < 1000; ++i) {
            final JSON value =
            jsonDataType.fromJsonString(String.format("{\\"count\\" : %d }",
            i));
            // Use the non-exclusive updater to update the topic
            without locking it
            updateControl.updater().update(
                "foo/counter",
                value,
                updateCallback);
        }
    }
}

```

```
        Thread.sleep(1000);
    }
}
```

Start publishing with .NET

Create a .NET client that publishes data through topics on the Diffusion server.

Before you begin

To complete this example, you need a Diffusion server and a development system with the .NET Framework installed on it.

You also require either a named user that has a role with the `modify_topic` and `update_topic` permissions. For example, the “ADMINISTRATOR” role. For more information about roles and permissions, see [Role-based authorization](#) on page 130.

About this task

This example steps through the lines of code required to subscribe to a JSON topic. The full code example is provided after the steps.

Procedure

1. Create a .NET project that references the following DLL file located in the `clients/dotnet` directory of your .NET installation:

PushTechnology.ClientInterface.dll

The assembly contains the interfaces classes and interfaces that you use when creating a control client.

Use the following command in the NuGet Package Manager Console:

```
PM> Install-Package PushTechnology.UnifiedClientInterface
```

2. In your project, create a C# file that uses the following packages:

```
using System;
using System.Threading;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Factories;
using
    PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Data.JSON;

namespace PushTechnology.ClientInterface.GettingStarted {
}
```

3. Create a Main method.

```
public sealed class PublishingClient
{
    public static void Main( string[] args ) {
    }
}
```

```
}
```

4. Inside the `Main` method, connect to the Diffusion server.

```
public static void Main( string[] args ) {  
    // Connect using a principal with 'modify_topic' and  
'update_topic'  
    // permissions  
    var session =  
Diffusion.Sessions.Principal( "principal" ).Password( "password" ).Open( "ws://hos  
}
```

Or you can connect securely to the Diffusion server using :

```
.Open( "wss://host:port" );
```

Replace the *host*, *port*, *principal*, and *password* values with your own information.

You can choose to connect anonymously if anonymous sessions are assigned the `modify_topic` and `update_topic` permissions. However, we do not recommend that anonymous sessions are given write access to data on the Diffusion server.

5. Next, in the `Main` method, get the `TopicControl` and `TopicUpdateControl` features.

```
// Get the TopicControl and TopicUpdateControl features  
var topicControl = session.GetTopicControlFeature();  
var updateControl =  
session.GetTopicUpdateControlFeature();
```

The `TopicControl` feature enables a client to create and delete topics. For more information, see .

The `TopicUpdateControl` feature enables a client to publish updates to a topic. For more information, see .

6. Next, in the `Main` method, use the `TopicControl` feature to create the `foo/counter` JSON topic.

```
// Create a JSON topic 'foo/counter'  
var topic = "foo/counter";  
var addCallback = new AddCallback();  
topicControl.AddTopicFromValue(  
    topic,  
    Diffusion.DataTypes.JSON.FromJSONString( "{ \"date  
\": \"To be updated\", \"time\": \"To be updated\" }" ),  
    addCallback );  
  
// Wait for the OnTopicAdded callback, or a failure  
if ( !  
addCallback.Wait( TimeSpan.FromSeconds( 5 ) ) ) {  
    Console.WriteLine( "Callback not received  
within timeout." );  
    session.Close();  
    return;  
} else if ( addCallback.Error != null ) {  
    Console.WriteLine( "Error : {0}",  
addCallback.Error.ToString() );  
    session.Close();  
    return;  
}  
}
```

The `AddTopicFromValue()` method requires a callback, which you must create.

7. Implement an instance of `ITopicControlAddCallback` as an internal sealed class.

```
internal sealed class AddCallback : ITopicControlAddCallback {
```

```

        private readonly AutoResetEvent resetEvent = new
AutoResetEvent( false );

        /// <summary>
        /// Any error from this AddCallback will be stored here.
        /// </summary>
        public Exception Error {
            get;
            private set;
        }

        /// <summary>
        /// Constructor.
        /// </summary>
        public AddCallback() {
            Error = null;
        }

        /// <summary>
        /// This is called to notify that a call context was closed
prematurely, typically due to a timeout or the
        /// session being closed.
        /// </summary>
        /// <remarks>
        /// No further calls will be made for the context.
        /// </remarks>
        public void OnDiscard() {
            Error = new Exception( "This context was closed
prematurely." );
            resetEvent.Set();
        }

        /// <summary>
        /// This is called to notify that the topic has been added.
        /// </summary>
        /// <param name="topicPath">The full path of the topic that
was added.</param>
        public void OnTopicAdded( string topicPath ) {
            Console.WriteLine( "Topic {0} added.", topicPath );
            resetEvent.Set();
        }

        /// <summary>
        /// This is called to notify that an attempt to add a topic
has failed.
        /// </summary>
        /// <param name="topicPath">The topic path as supplied to
the add request.</param>
        /// <param name="reason">The reason for failure.</param>
        public void OnTopicAddFailed( string topicPath,
TopicAddFailReason reason ) {
            Error = new Exception( string.Format( "Failed to add
topic {0} : {1}", topicPath, reason ) );
            resetEvent.Set();
        }

        /// <summary>
        /// Wait for one of the callbacks for a given time.
        /// </summary>
        /// <param name="timeout">Time to wait for the callback.</
param>
        /// <returns><c>>true</c> if either of the callbacks has
been triggered. Otherwise <c>>false</c>.</returns>

```

```

        public bool Wait( TimeSpan timeout ) {
            return resetEvent.WaitOne( timeout );
        }
    }

```

8. Back in the `Main` method, loop once a second updating the `foo/counter` topic with an incrementing count.

Use the `updateControl.Updater.Update()` method to update a topic without locking that topic.

```

        var updateCallback = new UpdateCallback( topic );
        for ( var i = 0; i < 1000; ++i ) {
            updateControl.Updater.Update( topic, i.ToString(),
            updateCallback );

            Thread.Sleep( 1000 );
        }

```

The `Update()` method requires a callback, which you must create.

9. Implement an instance of `ITopicUpdaterUpdateCallback` as an internal sealed class.

```

        internal sealed class UpdateCallback :
        ITopicUpdaterUpdateCallback {
            private readonly string topicPath;

            /// <summary>
            /// Constructor.
            /// </summary>
            /// <param name="topicPath">The topic path.</param>
            public UpdateCallback( string topicPath ) {
                this.topicPath = topicPath;
            }

            /// <summary>
            /// Notification of a contextual error related to
            this callback.
            /// </summary>
            /// <remarks>
            /// Situations in which <code>OnError</code> is
            called include the session being closed, a communication
            parameters. No further calls will be made to this callback.
            /// </remarks>
            /// <param name="errorReason">A value representing
            the error.</param>
            public void OnError( ErrorReason errorReason ) {
                Console.WriteLine( "Topic {0} could not be
            updated : {1}", topicPath, errorReason );
            }

            /// <summary>
            /// Indicates a successful update.
            /// </summary>
            public void OnSuccess() {
                Console.WriteLine( "Topic {0} updated
            successfully.", topicPath );
            }
        }

```

10. Finally, in the `Main` method, close the session between your client and the Diffusion server.

```
// Close session
session.Close();
```

11. Compile and run your client.

Results

The client publishes a value to the `foo/counter` topic every second. You can subscribe to the `foo/counter` topic by creating a client to subscribe to the topic. For more information, see [Start subscribing with .NET](#) on page 192.

Full example

The completed publishing client class contains the following code:

```
using System;
using System.Threading;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Factories;
using
    PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Data.JSON;

namespace PushTechnology.ClientInterface.GettingStarted {
    /// <summary>
    /// A client that publishes an incrementing count to the JSON
    topic "foo/counter".
    /// </summary>
    public sealed class PublishingClient {
        public static void Main( string[] args ) {
            // Connect using a principal with 'modify_topic' and
            'update_topic' permissions
            var session =
                Diffusion.Sessions.Principal( "control" ).Password( "password" ).Open( "ws://
                localhost:8080" );

            // Get the TopicControl and TopicUpdateControl
            features
                var topicControl = session.GetTopicControlFeature();
                var updateControl =
                    session.GetTopicUpdateControlFeature();

            // Create a JSON topic 'foo/counter'
            var topic = "foo/counter";
            var addCallback = new AddCallback();
            topicControl.AddTopicFromValue(
                topic,
                Diffusion.DataTypes.JSON.FromJSONString( "{ \"date
                \": \"To be updated\", \"time\": \"To be updated\" }" ),
                addCallback );

            // Wait for the OnTopicAdded callback, or a failure
            if ( !addCallback.Wait( TimeSpan.FromSeconds( 5 ) ) )
            {
                Console.WriteLine( "Callback not received within
                timeout." );
                session.Close();
                return;
            } else if ( addCallback.Error != null ) {
```

```

        Console.WriteLine( "Error : {0}",
addCallback.Error.ToString() );
        session.Close();
        return;
    }

    // Update topic every 300 ms for 30 minutes
    var updateCallback = new UpdateCallback( topic );
    for ( var i = 0; i < 3600; ++i ) {
        var newValue =
Diffusion.DataTypes.JSON.FromJSONString(
        "{ \"date\": \" " +
DateTime.Today.Date.ToString( "D" ) + "\", " +
        "\"time\": \" " +
DateTime.Now.TimeOfDay.ToString( "g" ) + "\" }" );

updateControl.Updater.ValueUpdater<IJSON>().Update( topic,
newValue, updateCallback );

        Thread.Sleep( 300 );
    }

    // Close session
    session.Close();
}

/// <summary>
/// Basic implementation of the ITopicControlAddCallback.
/// </summary>
internal sealed class AddCallback : ITopicControlAddCallback {
    private readonly AutoResetEvent resetEvent = new
AutoResetEvent( false );

    /// <summary>
    /// Any error from this AddCallback will be stored here.
    /// </summary>
    public Exception Error {
        get;
        private set;
    }

    /// <summary>
    /// Constructor.
    /// </summary>
    public AddCallback() {
        Error = null;
    }

    /// <summary>
    /// This is called to notify that a call context was
closed prematurely, typically due to a timeout or the
    /// session being closed.
    /// </summary>
    /// <remarks>
    /// No further calls will be made for the context.
    /// </remarks>
    public void OnDiscard() {
        Error = new Exception( "This context was closed
prematurely." );
        resetEvent.Set();
    }
}

```

```

        /// <summary>
        /// This is called to notify that the topic has been
added.
        /// </summary>
        /// <param name="topicPath">The full path of the topic
that was added.</param>
        public void OnTopicAdded( string topicPath ) {
            Console.WriteLine( "Topic {0} added.", topicPath );
            resetEvent.Set();
        }

        /// <summary>
        /// This is called to notify that an attempt to add a
topic has failed.
        /// </summary>
        /// <param name="topicPath">The topic path as supplied to
the add request.</param>
        /// <param name="reason">The reason for failure.</param>
        public void OnTopicAddFailed( string topicPath,
TopicAddFailReason reason ) {
            Error = new Exception( string.Format( "Failed to add
topic {0} : {1}", topicPath, reason ) );
            resetEvent.Set();
        }

        /// <summary>
        /// Wait for one of the callbacks for a given time.
        /// </summary>
        /// <param name="timeout">Time to wait for the callback.</
param>
        /// <returns><c>>true</c> if either of the callbacks has
been triggered. Otherwise <c>>false</c>.</returns>
        public bool Wait( TimeSpan timeout ) {
            return resetEvent.WaitOne( timeout );
        }
    }

    /// <summary>
    /// A simple ITopicUpdaterUpdateCallback implementation that
prints confirmation of the actions completed.
    /// </summary>
    internal sealed class UpdateCallback :
ITopicUpdaterUpdateCallback {
        private readonly string topicPath;

        /// <summary>
        /// Constructor.
        /// </summary>
        /// <param name="topicPath">The topic path.</param>
        public UpdateCallback( string topicPath ) {
            this.topicPath = topicPath;
        }

        /// <summary>
        /// Notification of a contextual error related to this
callback.
        /// </summary>
        /// <remarks>
        /// Situations in which <code>OnError</code> is called
include the session being closed, a communication
        /// timeout, or a problem with the provided parameters. No
further calls will be made to this callback.
        /// </remarks>

```

```

        /// <param name="errorReason">A value representing the
        error.</param>
        public void OnError( ErrorReason errorReason ) {
            Console.WriteLine( "Topic {0} could not be updated :
{1}", topicPath, errorReason );
        }

        /// <summary>
        /// Indicates a successful update.
        /// </summary>
        public void OnSuccess() {
            Console.WriteLine( "Topic {0} updated successfully.",
topicPath );
        }
    }
}

```

Start publishing with C

Create a C client that publishes data through topics on the Diffusion server.

Before you begin

The C client libraries rely on a number of dependencies. Depending on which platform you are using the C client libraries for, these dependencies might be included in the client library. If they are not included in the client library, ensure that the dependencies are available on your development system.

For more information about dependencies on each supported platform, see [C](#) on page 164.

The C client library statically links to APR version 1.5 with APR-util. Ensure that you set `APR_DECLARE_STATIC` and `APU_DECLARE_STATIC` before you use any APR includes. You can set these values in the following ways:

- By including `diffusion.h` before any APR includes. The `diffusion.h` file sets these values.
- As command-line flags

For more information, see <http://apr.apache.org>

To complete this example, you need a Diffusion server and a development system with the .NET Framework installed on it.

You also require either a named user that has a role with the `modify_topic` and `update_topic` permissions. For example, the “ADMINISTRATOR” role. For more information about roles and permissions, see [Role-based authorization](#) on page 130.

About this task

This example steps through the lines of code required to create and update a topic. The [full code example](#) is provided after the steps.

Procedure

1. Get the Diffusion C client library for your platform and extract the ZIP file.
The C client library is available in the `clients/c` directory of the Diffusion installation.
2. Create a C file called `getting-started-publisher.c`.
 - a) Include the following libraries:

```

#include <stdio.h>
#include <stdlib.h>

```

```

#include <unistd.h>

#include <apr.h>
#include <apr_thread_mutex.h>
#include <apr_thread_cond.h>

#include "diffusion.h"
#include "args.h"
#include "conversation.h"
#include "service/svc-update.h"

```

- b) Declare an int value that and APR objects to use to manage threading:

```

int active = 0;

apr_pool_t *pool = NULL;
apr_thread_mutex_t *mutex = NULL;
apr_thread_cond_t *cond = NULL;

```

- c) Create a main method:

```

int
main(int argc, char **argv)
{
}

```

- d) Inside the main method define a constant that is the topic name:

```

const char *topic_name = "foo/counter";

```

- e) Next, add the following lines to initialize the threading mechanism:

```

apr_initialize();
apr_pool_create(&pool, NULL);
apr_thread_mutex_create(&mutex,
APR_THREAD_MUTEX_UNNESTED, pool);
apr_thread_cond_create(&, pool);

```

3. Create a connection to the Diffusion server.

Inside the main method add the following lines:

```

/*
 * Create a session
 */
DIFFUSION_ERROR_T error = { 0 };
SESSION_T *session = NULL;
// Edit this line to include the host and port of your
Diffusion server
session = session_create("ws://hostname:port", "user",
credentials_create_password("password"), NULL, NULL, &error);
if(session == NULL) {
    fprintf(stderr, "TEST: Failed to create session
\n");
    fprintf(stderr, "ERR : %s\n", error.message);
    return EXIT_FAILURE;
} else {
    fprintf(stdout, "Connected\n");
}

```

Where *hostname* is the name of the system hosting your Diffusion server, *hostname* is the name the Diffusion server accepts client connections on, *user* is the name of a user with the permissions required to subscribe to a topic, and *password* is the user's password.

The client logs the string "Connected" to the console if the connection is a success.

4. Create the `foo/counter` topic.

- a) Above the `main` method, create a callback for when the topic is added.

```
static int
on_topic_added(SESSION_T *session, const
    SVC_ADD_TOPIC_RESPONSE_T *response, void *context)
{
    printf("Added topic\n");
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}
```

This callback prints a message to the console when the `foo/counter` topic is created.

- b) Above the `main` method, create a callback for if the topic add fails.

```
static int
on_topic_add_failed(SESSION_T *session, const
    SVC_ADD_TOPIC_RESPONSE_T *response, void *context)
{
    printf("Failed to add topic (%d)\n", response-
>response_code);
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}
```

This callback prints a message to the console if the client was unable to create the `foo/counter` topic.

- c) In the `main` method, define the topic details:

```
const TOPIC_DETAILS_T *string_topic_details =
    create_topic_details_single_value(M_DATA_TYPE_STRING);
```

This defines the topic type as a single value topic of data type string.

- d) Define the parameters for the add topic request:

```
const ADD_TOPIC_PARAMS_T add_topic_params = {
    .topic_path = topic_name,
    .details = string_topic_details,
    .on_topic_added = on_topic_added,
    .on_topic_add_failed = on_topic_add_failed
};
```

- e) Within a locked thread, call the `add_topic` method:

```
apr_thread_mutex_lock(mutex);
add_topic(session, add_topic_params);
apr_thread_cond_wait(cond, mutex);
apr_thread_mutex_unlock(mutex);
```

5. Register an update source.

- a) Above the main method, create a callback for when the update source is registered.

```
static int
on_update_source_registered(SESSION_T *session,
                           const CONVERSATION_ID_T *updater_id,
                           const
SVC_UPDATE_REGISTRATION_RESPONSE_T *response,
                           void *context)
{
    printf("Registered update source\n");
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}
```

This callback prints a message to the console when the update source is registered.

- b) Above the main method, create a callback for when the update source becomes active.

```
static int
on_update_source_active(SESSION_T *session,
                       const CONVERSATION_ID_T *updater_id,
                       const SVC_UPDATE_REGISTRATION_RESPONSE_T
*response,
                       void *context)
{
    printf("Topic source active\n");
    active = 1;
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}
```

This callback prints a message to the console when the update source becomes the active update source and can update the topic exclusively. The callback also sets the active flag to 1.

- c) In the main method, define the parameters for registering the update source:

```
const UPDATE_SOURCE_REGISTRATION_PARAMS_T
update_reg_params = {
    .topic_path = topic_name,
    .on_registered = on_update_source_registered,
    .on_active = on_update_source_active,
};
```

- d) Within a locked thread, call the `register_update_source` method:

```
apr_thread_mutex_lock(mutex);
const CONVERSATION_ID_T *updater_id =
register_update_source(session, update_reg_params);
apr_thread_cond_wait(cond, mutex);
apr_thread_mutex_unlock(mutex);
```

Registering an update source returns an updater ID that can be used to update topics.

6. Send updates to the topic.

- a) Above the main method, create a callback for when a topic update is a success.

```
static int
```

```

on_update_success(SESSION_T *session,
                  const CONVERSATION_ID_T *updater_id,
                  const SVC_UPDATE_RESPONSE_T *response,
                  void *context)
{
    printf("Updated topic\n");
    return HANDLER_SUCCESS;
}

```

This callback prints a message to the console when the foo/counter topic is successfully updated.

- b) Above the main method, create a callback for if a topic update fails.

```

static int
on_update_failure(SESSION_T *session,
                  const CONVERSATION_ID_T *updater_id,
                  const SVC_UPDATE_RESPONSE_T *response,
                  void *context)
{
    printf("Update failed\n");
    return HANDLER_SUCCESS;
}

```

This callback prints a message to the console if an update of the foo/counter topic fails.

- c) Define the unchanging parameters for updating a topic using the update source.

```

UPDATE_SOURCE_PARAMS_T update_source_params_base = {
    .updater_id = updater_id,
    .topic_path = topic_name,
    .on_success = on_update_success,
    .on_failure = on_update_failure
};

```

- d) Define a variable count which is used for the value published to the topic:

```
int count=1;
```

- e) Create a loop that runs while the update source is active:

```
while(active) {
}

```

- f) Inside the loop, create a buffer to contain the counter value:

```

BUF_T *buf = buf_create();
char str[15];
sprintf(str, "%d", count);
buf_write_string(buf, str);

```

- g) Next, use the buffer to create content:

```

CONTENT_T *content =
content_create(CONTENT_ENCODING_NONE, buf);

```

- h) Create an update from the content:

```

UPDATE_T *upd =
update_create(UPDATE_ACTION_REFRESH,
UPDATE_TYPE_CONTENT,

```

```
content);
```

- i) Add the update into the parameters defined for the update.

```
UPDATE_SOURCE_PARAMS_T update_source_params =  
update_source_params_base;  
update_source_params.update = upd;
```

- j) Update the topic.

```
update(session, update_source_params);
```

- k) Free the resources used by this iteration of the loop.

```
content_free(content);  
update_free(upd);  
buf_free(buf);
```

- l) Wait for a second and increment the count variable before the next iteration of the loop:

```
sleep(1);  
count++;
```

7. Close the session with the Diffusion server and close the client.

```
session_close(session, NULL);  
session_free(session);  
  
apr_thread_mutex_destroy(mutex);  
apr_thread_cond_destroy(cond);  
apr_pool_destroy(pool);  
apr_terminate();  
  
return EXIT_SUCCESS;
```

Ensure that you free all resources and destroy the threading objects.

8. Build your C client.

- a) Create a Makefile in the same directory as your C file.

An [example Makefile](#) is provided after the steps.

- b) Ensure that your Makefile links to the include and lib directory of the Diffusion C library.

```
DIFFUSION_C_CLIENT_INCDIR = ../path-to-client/include  
DIFFUSION_C_CLIENT_LIBDIR = ../path-to-client/lib
```

- c) Run the make command to build the example.

The `getting-started-publisher` binary is created in the `target/bin` directory.

9. Run your C client from the command line.

Results

The client updates the value of the `foo/counter` topic. You can see the value of the `foo/counter` topic by creating a subscribing client to subscribe to the topic. For more information, see [Start subscribing with C](#) on page 197.

Example

The completed `getting-started-publisher.c` file contains the following code:

```
#include <stdio.h>
```

```

#include <stdlib.h>
#include <unistd.h>

#include <apr.h>
#include <apr_thread_mutex.h>
#include <apr_thread_cond.h>

#include "diffusion.h"
#include "args.h"
#include "conversation.h"
#include "service/svc-update.h"

int active = 0;

apr_pool_t *pool = NULL;
apr_thread_mutex_t *mutex = NULL;
apr_thread_cond_t *cond = NULL;

/*
 * Handlers for add topic feature.
 */
static int
on_topic_added(SESSION_T *session, const SVC_ADD_TOPIC_RESPONSE_T
 *response, void *context)
{
    printf("Added topic\n");
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}

static int
on_topic_add_failed(SESSION_T *session, const
 SVC_ADD_TOPIC_RESPONSE_T *response, void *context)
{
    printf("Failed to add topic (%d)\n", response-
>response_code);
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}

/*
 * Handlers for creating update source
 */
static int
on_update_source_registered(SESSION_T *session,
                           const CONVERSATION_ID_T *updater_id,
                           const
 SVC_UPDATE_REGISTRATION_RESPONSE_T *response,
                           void *context)
{
    printf("Registered update source\n");
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}

static int

```

```

on_update_source_active(SESSION_T *session,
                        const CONVERSATION_ID_T *updater_id,
                        const SVC_UPDATE_REGISTRATION_RESPONSE_T
*response,
                        void *context)
{
    printf("Topic source active\n");
    active = 1;
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}

/*
 * Handlers for update of data.
 */
static int
on_update_success(SESSION_T *session,
                  const CONVERSATION_ID_T *updater_id,
                  const SVC_UPDATE_RESPONSE_T *response,
                  void *context)
{
    printf("Updated topic\n");
    return HANDLER_SUCCESS;
}

static int
on_update_failure(SESSION_T *session,
                  const CONVERSATION_ID_T *updater_id,
                  const SVC_UPDATE_RESPONSE_T *response,
                  void *context)
{
    printf("Update failed\n");
    return HANDLER_SUCCESS;
}

int
main(int argc, char** argv)
{
    const char *topic_name = "foo/counter";

    /*
     * Setup for condition variable.
     */
    apr_initialize();
    apr_pool_create(&pool, NULL);
    apr_thread_mutex_create(&mutex, APR_THREAD_MUTEX_UNNESTED,
pool);
    apr_thread_cond_create(&, pool);

    /*
     * Create a session
     */
    SESSION_T *session;
    DIFFUSION_ERROR_T error = { 0 };

    // Edit this line to include the host and port of your
Diffusion server

```

```

        session = session_create("ws://hostname:port", "user",
credentials_create_password("password"), NULL, NULL, &error);
        if(session == NULL) {
            fprintf(stderr, "TEST: Failed to create session
\n");
            fprintf(stderr, "ERR : %s\n", error.message);
            return EXIT_FAILURE;
        } else {
            fprintf(stdout, "Connected\n");
        }

        /*
        * Create a topic holding simple string content.
        */
        const TOPIC_DETAILS_T *string_topic_details =
create_topic_details_single_value(M_DATA_TYPE_STRING);
        const ADD_TOPIC_PARAMS_T add_topic_params = {
            .topic_path = "foo/counter",
            .details = string_topic_details,
            .on_topic_added = on_topic_added,
            .on_topic_add_failed = on_topic_add_failed
        };

        apr_thread_mutex_lock(mutex);
        add_topic(session, add_topic_params);
        apr_thread_cond_wait(cond, mutex);
        apr_thread_mutex_unlock(mutex);

    /*
        * Define the handlers for add_update_source()
        */
        const UPDATE_SOURCE_REGISTRATION_PARAMS_T
update_reg_params = {
            .topic_path = topic_name,
            .on_registered = on_update_source_registered,
            .on_active = on_update_source_active,
        };

        /*
        * Register an updater.
        */
        apr_thread_mutex_lock(mutex);
        const CONVERSATION_ID_T *updater_id =
register_update_source(session, update_reg_params);
        apr_thread_cond_wait(cond, mutex);
        apr_thread_mutex_unlock(mutex);

        /*
        * Define default parameters for an update source.
        */
        UPDATE_SOURCE_PARAMS_T update_source_params_base = {
            .updater_id = updater_id,
            .topic_path = topic_name,
            .on_success = on_update_success,
            .on_failure = on_update_failure
        };

        int count=1;
        while(active) {

            /*

```

```

        * Create an update structure containing the
counter.
        */
        BUF_T *buf = buf_create();
        char str[15];
        sprintf(str, "%d", count);
        buf_write_string(buf, str);

        CONTENT_T *content =
content_create(CONTENT_ENCODING_NONE, buf);

        UPDATE_T *upd =
update_create(UPDATE_ACTION_REFRESH,
                                                    UPDATE_TYPE_CONTENT,
                                                    content);

        UPDATE_SOURCE_PARAMS_T update_source_params =
update_source_params_base;
        update_source_params.update = upd;

        /*
        * Update the topic.
        */
        update(session, update_source_params);

        content_free(content);
        update_free(upd);
        buf_free(buf);

        sleep(1);
        count++;
    }

    /*
    * Close session and free resources.
    */
    session_close(session, NULL);
    session_free(session);

    apr_thread_mutex_destroy(mutex);
    apr_thread_cond_destroy(cond);
    apr_pool_destroy(pool);
    apr_terminate();

    return EXIT_SUCCESS;
}

```

The Makefile contains the following code:

```

# The following two variables must be set.
#
# Directory containing the C client include files.
DIFFUSION_C_CLIENT_INCDIR = ../path-to-client/include
#
# Directory containing libdiffusion.a
DIFFUSION_C_CLIENT_LIBDIR = ../path-to-client/lib

ifndef DIFFUSION_C_CLIENT_INCDIR
$(error DIFFUSION_C_CLIENT_INCDIR is not set)
endif

ifndef DIFFUSION_C_CLIENT_LIBDIR

```

```

$(error DIFFUSION_C_CLIENT_LIBDIR is not set)
endif

CC = gcc

# Extra definitions from parent directory, if they exist.
-include ../makefile.defs

CFLAGS += -g -Wall -Werror -std=c99 -D_POSIX_C_SOURCE=200112L -
D_XOPEN_SOURCE=700 -c -I$(DIFFUSION_C_CLIENT_INCDIR)
LDFLAGS += $(LIBS) $(DIFFUSION_C_CLIENT_LIBDIR)/libdiffusion.a -
lpthread -lpcrc -lssl -lcrypto

ARFLAGS +=
SOURCES = getting-started-publisher.c

TARGETDIR = target
OBJDIR = $(TARGETDIR)/objs
BINDIR = $(TARGETDIR)/bin
OBJECTS = $(SOURCES:.c=.o)
TARGETS = getting-started-publisher

all: prepare $(TARGETS)
.PHONY: all

prepare:
    mkdir -p $(OBJDIR) $(BINDIR)

$(OBJDIR)/%.o: %.c
    $(CC) $(CFLAGS) -o $@ $<

getting-started-publisher: $(OBJDIR)/getting-started-publisher.o
    $(CC) $< $(LDFLAGS) -o $(BINDIR)/$@

clean:
    rm -rf $(TARGETS) $(OBJECTS) $(TARGETDIR) core a.out

```

Connecting to the Diffusion server

One of the first actions your Diffusion client takes is to connect to the Diffusion server. Clients connect to the Diffusion server by opening a session. A session represents a logical context between a client and the Diffusion server. All interactions with the Diffusion server happen within a session.

Sessions

The act of opening the session establishes a connection with the the Diffusion server. When a session is opened it is assigned a unique session identifier by the Diffusion server, which identifies the session even if it becomes connected to another server.

The session does not receive input from the Diffusion server until it is started, but can be used to obtain features and perform certain setup actions before it is started.

Session state

The session between a client and the Diffusion server can be in one of a number of states.

The following diagram shows the session state model:

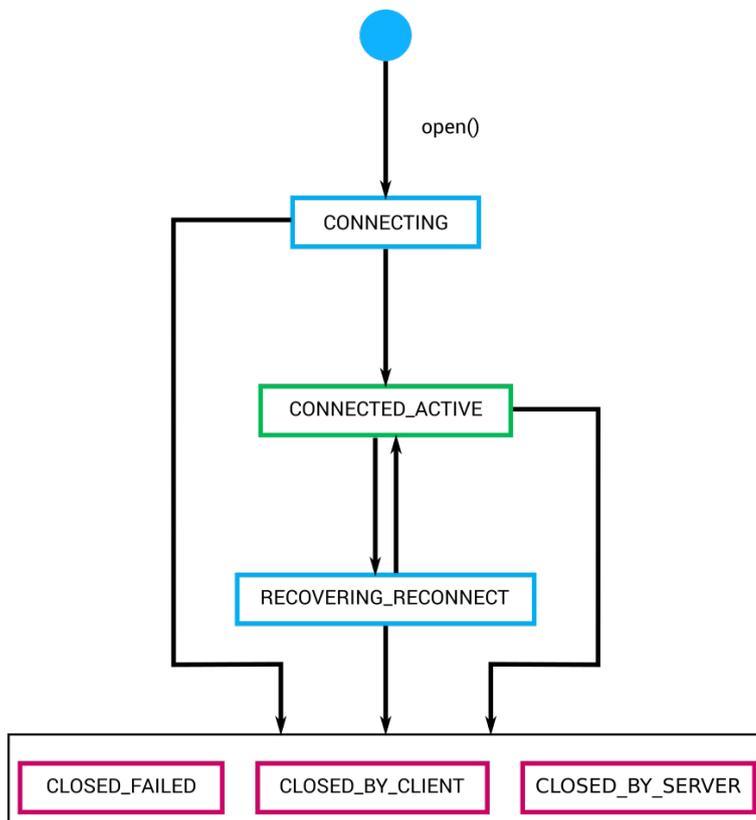


Figure 22: Session state model

CONNECTING

The client session is in this state while it attempts to connect to the Diffusion server. If the connection attempt is successful, the session changes to `CONNECTED_ACTIVE` state. If the connection is not successful, the session changes to one of the closed states.

CONNECTED_ACTIVE

The client session is in this state while it is connected to the Diffusion server. The session spends the majority of its lifetime in this state. If the session becomes disconnected and reconnect is enabled, the session changes to `RECOVERING_CONNECT` state. If the session closes, it changes to one of the closed states.

RECOVERING_CONNECT

The client session is in this state while it attempts to reconnect to the Diffusion server after a disconnection. If the reconnection attempt is successful, the session changes back to `CONNECTED_ACTIVE` state. If the reconnection attempt is not successful, the session changes to one of the closed states.

CLOSED_BY_CLIENT

The client session is in this state when it is closed by the client. If a session is in closed state, it cannot be reopened. A new session must be established.

CLOSED_BY_SERVER

The client session is in this state when it is closed by the Diffusion server. If a session is in closed state, it cannot be reopened. A new session must be established.

CLOSED_FAILED

The client session is in this state when it is closed for any reason other than a close by the client or by the Diffusion server. If a session is in closed state, it cannot be reopened. A new session must be established.

For more information, see [Managing your session](#) on page 243.

Session properties

When you connect to the Diffusion server by opening a session, the session is assigned a set of properties. These properties are assigned by either the Diffusion server or an authentication handler and can be used by clients to filter the set of connected client sessions to take actions on.

For more information, see [Session properties](#) on page 267.

Session roles

When a session authenticates with the Diffusion server, the session is assigned a set of roles. These roles are assigned by either the Diffusion server or an authentication handler and define the set of permissions a client session has.

For more information, see [Role-based authorization](#) on page 130.

Connecting basics

To make a connection to the Diffusion server the client must specify the host name and port number of the Diffusion server, the transport to use to connect, and whether that connection is secure.

The Diffusion API is an asynchronous API. As such, the client APIs for all languages provide asynchronous connect methods.

A subset of the Diffusion APIs also provide synchronous connect methods: the Android, Java, .NET, and C APIs. In the following sections, all examples for these APIs are synchronous for simplicity. For asynchronous examples for these APIs, see [Asynchronous connections](#) on page 241.

Connection parameters

host

The host name or IP address of the system on which the Diffusion server is located.

port

The port on which the Diffusion server accepts connections from clients using the Unified API. You can configure which ports to provide connectors for in the `Connectors.xml` configuration file. For more information, see [Connectors.xml](#) on page 583.

transport

The transport used to make the connection. For example, WebSocket (`ws`). The transports your client can use to make a connection depend on the client library capabilities. For more information, see [Platform support for the Diffusion Unified API libraries](#) on page 47.

secure

Whether the connection is made over SSL.

Connecting

In JavaScript, Android and Java, you can define each of these parameters individually:

JavaScript

```
diffusion.connect({
  host : 'host_name',
  port : 'port', // If not specified, port defaults to 80 for
                // standard connections or 443 for secure connections
  transports : 'transport', // If not specified, transports
                // defaults to 'WS' and the client uses a WebSocket connection
  secure : false // If not specified, secure defaults to false
}).then(function(session) { ... } );
```

Java and Android

```
final Session session = Diffusion
    .sessions()
    .serverHost("host_name")
    // If no port is specified, the port defaults to 80 for standard
    // connections or 443 for secure connections
    // There is no default port for DPT. If you use the DPT
    // transport, you must also define a port.
    .serverPort(port)
    // If no transports are specified, the connection defaults to
    // use the WebSocket transport
    .transports(transport)
    // If not specified, secure transport defaults to false
    .secureTransport(false)
    .open();
```

In Apple, Android, Java, .NET and C, composite the host, port, transport, and whether the connection is secure into a single URL-style string of the following form: *transport[s]://host:port*.

For example, `ws://diffusion.example.com:8080`.

Use this URL to open the connection to the Diffusion server:

Apple

```
// Excluding the port from the URL defaults to 80, or 443 for
// secure connections
[PTDiffusionSession openWithURL:[NSURL URLWithString:@"ws://
push.example.com"]
    completionHandler:^(PTDiffusionSession *
newSession, NSError * error)
{
    if (newSession) {
        NSLog(@"Session open.");
        self.session = newSession;
    } else {
        NSLog(@"Session Failed to open with error: %@", error);
    }
}
}];
```

Java and Android

```
Session session = Diffusion.sessions().open("url");
```

.NET

```
var session = Diffusion.Sessions.Open( "url" );
```

C

```
SESSION_T *session = NULL;
session = session_create(url, NULL, NULL, &session_listener, NULL,
NULL);
session_start(session, &error);
```

Connecting with multiple transports

In JavaScript, you can specify a list of transports. The client uses these transports to provide a *transport cascading* capability.

JavaScript

```
diffusion.connect({
  host : 'host_name',
  transports : ['transport', 'transport', 'transport']
}).then(function(session) { ... } );
```

Java and Android

```
final Session session = Diffusion
    .sessions()
    .serverHost("host_name")
    .serverPort(port)
    .transports(transport, transport, transport)
    .open();
```

1. The client attempts to connect using the first transport listed.
2. If the connection is unsuccessful, the client attempts to connect using the next transport listed.
3. This continues until the one of the following events happens:
 - The client makes a connection
 - The client has attempted to make a connection using every listed transport. If this happens, the connection fails.

You can use specify that a client attempt to connect with a transport more than once. This enables you to define retry behavior. For example:

JavaScript

```
transports: ['WS', 'XHR', 'WS']
```

Java and Android

```
.transports(WEBSOCKET, WEBSOCKET, WEBSOCKET)
```

Transport cascading is useful to specify in your clients as it enables them to connect from many different environments. Factors such as the firewall in use, your end-user's mobile provider, or your end-user's browser can affect which transports can successfully make a connection to the Diffusion server.

Asynchronous connections

All Diffusion APIs can connect asynchronously to the Diffusion server:

JavaScript

```
diffusion.connect({
  host : 'host_name',
```

```
port : 'port'
}).then(function(session) { ... } );
```

Apple

```
// Excluding the port from the URL defaults to 80, or 443 for
secure connections
[PTDiffusionSession openWithURL:[NSURL URLWithString:@"url"]
    completionHandler:^(PTDiffusionSession *
newSession, NSError * error)
{
    if (newSession) {
        NSLog(@"Session open.");
        self.session = newSession;
    } else {
        NSLog(@"Session Failed to open with error: %@", error);
    }
}
}];
```

Java and Android

```
// Define a callback that implements SessionFactory.OpenCallback and
pass this to the open method
Diffusion.sessions().open("url", callback);
```

.NET

```
// Define a callback that implements ISessionOpenCallback and pass
this to the open method
Diffusion.Sessions.Open("url", callback );
```

C

```
/*
 * Asynchronous connections have callbacks for notifying that
 * a connection has been made, or that an error occurred.
 */
SESSION_CREATE_CALLBACK_T *callbacks = calloc(1,
sizeof(SESSION_CREATE_CALLBACK_T));
callbacks->on_connected = &on_connected;
callbacks->on_error = &on_error;

session_create_async(url, principal, credentials,
&session_listener, reconnection_strategy, callbacks, &error);
```

Synchronous connections

The following APIs can connect synchronously to the Diffusion server:

Java and Android

```
Session session = Diffusion.sessions().open("url");
```

.NET

```
var session = Diffusion.Sessions.Open( "url" );
```

C

```
SESSION_T *session = NULL;
```

```
session = session_create(url, NULL, NULL, &session_listener, NULL,
    NULL);
session_start(session, &error);
```

When connecting to the Diffusion server using the Android API, prefer the asynchronous `open()` method with a callback. Using the synchronous `open()` method might open a connection on the same thread as the UI and cause a runtime exception. However, the synchronous `open()` method can be used in any thread that is not the UI thread.

Managing your session

When your client has opened a session with the Diffusion server, you can listen for session events to be notified when the session state changes. For more information about session states, see [Session state](#) on page 237.

JavaScript

In JavaScript, listen for the following events on the `session`:

- `disconnect`: The session has lost connection to the Diffusion server.
The session state changes from `CONNECTED_ACTIVE` to `RECOVERING_RECONNECT`. This event is only emitted if `reconnect` is enabled.
- `reconnect`: The session has re-established connection to the Diffusion server.
The session state changes from `RECOVERING_RECONNECT` to `CONNECTED_ACTIVE`.
- `close`: The session has closed. The provided close reason indicates whether this was caused by the client, the Diffusion server, a failure to connect, or an error.
The session state changes to one of `CLOSED_FAILED`, `CLOSED_BY_SERVER`, or `CLOSED_BY_CLIENT`.
- `error`: A session error occurs.

JavaScript

```
session.on('disconnect', function() {
    console.log('Lost connection to the server.');
```

Apple

In Apple, the following boolean properties are available on the states that are broadcast through the default notification center for the application process and posted on the main dispatch queue:

- `isConnected`: If true, the state is equivalent to the `CONNECTED_ACTIVE` state.
- `isRecovering`: If true, the state is equivalent to the `RECOVERING_RECONNECT` state.
- `isClosed`: If true, the state is one of `CLOSED_FAILED`, `CLOSED_BY_SERVER`, or `CLOSED_BY_CLIENT`.

The broadcast includes both the old state and new state of the session. It also includes an error property that is `nil` unless the session closure was caused by a failure.

Apple

```
NSNotificationCenter * nc = [NSNotificationCenter defaultCenter];
[nc addObserverForName:PTDiffusionSessionStateDidChangeNotification
    object:session
    queue:nil
    usingBlock:^(NSNotification * note)
{
    PTDiffusionSessionStateChange * change =
    note.userInfo[PTDiffusionSessionStateChangeUserInfoKey];
    NSLog(@"Session state change: %@", change);
}];
```

Other SDKs

In Android, Java, .NET, and C listen for changes to the session state. The listener provides both the old state and new state of the session. The states provided are those listed in the session state diagram. For more information, see [Session state](#) on page 237.

Java and Android

```
// Add the listener to the session
session.addListener(new Listener() {
    @Override
    public void onSessionStateChanged(Session session, State
    oldState, State newState) {

        System.out.println("Session state changed from " +
        oldState.toString() + " to " + newState.toString());

    }
});
```

.NET

```
// Add the listener to the session factory you will use to create the
session
var sessionFactory =
Diffusion.Sessions.SessionStateChangedHandler( ( sender, args ) => {

    Console.WriteLine( "Session state changed from " +
    args.OldState.ToString() + " to " + args.NewState.ToString() );

} );
```

C

```
// Define a session listener
static void
on_session_state_changed(SESSION_T *session,
    const SESSION_STATE_T old_state,
    const SESSION_STATE_T new_state)
{
    printf("Session state changed from %s (%d) to %s (%d)\n",
        session_state_as_string(old_state), old_state,
        session_state_as_string(new_state), new_state);
}

// ...

// Use the session listener when opening your session
SESSION_LISTENER_T session_listener = { 0 };
```

```
        session_listener.on_state_changed =
&on_session_state_changed;

        session_create_async(url, principal, credentials,
&session_listener, &reconnection_strategy, callbacks, &error);
```

Connecting securely

A Diffusion client can make secure connections to the Diffusion server over TLS. All supported transports can connect securely.

To connect securely do one of the following:

- In JavaScript, set the `secure` parameter to `true`
- In Android and Java, when specifying parameters individually, pass `true` to the `secureTransport()` method.
- If using a URL to connect, insert an “s” after the transport value in the `url` parameter. For example, `wss://diffusion.example.com:443`.

Configure the SSL context or behavior

A secure connection to the Diffusion server uses SSL to secure the communication.

When connecting over SSL, you might need to configure SSL.

- In JavaScript, the SSL context is provided by the browser.
- In Android, Java, and .NET, you can provide an SSL context when creating the session.
- In Apple, you can use the `sslOptions` property of the `Diffusion.Session` to provide a dictionary of values that specify the SSL behavior. For more information, see the [CFStreamConstants documentation](#).

Java and Android

```
Session session =
Diffusion.sessions().sslContext(ssl_context).open("secure_url");
```

.NET

```
var session =
Diffusion.Sessions.SslContext(ssl_context).Open( "secure_url" );
```

If no SSL context or behavior is specified, the client uses the default context or configuration.

Validating server-side certificates

Diffusion clients that connect over a secure transport use certificates to validate the security of their connection to the Diffusion server. These certificates are validated against any certificates in the set trusted by the framework, runtime, or platform that the client library runs on.

If the client does not trust the certificate provided by a CA, you can configure the client to add certificates to its trust store:

- For Java, see [Certificates](#) on page 161
- For .NET, see [Certificates](#) on page 163

You can also write a trust manager that explicitly allows the CA's certificates.

Disabling certificate validation on the client

You can disable client validation of the server-side certificates.

Note: We do not recommend disabling this validation on your production clients. However, it can be useful for testing.

Certificates can only be strictly validated if they have been issued by an appropriate Certificate Authority (CA) and if the CA's certificates are also known to your client.

Since certificates are specific to the domain name that the server is deployed on, Diffusion ships with demo certificates and these cannot be strictly validated. To test against a server with demo certificates, disable client-side SSL certificate validation as shown in the following examples:

Apple

```
// Create a session configuration with non-standard SSL options...
PTDiffusionMutableSessionConfiguration *const configuration =
    [PTDiffusionMutableSessionConfiguration new];
configuration.sslOptions = @{
    (__bridge id)kCFStreamSSLValidatesCertificateChain
        : (__bridge id)kCFBooleanFalse
};

// Use the configuration to open a new session...
[PTDiffusionSession openWithURL:[NSURL URLWithString:@"wss://
TestServer"]
                    configuration:configuration
                    completionHandler:^(PTDiffusionSession *session,
                                        NSError *error)
{
    // Check error is `nil`, then use session as required.
    // Ensure to maintain a strong reference to the session beyond
    the lifetime
    // of this callback, for example by assigning it to an instance
    variable.
}];
```

Java and Android

```
TrustManager tm = new X509TrustManager() {
    public void checkClientTrusted(X509Certificate[] chain,
    String authType) throws CertificateException {
    }

    public void checkServerTrusted(X509Certificate[] chain,
    String authType) throws CertificateException {
    }

    public X509Certificate[] getAcceptedIssuers()
    {
        return new X509Certificate[0];
    }
};

final SSLContext context = SSLContext.getInstance("TLS");
context.init( null, new TrustManager[] { tm }, null );

Session session =
Diffusion.sessions().sslContext(context).open("secure_url");
```

Connect to the Diffusion server with a security principal and credentials

The Diffusion server can accept anonymous connections. However, if your clients specify a security principal (for example, a username) and its associated credentials (for example, a password) when they connect, these client sessions can be authenticated and authorized in a more granular way.

Authentication parameters

principal

A string that contains the name of the principal or identity that is connecting to the Diffusion server. If a value is not specified when connecting, the principal defaults to ANONYMOUS.

credentials

Credentials are a piece of information that authenticates the principal. This can be empty or contain a password, a cryptographic key, an image, or any other piece of information.

If you connect to the Diffusion server using a principal and credentials, connect over SSL to ensure that these details are encrypted.

Connecting using any type of credentials

In JavaScript and C the method that opens a connection to the Diffusion server takes `principal` and `credentials` as parameters:

JavaScript

```
diffusion.connect({
  host : 'host_name',
  port : 'port',
  principal: 'principal',
  credentials: 'credentials'
});
```

C

```
SESSION_T *session = NULL;
session = session_create(url, principal, credentials,
    &session_listener, NULL, NULL);
session_start(session, &error);
```

Any form of credentials can be wrapped in a credentials object. This can be empty or contain a password, a cryptographic key, an image, or any other piece of information. The authentication handler is responsible for interpreting the bytes.

In the Apple, Android, Java, and .NET Unified API specify the credentials as a credentials object. The principal and credentials are specified when configuring the session before opening it:

Apple

```
NSData *const credentialsData = [NSData dataWithBytes:(char[])
{2,4,6,8} length:4];

PTDiffusionCredentials *const credentials =
[[PTDiffusionCredentials alloc] initWithData:credentialsData];
PTDiffusionSessionConfiguration *const sessionConfiguration =
[[PTDiffusionSessionConfiguration alloc]
initWithPrincipal:@"somePrincipalName"
```


Java and Android

```
Session session = Diffusion.sessions()  
    .principal("principal")  
    .password("credentials")  
    .open("url");
```

.NET

```
var session = Diffusion.Sessions  
    .Principal("principal")  
    .Password("credentials")  
    .Open("url");
```

Connecting using a byte array as credentials

The Android, Java, and .NET Unified API provide a convenience method that enables you to specify credentials as a byte array. The principal and credentials are specified when configuring the session before opening it:

Java and Android

```
Session session = Diffusion.sessions()  
    .principal("principal")  
    .customCredentials(credentials)  
    .open("url");
```

.NET

```
var session = Diffusion.Sessions  
    .Principal("principal")  
    .CustomCredentials(credentials)  
    .Open("url");
```

Changing the principal and credentials a session uses

The client session can change the principal and credentials it uses to connect to the Diffusion server at any time. For more information, see [Change the security principal and credentials associated with your client session](#) on page 266.

Connecting through an HTTP proxy

Clients can connect to the Diffusion server through an HTTP proxy by using the HTTP CONNECT verb to create the connection and tunneling any of the supported transports through that connection.

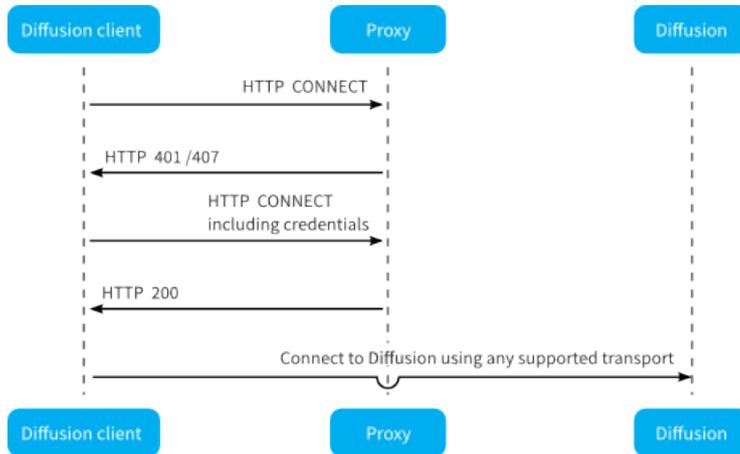


Figure 23: Flow of requests and responses when connecting to Diffusion through a proxy.

Android, Java, and .NET clients can connect to the Diffusion server through an HTTP proxy by specifying additional information on connection.

With no authentication at the proxy

When creating your session, add an HTTP proxy to the session by passing in the host and port number of the proxy.

Java and Android

```
Diffusion.sessions().httpProxy(host, port)
```

.NET

```
var session = Diffusion.Sessions
    .HttpProxy( host, port )
    .Open( diffusionUrl );
```

With basic authentication at the proxy

If the proxy requires basic authentication, the client can use the implementation in the Unified API to authenticate.

When creating your session, add an HTTP proxy to the session by passing in the host and port number of the proxy and a proxy authentication object that provides the challenge handler for basic authentication.

Java and Android

```
HTTPProxyAuthentication auth =
    Diffusion.proxyAuthentication().basic(username, password);
Diffusion.sessions().httpProxy(host, port, auth);
```

.NET

```
var clientAuth = Diffusion.ProxyAuthentication.Basic( username,
    password );
```

```
var session = Diffusion.Sessions
    .HttpProxy( host, port, clientAuth )
    .Open( diffusionUrl );
```

With another form of authentication at the proxy

If the proxy requires another form of authentication, the client can implement a challenge handler that the client uses to authenticate.

Implement the `HTTPProxyAuthentication` interface to provide a challenge handler that can handle the type of authentication your proxy uses. When creating your session, add an HTTP proxy to the session by passing in the host and port number of the proxy and a proxy authentication object that provides your challenge handler.

Note: The proxy authentication mechanism is separate from the client authentication mechanism and is transparent to the Diffusion server.

Connecting through a load balancer

Connections between Diffusion clients and Diffusion servers can be routed through a load balancer. Some clients can pass additional information to a load balancer in the request path of their URL.

Supported in: JavaScript, Android, and Java APIs

An additional request path can be specified to define the connection URL context. This request path can only be specified when connecting with individual parameters and not with the URL-style string.

JavaScript

```
diffusion.connect({
  host : 'host_name',
  port : 'port',
  transports : 'transport',
  secure : false,
  path: '/path/diffusion'
}).then(function(session) { ... } );
```

Java and Android

```
final Session session = Diffusion
    .sessions()
    .serverHost("host_name")
    .serverPort(port)
    .transports(transport)
    .secureTransport(false)
    .requestPath("/path/diffusion");
session.open();
```

The value of the request path must begin with `/` and end with `/diffusion`. The default value is `/diffusion`.

Load balancer configuration

Connections between Diffusion clients and Diffusion servers have specific requirements. If your load balancer handles Diffusion connections incorrectly, for example by routing subsequent client requests to different backend Diffusion servers, this can cause problems for your solution.

For more information about how to configure your load balancers to work with Diffusion, see [Load balancers](#) on page 642.

Reconnect to the Diffusion server

When clients connect to the Diffusion server over unreliable networks these connections can be lost. Clients can attempt to reconnect to the Diffusion server after they lose connection.

Diffusion keeps client sessions in the DISCONNECTED state for a period of time, during which the client can reconnect to the same session. The length of time the Diffusion server keeps a client session in the DISCONNECTED state for is configured for the connector that the client uses. For more information, see [Configuring connectors](#) on page 581.

Configuring reconnection on the client

Clients have reconnection enabled by default.

You can configure a reconnection timeout that restricts the amount of time the client can be disconnected and still reconnect to its session on the Diffusion server. The period of time that the Diffusion server keeps the session available for reconnect is the lowest of the following values:

- The reconnection timeout configured by the client when it creates its session
- The reconnection timeout configured on the Diffusion server for the connector that the client connects on

When the reconnection timeout period configured by the client ends, the client stops attempting to reconnect and closes its session.

JavaScript

```
diffusion.connect({
  host : 'url',
  reconnect : {
    // Specify the timeout in milliseconds
    timeout : reconnection_time
  }
})
```

Apple

```
PTDiffusionMutableSessionConfiguration *const
sessionConfiguration = [PTDiffusionMutableSessionConfiguration new];

// Specify the timeout in seconds
sessionConfiguration.reconnectionTimeout = @10;

[PTDiffusionSession openWithURL:url
 configuration:sessionConfiguration
 completionHandler:^(PTDiffusionSession *newSession,
 NSError *error)
 {
   if (newSession) {
     NSLog(@"Session open");
   } else {
     NSLog(@"Session Failed to open with error: %@", error);
   }
 }];
```

Java and Android

```
final Session session = Diffusion
    .sessions()
```

```
// Specify the timeout in milliseconds
.reconnectionTimeout(reconnection_time)
.open("url");
```

.NET

```
var session = Diffusion.Sessions
    // Specify the timeout in milliseconds
    .ReconnectionTimeout(reconnection_time)
    .Open("url");
```

C

```
reconnection_strategy_set_timeout(&reconnection_strategy, reconnection_time);
SESSION_T *session = session_create(url, NULL, NULL, NULL,
    &reconnection_strategy, NULL);
```

Set the value of the reconnection timeout to zero to disable reconnection. If no reconnection timeout is specified, a default of 60 seconds (60000 ms) is used.

You can also define your own custom reconnection behavior using reconnection strategies. For more information, see [Specifying a reconnection strategy](#) on page 254.

If no custom reconnection strategy is defined, the client attempts to reconnect at five second intervals until the reconnection timeout is reached.

Reliable reconnection

If a client loses connection to the Diffusion server, data sent between the client and the Diffusion server in either direction might be lost in transmission. If this happens and the client reconnects, lost data might cause the client state or topic data to be incorrect.

To prevent any data being lost, the reconnection process re-synchronizes the streams of messages from client to the Diffusion server and from the Diffusion server to client. When reconnecting, the client notifies the Diffusion server of the last message received and the earliest message it can send again. The Diffusion server resends any missing messages and instructs the client to resume from the appropriate message.

To be able to send messages again, the Diffusion server maintains a recovery buffer of sent messages. Some types of client also maintain a recovery buffer of sent messages that can be sent again if necessary.

If a message has been lost and is no longer present in the recovery buffer, the server will abort the reconnection. If reconnection succeeds, delivery of all messages is assured.

Configuring the recovery buffer on the client

JavaScript, Java, Android, and C clients can retain a buffer of messages that they have sent to the Diffusion server. In the case when messages from the client are lost in transmission during a disconnection and subsequent reconnection, the client can resend the missing messages to the Diffusion server.

In Java and Android, you can configure the size of this buffer, in messages, when creating your session on the Diffusion server:

Java and Android

```
final Session session = Diffusion
    .sessions()
    .recoveryBufferSize(number_of_messages)
    .open("url");
```

The default size of the recovery buffer is 128 messages.

The larger this buffer is, the greater the chance of successful reconnection. However, a larger buffer of messages increases the memory footprint of a client.

Configuring the recovery buffer on the Diffusion server

The recovery buffers on the Diffusion server can be configured on a per-connector basis in the `Connectors.xml` configuration file. For more information, see [Configuring connectors](#) on page 581.

Detecting connection problems

A client can automatically detect if there are problems with its connection to the Diffusion server and take action to handle any disconnection.

When a client detects that it has become disconnected from the Diffusion server, the session state changes from `CONNECTED` to one of the following states:

- If reconnection is enabled at the client and at the Diffusion server, the session state changes to `RECOVERING`.
- If reconnection is not enabled, the session state changes to `DISCONNECTED`.

The client can detect that it has become disconnected from the Diffusion server using the following methods:

Monitoring the connection activity

The client automatically monitors the activity between the client and the Diffusion server and uses this information to quickly discover any connection problems.

Using TCP state

Depending on the transport the client uses to connect to the Diffusion server, the client can use the TCP state to detect whether to change its state from `CONNECTED` to one of `RECOVERING` or `DISCONNECTED`.

- **WebSocket or DPT:** The client uses the TCP state to detect whether to trigger a state change.
- **HTTP Polling:** The client uses the TCP state at certain points during an HTTP request to detect whether to trigger a state change.

Specifying a reconnection strategy

Reconnection behavior can be configured using custom reconnection strategies.

The reconnection behavior of a client session can be configured using reconnection strategies. A reconnection strategy is applied when the session enters the `RECOVERING_RECONNECT` state, enabling the session to attempt to reconnect and recover its previous state.

Reconnection can only succeed if the client session is still available on the Diffusion server. The maximum time that the Diffusion server keeps client sessions in the `DISCONNECTED` state before closing them can be configured using the `Connectors.xml` configuration file. For more information, see [Configuring connectors](#) on page 581.

Individual client sessions can request a shorter reconnection timeout for their sessions or request to disable reconnection when they first connect to the Diffusion server

Examples

JavaScript

```
// When establishing a session, it is possible to specify whether
// reconnection
// should be attempted in the event of an unexpected disconnection.
// This allows
// the session to recover its previous state.

// Set the maximum amount of time we'll try and reconnect for to 10
// minutes
var maximumTimeoutDuration = 1000 * 60 * 10;

// Set the maximum interval between reconnect attempts to 60 seconds
var maximumAttemptInterval = 1000 * 60;

// Set an upper limit to the number of times we'll try to reconnect
// for
var maximumAttempts = 25;

// Count the number of reconnection attempts we've made
var attempts = 0;

// Create a reconnection strategy that applies an exponential back-
// off
// The strategy will be called with two arguments, start & abort.
// Both
// of these are functions, which allow the strategy to either start a
// reconnection attempt, or to abort reconnection (which will close
// the session)
var reconnectionStrategy = function(start, abort) {
  if (attempts > maximumAttempts) {
    abort();
  } else {
    var wait = Math.min(Math.pow(2, attempts++) * 100,
maximumAttemptInterval);

    // Wait the specified time period, and then start the
    // reconnection attempt
    setTimeout(start, wait);
  }
};

// Connect to the server.
diffusion.connect({
  host : 'diffusion.example.com',
  port : 443,
  secure : true,
  principal : 'control',
  credentials : 'password',
  reconnect : {
    timeout : maximumTimeoutDuration,
    strategy : reconnectionStrategy
  }
}).then(function(session) {

  session.on('disconnect', function() {
    // This will be called when we lose connection. Because we've
    // specified the
    // reconnection strategy, it will be called automatically
    // when this event
    // is dispatched
  });
});
```

```

    });

    session.on('reconnect', function() {
        // If the session is able to reconnect within the reconnect
        // timeout, this
        // event will be dispatched to notify that normal operations
        // may resume
        attempts = 0;
    });

    session.on('close', function() {
        // If the session is closed normally, or the session is
        // unable to reconnect,
        // this event will be dispatched to notify that the session
        // is no longer
        // operational.
    });
});

```

Apple

```

#import Diffusion;

@interface ExponentialBackoffReconnectionStrategy : NSObject
<PTDiffusionSessionReconnectionStrategy>
@end

@implementation CustomReconnectionStrategyExample {
    PTDiffusionSession* _session;
}

-(void)startWithURL:(NSURL*)url {
    NSLog(@"Connecting...");

    PTDiffusionMutableSessionConfiguration *const
    sessionConfiguration =
        [PTDiffusionMutableSessionConfiguration new];

    // Set the maximum amount of time we'll try and reconnect for to
    // 10 minutes.
    sessionConfiguration.reconnectionTimeout = @(10.0 * 60.0); //
    seconds

    // Set the reconnection strategy to be used.
    sessionConfiguration.reconnectionStrategy =
    [ExponentialBackoffReconnectionStrategy new];

    // Start connecting asynchronously.
    [PTDiffusionSession openWithURL:url
                        configuration:sessionConfiguration
                        completionHandler:^(PTDiffusionSession *session,
NSError *error)
    {
        if (!session) {
            NSLog(@"Failed to open session: %@", error);
            return;
        }

        // At this point we now have a connected session.
        NSLog(@"Connected.");

        // Set ivar to maintain a strong reference to the session.

```

```

        _session = session;
    }];
}

@end

@implementation ExponentialBackoffReconnectionStrategy {
    NSUInteger _attemptCount;
}

-(void) diffusionSession:(PTDiffusionSession *const)session
wishesToReconnectWithAttempt:
(PTDiffusionSessionReconnectionAttempt *const)attempt {
    // Limit the maximum time to delay between reconnection attempts
    to 60 seconds.
    const NSTimeInterval maximumAttemptInterval = 60.0;

    // Compute delay for exponential backoff based on the number of
    attempts so far.
    const NSTimeInterval delay = MIN(pow(2.0, _attemptCount++) * 0.1,
maximumAttemptInterval);

    // Schedule asynchronous execution.
    NSLog(@"Reconnection attempt scheduled for %.2fs", delay);
    dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(delay *
NSEC_PER_SEC)),
        dispatch_get_main_queue(), ^
    {
        NSLog(@"Attempting reconnection.");
        [attempt start];
    });
}

@end

```

Java and Android

```

import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.session.Session;
import com.pushtechology.diffusion.client.session.Session.Listener;
import com.pushtechology.diffusion.client.session.Session.State;
import
    com.pushtechology.diffusion.client.session.reconnect.ReconnectionStrategy;

/**
 * This example class demonstrates the ability to set a custom {@link
ReconnectionStrategy}
 * when creating sessions.
 *
 * @author Push Technology Limited
 * @since 5.5
 */
public class ClientWithReconnectionStrategy {

    private volatile int retries = 0;
    /**

```

```

    * Constructor.
    */
    public ClientWithReconnectionStrategy() {

        // Set the maximum amount of time we'll try and reconnect for
        to 10 minutes.
        final int maximumTimeoutDuration = 1000 * 60 * 10;

        // Set the maximum interval between reconnect attempts to 60
        seconds.
        final long maximumAttemptInterval = 1000 * 60;

        // Create a new reconnection strategy that applies an
        exponential backoff
        final ReconnectionStrategy reconnectionStrategy = new
        ReconnectionStrategy() {
            private final ScheduledExecutorService scheduler =
            Executors.newScheduledThreadPool(1);

            @Override
            public void performReconnection(final ReconnectionAttempt
            reconnection) {
                final long exponentialWaitTime =
                Math.min((long) Math.pow(2, retries++) * 100L,
                maximumAttemptInterval);

                scheduler.schedule(new Runnable() {
                    @Override
                    public void run() {
                        reconnection.start();
                    }
                }, exponentialWaitTime, TimeUnit.MILLISECONDS);
            }
        };

        final Session session =
        Diffusion.sessions().reconnectionTimeout(maximumTimeoutDuration)

        .reconnectionStrategy(reconnectionStrategy)

        .open("ws://
diffusion.example.com:80");
        session.addListener(new Listener() {
            @Override
            public void onSessionStateChanged(Session session, State
            oldState, State newState) {

                if (newState == State.RECOVERING_RECONNECT) {
                    // The session has been disconnected, and has
                    entered recovery state. It is during this state that
                    // the reconnect strategy will be called
                }

                if (newState == State.CONNECTED_ACTIVE) {
                    // The session has connected for the first time,
                    or it has been reconnected.
                    retries = 0;
                }

                if (oldState == State.RECOVERING_RECONNECT) {
                    // The session has left recovery state. It may
                    either be attempting to reconnect, or the attempt has
                    // been aborted; this will be reflected in the
                    newState.
                }
            }
        });
    }
}

```



```

        /// Here we put our actual reconnection logic. The async
keyword should always be added since it makes
        /// things easier for a void return type.
        /// </summary>
        /// <param name="reconnectionAttempt">The reconnection
attempt will be given by the session.</param>
        public async Task
PerformReconnection( IReconnectionAttempt reconnectionAttempt ) {
            ++counter;
            if ( counter <= 3 ) {
                // We start the next reconnection attempt
                reconnectionAttempt.Start();
            } else {
                counter = 0;

                // We abort any other reconnection attempt and
let the session switch to CLOSED_BY_SERVER.
                reconnectionAttempt.Abort();
            }
        }
    }

    /// <summary>
    /// This applies the custom reconnection strategy.
    /// </summary>
    public void SetCustomReconnectionStrategy() {
        // We don't need to hold a reference to the reconnection
strategy
        var sessionFactoryWithCustomStrategy
= Diffusion.Sessions.ReconnectionStrategy( new
MyReconnectionStrategy() );
    }

    /// <summary>
    /// Reconnection can be observed via session state changes
within the SessionStateChangeHandler.
    /// </summary>
    public void ObserveReconnection() {
        var sessionFactory =
Diffusion.Sessions.SessionStateChangedHandler( ( sender, args ) => {
            if
( args.NewState.Equals( SessionState.RECOVERING_RECONNECT ) ) {
                // This will be set on a connection loss and
indicates a reconnection attempt.
                // Unless reconnection is disabled, at which
point the session never gets switched to this state.
                Console.WriteLine( "We are in the process of
reconnecting." );
            } else if
( args.NewState.Equals( SessionState.CONNECTION_ATTEMPT_FAILED ) ) {
                // If a reconnection attempt fails because the
server session timed out, we won't be able
                // to reconnect anymore. At which point the
session will switch to this state.
                Console.WriteLine( "We couldn't connect." );
            } else if
( args.NewState.Equals( SessionState.CLOSED_BY_SERVER ) ) {
                // If the reconnection timeout is over, we will
switch to this state. In case of disabled
                // reconnection we will switch directly to this
state on a connection loss.
                Console.WriteLine( "We lost connection." );
            }
        } );
    }
}

```



```

} BACKOFF_STRATEGY_ARGS_T;

static RECONNECTION_ATTEMPT_ACTION_T
backoff_reconnection_strategy(SESSION_T *session, void *args)
{
    BACKOFF_STRATEGY_ARGS_T *backoff_args = args;

    printf("Waiting for %ld ms\n", backoff_args->current_wait);

    apr_sleep(backoff_args->current_wait * 1000); // µs -> ms

    // But only up to some maximum time.
    if(backoff_args->current_wait > backoff_args->max_wait) {
        backoff_args->current_wait = backoff_args->max_wait;
    }

    return RECONNECTION_ATTEMPT_ACTION_START;
}

static void
backoff_success(SESSION_T *session, void *args)
{
    printf("Reconnection successful\n");

    BACKOFF_STRATEGY_ARGS_T *backoff_args = args;
    backoff_args->current_wait = 0; // Reset wait.
}

static void
backoff_failure(SESSION_T *session, void *args)
{
    printf("Reconnection failed (%s)\n",
        session_state_as_string(session->state));

    BACKOFF_STRATEGY_ARGS_T *backoff_args = args;

    // Exponential backoff.
    if(backoff_args->current_wait == 0) {
        backoff_args->current_wait = 1;
    }
    else {
        backoff_args->current_wait *= 2;
    }
}

/*
 * Entry point for the example.
 */
int
main(int argc, char **argv)
{
    /*
     * Standard command-line parsing.
     */
    HASH_T *options = parse_cmdline(argc, argv, arg_opts);
    if(options == NULL || hash_get(options, "help") != NULL) {
        show_usage(argc, argv, arg_opts);
        return EXIT_FAILURE;
    }

    const char *url = hash_get(options, "url");
    const char *principal = hash_get(options, "principal");
    CREDENTIALS_T *credentials = NULL;

```

```

    const char *password = hash_get(options, "credentials");
    if(password != NULL) {
        credentials = credentials_create_password(password);
    }

    const unsigned int sleep_time = atol(hash_get(options,
"sleep"));

    SESSION_T *session;
    DIFFUSION_ERROR_T error = { 0 };

    SESSION_LISTENER_T session_listener = { 0 };
    session_listener.on_state_changed =
&on_session_state_changed;

    /*
     * Set the arguments to our exponential backoff strategy.
     */
    BACKOFF_STRATEGY_ARGS_T *backoff_args = calloc(1,
sizeof(BACKOFF_STRATEGY_ARGS_T));
    backoff_args->current_wait = 0;
    backoff_args->max_wait = 5000;

    /*
     * Create the backoff strategy.
     */
    RECONNECTION_STRATEGY_T *reconnection_strategy =
make_reconnection_strategy_user_function(backoff_reconnection_strategy,
backoff_args,
backoff_success,
backoff_failure,
NULL);

    /*
     * Only ever retry for 30 seconds.
     */
    reconnection_strategy_set_timeout(reconnection_strategy, 30 *
1000);

    /*
     * Create a session, synchronously.
     */
    session = session_create(url, principal, credentials,
&session_listener, reconnection_strategy, &error);
    if(session != NULL) {
        char *sid_str = session_id_to_string(session->id);
        printf("Session created (state=%d, id=%s)\n",
session_state_get(session), sid_str);
        free(sid_str);
    }
    else {
        printf("Failed to create session: %s\n",
error.message);
        free(error.message);
    }

    // With the exception of backoff_args, the reconnection
strategy is
    // copied withing session_create() and may be freed now.

```

```

    free(reconnection_strategy);

    /*
     * Sleep for a while.
     */
    sleep(sleep_time);

    /*
     * Close the session, and release resources and memory.
     */
    session_close(session, NULL);
    session_free(session);

    free(backoff_args);

    credentials_free(credentials);
    hash_free(options, NULL, free);

    return EXIT_SUCCESS;
}

```

Session failover

Session failover occurs when a client that disconnects from a Diffusion server attempts to connect to a different Diffusion server that also has information about that client's session.

For session failover to occur, session replication must be configured for a cluster of Diffusion servers. For more information, see [Configuring replication](#) on page 611.

Differences between session reconnection and session failover

When a client loses a load-balanced connection to Diffusion, one of the following things can occur when the client attempts to reconnect through the load balancer:

Session reconnection

The load balancer forwards the client connection to the Diffusion server it was previously connected to, if that server is still available. For more information, see [Reconnect to the Diffusion server](#) on page 252.

Session failover

The load balancer forwards the client connection to a different Diffusion server that shares information about the client's session, if session replication is enabled between the servers.

Prefer session reconnection to session failover wherever possible by ensuring that the load balancer is configured to route all connections from a specific client to the same server if that server is available.

Session reconnection is more efficient as less data must be sent to the client and has less risk of data loss, as sent messages can be recovered, in-flight requests are not lost, and handlers do not need to be registered again.

For more information, see [Routing strategies at your load balancer](#) on page 643.

To a client the process of disconnection and subsequent reconnection has the following differences for session reconnection or session failover.

Session reconnection	Session failover
The client connects to the same Diffusion server it was previously connected to.	The client connects to a Diffusion server different to the one it was previously connected to.
The client sends its last session token to the server.	

Session reconnection	Session failover
The server authenticates the client connection or validates its session token.	
<p>The server uses the session token to resynchronize the streams of messages between the server and client by resending any messages that were lost in transmission from a buffer of sent messages.</p> <p>If lost messages cannot be recovered because they are no longer present in a buffer, the server aborts the reconnection.</p>	<p>The server uses the session token to retrieve the session state and topic selections from the datagrid.</p>
<p>The server sends any messages that have been queued since the session disconnected.</p>	<p>The server uses the state to recover the session, uses the topic selections to match the subscribed topics, and sends the session the current topic value for each subscribed topic.</p> <p>Any in-flight requests made by the client session to the previous server are cancelled and the client session is notified by a callback. All handlers, including authentication handlers and update sources, that the client session had registered with the previous server are closed and receive a callback to notify them of the closure.</p>

Ping the Diffusion server

Ping the Diffusion server from your client. If the ping is successful it reports the round-trip time between your client and the Diffusion server.

Supported in: Java Unified API, .NET Unified API, Apple Unified API, Android Unified API, C Unified API

The Diffusion client libraries and the Diffusion server include capabilities that automatically check whether the connection is active. However, there might be times when you want to check the connection from within your client code. For example, if the client is aware that the device it is hosted on has recently changed from a 3G connection to a WiFi connection.

Use the pings capability to asynchronously ping the Diffusion server.

Apple

```
[_session.pings
 pingServerWithCompletionHandler:pingCompletionHandler];
```

Java and Android

```
Pings pings = session.feature(Pings.class);
pings.pingServer(context, callback);
```

.NET

```
IPings pings = session.GetPingFeature();
pings.PingServer( context, callback );
```

C

```
PING_USER_PARAMS_T params = {
    .on_ping_response = on_ping_response_user
};
ping_user(session, params);
```

Change the security principal and credentials associated with your client session

A client session can change the credentials it uses to authenticate with the Diffusion server at any time.

JavaScript

```
session.security.changePrincipal('admin',
'password').then(function() {
    console.log('Authenticated as admin');
});
```

Apple

```
[_session.security changePrincipal:@"somePrincipalWithAuthority"
                           credentials:[[PTDiffusionCredentials
alloc] initWithPassword:@"s3cret"]
      completionHandler:^(NSError *const error)
{
    if (error) {
        NSLog(@"Failed to change principal: %@", error);
    }
}];
```

Java and Android

```
security = session.feature(Security.class);
security.changePrincipal(
    principal,
    Diffusion.credentials().password(password),
    callback);
```

.NET

```
security = session.GetSecurityFeature();
security.ChangePrincipal( principal,
Diffusion.Credentials.Password( password ), callback );
```

C

```
// Specify callbacks for the change_principal request.
CHANGE_PRINCIPAL_PARAMS_T params = {
    .principal = hash_get(options, "principal"),
    .credentials = credentials,
    .on_change_principal = on_change_principal,
    .on_change_principal_failure =
on_change_principal_failure
};

// Do the change.
```

```
change_principal(session, params);
```

When the principal associated with a session changes, the following happens:

- The `$Principal` session property is updated to contain the new principal.
- The roles associated with the old principal are removed from the session and those roles associated with the new principal are assigned to the session.
- Topic subscriptions made with the old principal are not re-evaluated. The session remains subscribed to any topics the new principal does not have permissions for.

Session properties

A client session has a number of properties associated with it. Properties are key-value pairs. Both the key and the value are case sensitive.

Session properties provide a powerful way for clients to target actions at a specific client or set of clients whose session properties match a given criteria. Clients can use session filtering to select a set of clients to perform one of the following actions on:

- Send messages directly to that client or set of clients.
- Subscribe that client or set of clients to a topic.
- Unsubscribe that client or set of clients from a topic.

For more information, see [Session filtering](#) on page 268.

Clients can also request the full set of properties or a subset, for a particular client.

Fixed properties

Fixed properties are set by the Diffusion server when a client opens a session with it. Fixed property keys are prefixed by a dollar sign (\$). The fixed session properties are:

\$SessionId

The session identifier.

\$Principal

The security principal the session uses to connect to the Diffusion server.

\$ClientType

The client type of the session. For more information, see [Client types](#) on page 111.

\$Transport

The transport the client session uses to connect to the Diffusion server. For more information, see [Client types](#) on page 111.

\$ServerName

The name of the Diffusion server that the client connects to.

\$Connector

The name of the connector on which the client connected to the Diffusion server.

\$Country

The two letter country code for the country where the client's internet address is located. The value is uppercase.

\$Language

The two letter language code for the most common language of the country where the client's internet address is located. The value is lowercase.

User-defined properties

An authentication handler that allows the client session to connect can assign additional properties to the session. The keys of these properties are case sensitive, must begin with an alphabetic character, must be alphanumeric, and must not include any whitespace.

Related concepts

[Session filtering](#) on page 268

Session filters enable you to query the set of connected client sessions on the Diffusion server based on their session properties.

[Messaging to sessions](#) on page 375

A client session can use the MessagingControl feature to send individual messages to any known client session on any topic path. It can also register a handler for messages sent from client sessions.

[Managing subscriptions](#) on page 354

A client can use the SubscriptionControl feature to subscribe other client sessions to topics that they have not requested subscription to themselves and also to unsubscribe clients from topics. It also enables the client to register as the handler for routing topic subscriptions.

[Managing clients](#) on page 431

A client with the appropriate permissions can receive notifications and information about other client sessions. A client with the appropriate permissions can also manage these client sessions.

Session filtering

Session filters enable you to query the set of connected client sessions on the Diffusion server based on their session properties.

To perform an action on a subset of the connected client sessions, you can create a query expression that filters the set of connected client sessions by the values of their session properties. Filter query expressions are parsed and evaluated by the Diffusion server.

The query expression used to filter the session is made up of one or more search clauses chained together by boolean operators.

Creating a single search clause

Search clauses have the following form:

```
key operator 'value'
```

key

The key name of the session property to be tested. The key name is case sensitive.

operator

The operator that defines the test to be performed. The operator is not case sensitive.

value

The test value to be compared to the session property value. This value is a string and must be contained in single or double quotation marks. Any special characters must be escaped with Java escaping. The value is case sensitive.

Table 30: Session filter search clause operators

Operator	Description
IS	Tests whether the session property value associated with the property key matches the test value.
EQ	Equals. Tests whether the session property value associated with the property key matches the test value. Equivalent to 'IS'.
NE	Not equal. Tests whether the session property value associated with the key is not equal to the test value.

Examples: single search clause

Filter by clients that connect with the principal Ellington:

```
$Principal IS 'Ellington'
```

Filter by clients that connect to the Diffusion server using WebSocket:

```
$Transport EQ 'WEBSOCKET'
```

Filter by clients that are not located in the United Kingdom:

```
$Country NE 'GB'
```

Filter by clients that have the user-defined property Location set to San Jose:

```
Location IS "San Jose"
```

Filter by clients that have the user-defined property Status set to Active:

```
Status EQ 'Active'
```

Filter by clients that do not have the user-defined property Tier set to Premium:

```
Tier NE 'Premium'
```

Chaining multiple search clauses

Chain individual search clauses together using boolean operator or use the NOT operator to negate a search clause. Boolean operators are not case sensitive.

Table 31: Session filter boolean operators

Operator	Description
AND	Specifies that both joined search clauses must be true.
OR	Specifies that at least one of the joined search clauses must be true.

Operator	Description
NOT	Specifies that the following search clause or set of search clauses must not be true.

Use parentheses to group sets of search clauses and indicate the order of precedence for evaluation. If no order of precedence is explicitly defined, the AND operator takes precedence over the OR operator.

Examples: multiple search clauses

Filter by clients that connect with one of the principals Fitzgerald, Gillespie, or Hancock:

```
$Principal IS 'Fitzgerald' OR $Principal IS 'Gillespie' OR $Principal IS 'Hancock'
```

Filter by clients that connect to the Diffusion server using WebSocket and are located in France and have the user-defined property Status set to Active:

```
$Transport EQ 'WEBSOCKET' AND $Country IS 'FR' AND Status EQ 'Active'
```

Filter by clients that are located in the United States, but do not connect with either of the principals Monk or Peterson:

```
$Country EQ 'US' AND NOT ($Principal IS 'Monk' OR $Principal IS 'Peterson')
```

Filter by clients excluding those that have both the user-defined property Status set to Inactive and the user-defined property Tier set to Free:

```
NOT (Status IS 'Inactive' AND Tier IS 'Free')
```

Related concepts

[Session properties](#) on page 267

A client session has a number of properties associated with it. Properties are key-value pairs. Both the key and the value are case sensitive.

[Messaging to sessions](#) on page 375

A client session can use the MessagingControl feature to send individual messages to any known client session on any topic path. It can also register a handler for messages sent from client sessions.

[Managing subscriptions](#) on page 354

A client can use the SubscriptionControl feature to subscribe other client sessions to topics that they have not requested subscription to themselves and also to unsubscribe clients from topics. It also enables the client to register as the handler for routing topic subscriptions.

[Managing clients](#) on page 431

A client with the appropriate permissions can receive notifications and information about other client sessions. A client with the appropriate permissions can also manage these client sessions.

Receiving data from topics

A client can use the Topics feature to subscribe to a topic or to fetch the state of a topic.

Streams

Clients use streams registered against sets of topics to receive values published to those topics.

Streams are registered using topic selectors and can be registered multiple times with different selectors. If more than one of the topic selectors used to register a stream matches a topic, the stream receives each value for that topic only once.

Multiple streams can be registered against the same topic. All streams registered against the topic receive a value for it. The order that these streams receive the value is not defined.

The following table describes the types of stream a client can use to receive values from topics:

Stream type	Description
Value stream	<p>Value streams are typed. Register value streams against a set of topics by using a topic selector. A value stream receives updates for any subscribed topics that match the value stream's type and the topic selectors used when registering the value stream.</p> <p>If a value stream receives a delta update, this delta is automatically applied to a locally cached value so that the stream always receives full values.</p> <p>A value stream can have one of the following types:</p> <p>JSON</p> <p>JSON topics are routed to this type of stream.</p> <p>Binary</p> <p>Binary topics are routed to this type of stream.</p> <p>Content</p> <p>JSON, binary, and single value topics are routed to this type of stream.</p> <p>Value streams are provided in the JavaScript, Android, and Java APIs.</p>
Topic stream	<p>Topic streams are not typed and are used to receive value and delta updates for all subscribed topics that match the topic selectors used when registering the value stream. This type of stream provides the value and the deltas but relies upon the application to apply the deltas to a client maintained current value.</p> <p>Where a value type is available for your topic, we recommend you use a value stream instead of a topic stream.</p>
Fetch stream	<p>Fetch streams are not typed and are used to receive responses to fetch requests for all topics that match the topic selectors used when registering the value stream.</p>

You can register one or more fallback streams to receive updates to subscribed topics that do not have a value stream or topic stream registered against them.

Subscribing to a topic

Required permissions: `select_topic` and `read_topic` permissions for the specified topic

A client can subscribe to a topic to receive updates that are published to the topic. If the topic has state, when the client subscribes to that topic it receives the topic state as a full value. Subsequent updates to the data on the topic can be received as delta update messages or as values depending on the type of the topic and the structure of its data.

Subscribing to multiple topics using a topic selector

Required permissions: `select_topic` and `read_topic` permissions for the specified topics

A client can subscribe to multiple topics in a single request by using topic selectors. Topic selectors enable you to select whole branches of the topic tree or use regular expressions to select topics based on the names in the topic path.

For more information, see [Topic selectors in the Unified API](#) on page 61.

Fetching the state of a topic

Required permissions: `select_topic` and `read_topic` permissions for the specified topic

A client can send a fetch request for the state of a topic. If the topic is of a type that maintains its state, the Diffusion server provides the current state of that topic to the client.

Example: Subscribe to a topic

The following examples use the Unified API to subscribe to topics and assign handlers to topics to receive the topic content.

JavaScript

```
diffusion.connect({
  host   : 'diffusion.example.com',
  port   : 443,
  secure : true
}).then(function(session) {

  // 1. Subscriptions are how sessions receive streams of data from
  // the server.

  // When subscribing, a topic selector is used to select which
  // topics to subscribe to. Topics do not need to exist
  // at the time of subscription - the server dynamically resolves
  // subscriptions as topics are added or removed.

  // Subscribe to the "foo" topic with an inline callback function
  var subscription = session.subscribe('foo', function(update) {
    // Log the new value whenever the 'foo' topic is updated
    // By default, we get a Buffer object which preserves binary
    // data.
    console.log(update);
  });

  // Callbacks can also be registered after the subscription has
  // occurred
  subscription.on({
```

```

    update : function(value, topic) {
        console.log('Update for topic: ' + topic, value);
    },
    subscribe : function(details, topic) {
        console.log('Subscribed to topic: ' + topic);
    },
    unsubscribe : function(reason, topic) {
        console.log('Unsubscribed from topic:' + topic);
        subscription.close();
    }
});

// 2. Sessions may unsubscribe from any topic to stop receiving
data

// Unsubscribe from the "foo" topic. Sessions do not need to have
previously been subscribed to the topics they are
// unsubscribing from. Unsubscribing from a topic will result in
the 'unsubscribe' callback registered above being
// called.
session.unsubscribe('foo');

// 3. Subscriptions / Unsubscriptions can select multiple topics
using Topic Selectors

// Topic Selectors provide regex-like capabilities for
subscribing to topics. These are resolved dynamically, much
// like subscribing to a single topic.
var subscription2 = session.subscribe('?foo/.*/[a-z]');

// 4. Subscriptions can use transformers to convert update values

// Subscribe to a topic and then convert all received values to
JSON. Transforming a subscription creates a new
// subscription stream, rather than modifying the original. This
assumes that the topic is a single value topic
// receiving stringified JSON and is not a JSON topic.
session.subscribe('bar').transform(JSON.parse).on('update',
function(value, topic) {
    console.log('Got JSON update for topic: ' + topic, value);
});

// 5. Metadata can be used within transformers to parse data

// Create a simple metadata instance
var meta = new diffusion.metadata.RecordContent();

// Add a single record/field
meta.addRecord('record', {
    'field' : meta.string('some-value')
});

// Subscribe to a topic and transform with the metadata
session.subscribe('baz').transform(meta).on('update',
function(value) {
    console.log('Field value: ',
value.get('record').get('field'));
});
});

```

Apple

```
@import Diffusion;

@interface SubscribeUnsubscribeExample
    (PTDiffusionTopicStreamDelegate) <PTDiffusionTopicStreamDelegate>
@end

@implementation SubscribeUnsubscribeExample {
    PTDiffusionSession* _session;
}

-(void)startWithURL:(NSURL*)url {
    NSLog(@"Connecting...");

    [PTDiffusionSession openWithURL:url
        completionHandler:^(PTDiffusionSession *session,
            NSError *error)
        {
            if (!session) {
                NSLog(@"Failed to open session: %@", error);
                return;
            }

            // At this point we now have a connected session.
            NSLog(@"Connected.");

            // Set ivar to maintain a strong reference to the session.
            _session = session;

            // Register self as the fallback handler for topic updates.
            [session.topics addFallbackTopicStreamWithDelegate:self];

            // Wait 5 seconds and then subscribe.
            [self performSelector:@selector(subscribe:)
                withObject:session afterDelay:5.0];
        }];
}

static NSString *const _TopicSelectorExpression = @"*Assets//";

-(void)subscribe:(const id)object {
    PTDiffusionSession *const session = object;

    NSLog(@"Subscribing...");
    [session.topics
        subscribeWithTopicSelectorExpression:_TopicSelectorExpression
        completionHandler:^(NSError *
            const error)
        {
            if (error) {
                NSLog(@"Subscribe request failed. Error: %@", error);
            } else {
                NSLog(@"Subscribe request succeeded.");

                // Wait 5 seconds and then unsubscribe.
                [self performSelector:@selector(unsubscribe:)
                    withObject:session afterDelay:5.0];
            }
        }];
}

-(void)unsubscribe:(const id)object {
```

```

    PTDiffusionSession *const session = object;

    NSLog(@"Unsubscribing...");
    [session.topics
unsubscribeFromTopicSelectorExpression:_TopicSelectorExpression
                                     completionHandler:^(NSError
* const error)
    {
        if (error) {
            NSLog(@"Unsubscribe request failed. Error: %@", error);
        } else {
            NSLog(@"Unsubscribe request succeeded.");

            // Wait 5 seconds and then subscribe.
            [self performSelector:@selector(subscribe:)
withObject:session afterDelay:5.0];
        }
    }];
}

@end

@implementation SubscribeUnsubscribeExample
(PTDiffusionTopicStreamDelegate)

-(void)diffusionStream:(PTDiffusionStream * const)stream
    didUpdateTopicPath:(NSString * const)topicPath
        content:(PTDiffusionContent * const)content
        context:(PTDiffusionUpdateContext * const)context {
    NSString *const string = [[NSString alloc]
initWithData:content.data encoding:NSUTF8StringEncoding];
    NSLog(@"\t%@ = \"%@\\"", topicPath, string);
}

-(void)    diffusionStream:(PTDiffusionStream * const)stream
    didSubscribeToTopicPath:(NSString * const)topicPath
        details:(PTDiffusionTopicDetails * const)details
    {
    NSLog(@"Subscribed: \"%@\\" (%@)", topicPath, details);
}

-(void)    diffusionStream:(PTDiffusionStream * const)stream
    didUnsubscribeFromTopicPath:(NSString * const)topicPath
        reason:(const
PTDiffusionTopicUnsubscriptionReason)reason {
    NSLog(@"Unsubscribed: \"%@\\" [Reason: %@]", topicPath,
PTDiffusionTopicUnsubscriptionReasonToString(reason));
}

@end

```

Java and Android

```

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.content.Content;
import com.pushtechology.diffusion.client.features.Topics;
import
    com.pushtechology.diffusion.client.features.Topics.ValueStream;
import com.pushtechology.diffusion.client.session.Session;

```

```

import
  com.pushtechnology.diffusion.client.topics.details.TopicSpecification;
import com.pushtechnology.diffusion.datatype.json.JSON;

/**
 * In this simple and commonest case for a client we just subscribe
 * to a few
 * topics and assign handlers for each to receive content.
 * <P>
 * This makes use of the 'Topics' feature only.
 * <P>
 * To subscribe to a topic, the client session must have the
 * 'select_topic' and 'read_topic' permissions for that branch of the
 * topic tree.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public final class ClientSimpleSubscriber {

    private static final Logger LOG =
        LoggerFactory.getLogger(ClientSimpleSubscriber.class);

    private final Session session;

    /**
     * Constructor.
     */
    public ClientSimpleSubscriber() {

        session =

Diffusion.sessions().principal("client").password("password")
        .open("ws://diffusion.example.com:80");

        // Use the Topics feature to add a topic stream for
        // Foo and all topics under Bar and request subscription to
those topics
        final Topics topics = session.feature(Topics.class);
        topics.addStream(">Foo", Content.class, new FooStream());
        topics.addStream(">Bar//", JSON.class, new BarStream());
        topics.subscribe(
            Diffusion.topicSelectors().anyOf("Foo", "Bar//"),
            new Topics.CompletionCallback.Default());
    }

    /**
     * Close session.
     */
    public void close() {
        session.close();
    }

    /**
     * The stream for all messages on the 'Foo' topic.
     */
    private class FooStream extends ValueStream.Default<Content> {
        @Override
        public void onValue(
            String topicPath,
            TopicSpecification specification,
            Content oldValue,
            Content newValue) {

```

```

        LOG.info(newValue.asString());
    }
}

/**
 * The stream for all messages on 'Bar' topics which are JSON
 topics.
 */
private class BarStream extends ValueStream.Default<JSON> {
    @Override
    public void onValue(
        String topicPath,
        TopicSpecification specification,
        JSON oldValue,
        JSON newValue) {
        LOG.info(newValue.toJsonString());
    }
}
}

```

.NET

```

using System;
using System.Threading;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Content;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Features.Topics;
using PushTechnology.ClientInterface.Client.Topics.Details;

namespace PushTechnology.ClientInterface.GettingStarted {
    /// <summary>
    /// A client that subscribes to the topic 'foo/counter'.
    /// </summary>
    public sealed class SubscribingClient {
        public static void Main( string[] args ) {

            // Connect anonymously
            var session = Diffusion.Sessions.Open( "ws://
localhost:8080" );

            // Get the Topics feature to subscribe to topics
            var topics = session.GetTopicsFeature();

            // Add a topic stream for 'foo/counter' and request
subscription
            topics.AddStream( ">foo/counter", new
CounterTopicStream() );

            topics.Subscribe( ">foo/counter", new
TopicsCompletionCallbackDefault() );

            // Stay connected for 1 minute
            Thread.Sleep( TimeSpan.FromMinutes( 1 ) );

            session.Close();
        }
    }

    /// <summary>
    /// A simple IValueStream implementation.

```

```

    /// </summary>
    internal sealed class CounterTopicStream : IValueStream<IContent>
    {
        /// <summary>
        /// Notification of stream being closed normally.
        /// </summary>
        public void OnClose() {
            Console.WriteLine( "The subscription stream is now
closed." );
        }
        /// <summary>
        /// Notification of a contextual error related to this
callback.
        /// </summary>
        /// <remarks>
        /// Situations in which <code>OnError</code> is called
include the session being closed, a communication
        /// timeout, or a problem with the provided parameters. No
further calls will be made to this callback.
        /// </remarks>
        /// <param name="errorReason"></param>
        public void OnError( ErrorReason errorReason ) {
            Console.WriteLine( "An error has occurred : {0}",
errorReason );
        }
        /// <summary>
        /// Notification of a successful subscription.
        /// </summary>
        /// <param name="topicPath"></param>
        /// <param name="specification"></param>
        public void OnSubscription( string topicPath,
ITopicSpecification specification ) {
            Console.WriteLine( "Client subscribed to {0} ",
topicPath );
        }
        /// <summary>
        /// Notification of a successful unsubscription.
        /// </summary>
        /// <param name="topicPath">topic</param>
        /// <param name="specification">the specification of the
topic</param>
        /// <param name="reason">error reason</param>
        public void OnUnsubscription( string topicPath,
ITopicSpecification specification, TopicUnsubscribeReason reason ) {
            Console.WriteLine( "Client unsubscribed from {0} : {1}",
topicPath, reason );
        }

        /// <summary>
        /// Topic update received.
        /// </summary>
        /// <param name="topicPath">topic</param>
        /// <param name="specification">the specification of the
topic</param>
        /// <param name="oldValue">value prior to update</param>
        /// <param name="newValue">value after update</param>
        public void OnValue( string topicPath, ITopicSpecification
specification, IContent oldValue, IContent newValue ) {
            Console.WriteLine( "New value of {0} is {1}", topicPath,
newValue.AsString() );
        }
    }
}

```

```
}
```

C

```
/*
 * This is a sample client which connects to Diffusion and subscribes
 * to topics using a user-specified selector. Any messages received
 * on
 * those topics are then displayed to standard output.
 */

#include <stdio.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"

ARG_OPTS_T arg_opts[] = {
    ARG_OPTS_HELP,
    {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
     ARG_HAS_VALUE, "ws://localhost:8080"},
    {'t', "topic_selector", "Topic selector", ARG_REQUIRED,
     ARG_HAS_VALUE, NULL},
    END_OF_ARG_OPTS
};

/*
 * This callback is used when the session state changes, e.g. when a
 * session
 * moves from a "connecting" to a "connected" state, or from
 * "connected" to
 * "closed".
 */
static void
on_session_state_changed(SESSION_T *session,
                        const SESSION_STATE_T old_state,
                        const SESSION_STATE_T new_state)
{
    printf("Session state changed from %s (%d) to %s (%d)\n",
          session_state_as_string(old_state), old_state,
          session_state_as_string(new_state), new_state);
}

/*
 * When a subscribed message is received, this callback is invoked.
 */
static int
on_topic_message(SESSION_T *session, const TOPIC_MESSAGE_T *msg)
{
    printf("Received message for topic %s\n", msg->name);
    printf("Payload: (%d bytes) %.*s\n",
          (int)msg->payload->len,
          (int)msg->payload->len,
          msg->payload->data);

    hexdump_buf(msg->payload);

    return HANDLER_SUCCESS;
}

/*
 * This callback is fired when Diffusion responds to say that a topic
```

```

    * subscription request has been received and processed.
    */
static int
on_subscribe(SESSION_T *session, void *context_data)
{
    printf("on_subscribe\n");
    return HANDLER_SUCCESS;
}

/*
 * This is callback is for when Diffusion response to an
 * unsubscription
 * request to a topic, and only indicates that the request has been
 * received.
 */
static int
on_unsubscribe(SESSION_T *session, void *context_data)
{
    printf("on_unsubscribe\n");
    return HANDLER_SUCCESS;
}

/*
 * Publishers and control clients may choose to subscribe any other
 * client to
 * a topic of their choice at any time. We register this callback to
 * capture
 * messages from these topics and display them.
 */
static int
on_unexpected_topic_message(SESSION_T *session, const TOPIC_MESSAGE_T
 *msg)
{
    printf("Received a message for a topic we didn't subscribe to
 (%s)\n", msg->name);
    printf("Payload: %.*s\n", (int)msg->payload->len, msg-
 >payload->data);
    return HANDLER_SUCCESS;
}

/*
 * We use this callback when Diffusion notifies us that we've been
 * subscribed
 * to a topic. Note that this could be called for topics that we
 * haven't
 * explicitly subscribed to - other control clients or publishers may
 * ask to
 * subscribe us to a topic.
 */
static int
on_notify_subscription(SESSION_T *session, const
 SVC_NOTIFY_SUBSCRIPTION_REQUEST_T *request, void *context)
{
    printf("on_notify_subscription: %d: \"%s\"\n",
        request->topic_info.topic_id,
        request->topic_info.topic_path);
    return HANDLER_SUCCESS;
}

/*
 * This callback is used when we receive notification that this
 * client has been

```

```

* unsubscribed from a specific topic. Causes of the unsubscription
are the same
* as those for subscription.
*/
static int
on_notify_unsubscription(SESSION_T *session, const
    SVC_NOTIFY_UNSUBSCRIPTION_REQUEST_T *request, void *context)
{
    printf("on_notify_unsubscription: ID: %d, Path: %s, Reason:
%d\n",
        request->topic_id,
        request->topic_path,
        request->reason);
    return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
    /*
    * Standard command-line parsing
    */
    HASH_T *options = parse_cmdline(argc, argv, arg_opts);
    if(options == NULL || hash_get(options, "help") != NULL) {
        show_usage(argc, argv, arg_opts);
        return EXIT_FAILURE;
    }

    char *url = hash_get(options, "url");
    char *topic = hash_get(options, "topic_selector");

    /*
    * A SESSION_LISTENER_T holds callbacks to inform the client
    * about changes to the state. Used here for informational
    * purposes only.
    */
    SESSION_LISTENER_T session_listener = { 0 };
    session_listener.on_state_changed =
&on_session_state_changed;

    /*
    * Creating a session requires at least a URL. Creating a
    * session initiates a connection with Diffusion.
    */
    DIFFUSION_ERROR_T error = { 0 };
    SESSION_T *session = NULL;
    session = session_create(url, NULL, NULL, &session_listener,
NULL, &error);
    if(session == NULL) {
        fprintf(stderr, "TEST: Failed to create session\n");
        fprintf(stderr, "ERR : %s\n", error.message);
        return EXIT_FAILURE;
    }

    /*
    * When issuing commands to Diffusion (in this case,
subscribe
    * to a topic), it's typical that more than one message may
be
    * received in response and a handler can be installed for
    * each message type. In the case of subscription, we can
    * install handlers for:
    * 1. The topic message data (on_topic_message).

```

```

        * 2. Notification that the subscription has been received
        *   (on_subscribe).
        * 3. Topic details (on_topic_details).
        */
        notify_subscription_register(session,
(NOTIFY_SUBSCRIPTION_PARAMS_T) { .on_notify_subscription =
on_notify_subscription });
        notify_unsubscription_register(session,
(NOTIFY_UNSUBSCRIPTION_PARAMS_T) { .on_notify_unsubscription =
on_notify_unsubscription });

        subscribe(session, (SUBSCRIPTION_PARAMS_T) { .topic_selector
= topic, .on_topic_message = on_topic_message, .on_subscribe =
on_subscribe });

        /*
        * Install a global topic handler to capture messages for
        * topics we haven't explicitly subscribed to, and therefore
        * don't have a specific handler for.
        */
        session->global_topic_handler = on_unexpected_topic_message;

        /*
        * Receive messages for 5 seconds.
        */
        sleep(5);

        /*
        * Unsubscribe from the topic
        */
        unsubscribe(session, (UNSUBSCRIPTION_PARAMS_T)
{.topic_selector = topic, .on_unsubscribe = on_unsubscribe} );

        /*
        * Wait for any unsubscription notifications to be received.
        */
        sleep(5);

        /*
        * Politely tell Diffusion we're closing down.
        */
        session_close(session, NULL);
        session_free(session);

        return EXIT_SUCCESS;
}

```

Change the URL from that provided in the example to the URL of the Diffusion server.

Related concepts

[Topic selectors in the Unified API](#) on page 61

A topic selector identifies one or more topics. You can create a topic selector object from a pattern expression.

[Topic selectors in the Classic API \(deprecated\)](#) on page 69

A topic selector is a string that can be used by the Classic API to select more than one topic by indicating that subordinate topics are to be included or by fuzzy matching on topic names or both.

Example: Subscribe to a JSON topic

The following examples subscribe to JSON topics and receive a stream of values from the topics.

JavaScript

```
diffusion.connect({
  host    : 'diffusion.example.com',
  port    : 443,
  secure  : true,
  principal : 'control',
  credentials : 'password'
}).then(function(session) {

  // 1. Data Types are exposed from the top level Diffusion
  namespace. It is often easier
  // to assign these directly to a local variable.
  var jsonDataType = diffusion.datatypes.json();

  // 2. Data Types are currently provided for JSON and Binary topic
  types.
  session.topics.add('topic/json',
diffusion.topics.TopicType.JSON);

  // 3. Values can be created directly from the data type.
  var jsonValue = jsonDataType.from({
    "foo" : "bar"
  });

  // Topics are updated using the standard update mechanisms
  session.topics.update('topic/json', jsonValue);

  // Subscriptions are performed normally
  session.subscribe('topic/json');

  // 4. Streams can be specialised to provide values from a
  specific datatype.
  session.stream('topic/json').asType(jsonDataType).on('value',
function(topic, specification, newValue, oldValue) {
  // When a JSON or Binary topic is updated, any value handlers
  on a subscription will be called with both the
  // new value, and the old value.

  // The oldValue parameter will be undefined if this is the
  first value received for a topic.

  // For JSON topics, value#get returns a JavaScript object
  // For Binary topics, value#get returns a Buffer instance
  console.log("Update for " + topic, newValue.get());
});

  // 5. Raw values of an appropriate type can also be used for JSON
  and Binary topics.
  // For example, plain JSON objects can be used to update JSON
  topics.
  session.topics.update('topic/json', {
    "foo" : "baz",
```

```

        "numbers" : [1, 2, 3]
    });
});

```

Apple

```

#import Diffusion;

@interface JSONSubscribeExample (PTDiffusionJSONValueStreamDelegate)
<PTDiffusionJSONValueStreamDelegate>
@end

/**
 This example demonstrates a client consuming JSON topics.

 It is assumed that under the FX topic there is a JSON topic for each
 currency
 which contains a map of conversion rates to each target currency.
 For example,
 FX/GBP could contain {"USD":"123.45","HKD":"456.3"}.

 @note For a topic updater compatible with this example, see the
 following
 in our Java examples: ControlClientUpdatingJSONTopics
 */
@implementation JSONSubscribeExample {
    PTDiffusionSession* _session;
}

-(void)startWithURL:(NSURL *const)url {
    NSLog(@"Connecting...");

    [PTDiffusionSession openWithURL:url
                        completionHandler:^(PTDiffusionSession *session,
NSLError *error)
    {
        if (!session) {
            NSLog(@"Failed to open session: %@", error);
            return;
        }

        // At this point we now have a connected session.
        NSLog(@"Connected.");

        // Set ivar to maintain a strong reference to the session.
        _session = session;

        // Register self as the fallback handler for JSON value
updates.
        PTDiffusionValueStream *const valueStream =
            [PTDiffusionJSON valueStreamWithDelegate:self];
        [session.topics addFallbackStream:valueStream];

        // Subscribe.
        NSLog(@"Subscribing...");
        [session.topics subscribeWithTopicSelectorExpression:@"?FX/"

completionHandler:^(NSError * const error)
    {
        if (error) {
            NSLog(@"Subscribe request failed. Error: %@", error);
        } else {

```

```

        NSLog(@"Subscribe request succeeded.");
    }
    }];
}

-(NSString *)currencyFromTopicPath:(NSString *const)topicPath {
    // The currency from which we're converting is the last component
    // of the
    // topic path - e.g. topic path "FX/GBP" is currency "GBP".
    return [topicPath lastPathComponent];
}

@end

@implementation JSONSubscribeExample
    (PTDiffusionJSONValueStreamDelegate)

-(void) diffusionStream:(PTDiffusionStream *const)stream
    didSubscribeToTopicPath:(NSString *const)topicPath
    specification:(PTDiffusionTopicSpecification
    *const)specification {
    NSString *const currency = [self
    currencyFromTopicPath:topicPath];
    NSLog(@"Subscribed: Rates from %@", currency);
}

-(void)diffusionStream:(PTDiffusionValueStream *const)stream
    didUpdateTopicPath:(NSString *const)topicPath
    specification:(PTDiffusionTopicSpecification
    *const)specification
    oldJSON:(PTDiffusionJSON *const)oldJSON
    newJSON:(PTDiffusionJSON *const)newJSON {
    NSString *const currency = [self
    currencyFromTopicPath:topicPath];

    // We're assuming that the incoming JSON document is correct as
    // expected,
    // in that the root element is a map of currencies to which we
    // have
    // conversion rates.
    NSError * error;
    NSDictionary *const map = [newJSON objectWithError:&error];
    if (!map) {
        NSLog(@"Failed to create map from received JSON. Error: %@",
        error);
        return;
    }

    // For the purposes of a meaningful example, only emit a log line
    if we
    // have a rate for GBP to USD.
    if ([currency isEqualToString:@"GBP"]) {
        const id rate = map[@"USD"];
        if (rate) {
            NSLog(@"Rate for GBP to USD: %@", rate);
        }
    }
}

-(void) diffusionStream:(PTDiffusionStream *const)stream
    didUnsubscribeFromTopicPath:(NSString *const)topicPath

```

```

        specification:(PTDiffusionTopicSpecification
*const)specification
        reason:(const
PTDiffusionTopicUnsubscriptionReason)reason {
    NSString *const currency = [self
currencyFromTopicPath:topicPath];
    NSLog(@"Unsubscribed: Rates from %@", currency);
}
@end

```

Java and Android

```

package com.pushtechology.diffusion.examples;

import static java.util.Objects.requireNonNull;

import java.io.IOException;
import java.math.BigDecimal;
import java.util.Map;

import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.dataformat.cbor.CBORFactory;
import com.fasterxml.jackson.dataformat.cbor.CBORParser;
import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.features.Topics;
import
    com.pushtechology.diffusion.client.features.Topics.UnsubscribeReason;
import com.pushtechology.diffusion.client.session.Session;
import
    com.pushtechology.diffusion.client.topics.details.TopicSpecification;
import com.pushtechology.diffusion.datatype.json.JSON;

/**
 * This demonstrates a client consuming JSON topics.
 * <P>
 * It is assumed that under the FX topic there is a JSON topic for
each currency
 * which contains a map of conversion rates to each target currency.
For
 * example, FX/GBP could contain {"USD":"123.45","HKD":"456.3"}.
 * <P>
 * All updates will be notified to a listener.
 *
 * @author Push Technology Limited
 * @since 5.7
 * @see ControlClientUpdatingJSONTopics
 */
public final class ClientConsumingJSONTopics {

    private static final String ROOT_TOPIC = "FX";
    private static final String TOPIC_SELECTOR = String.format("?
%s/", ROOT_TOPIC);

    private final RatesListener listener;

    private final Session session;

    /**
     * Constructor.
     *

```

```

    * @param serverUrl for example "ws://diffusion.example.com:80
    */
    public ClientConsumingJSONTopics(String serverUrl, RatesListener
listener) {

        this.listener = requireNonNull(listener);

        session =

Diffusion.sessions().principal("client").password("password")
        .open(serverUrl);

        // Use the Topics feature to add a topic stream and subscribe
to all
        // topics under the root
        final Topics topics = session.feature(Topics.class);
        topics.addStream(TOPIC_SELECTOR, JSON.class, new
RatesStream());
        topics.subscribe(TOPIC_SELECTOR, new
Topics.CompletionCallback.Default());
    }

    /**
     * Close session.
     */
    public void close() {
        session.feature(Topics.class).unsubscribe(
            TOPIC_SELECTOR,
            new Topics.CompletionCallback.Default() {
                @Override
                public void onComplete() {
                    session.close();
                }
            });
    }

    private static String pathToCurrency(String path) {
        return path.substring(path.indexOf('/') + 1);
    }

    /**
     * A listener for Rates updates.
     */
    public interface RatesListener {

        /**
         * Notification of a new rate or rate update.
         *
         * @param currency the base currency
         * @param rates map of rates
         */
        void onNewRates(String currency, Map<String, BigDecimal>
rates);

        /**
         * Notification of a rate being removed.
         *
         * @param currency the base currency
         */
        void onRatesRemoved(String currency);
    }

    /**

```

```

    * The value stream.
    */
    private final class RatesStream extends
Topics.ValueStream.Default<JSON> {

        private final CBORFactory factory = new CBORFactory();
        private final ObjectMapper mapper = new ObjectMapper();
        private final TypeReference<Map<String, BigDecimal>>
typeReference =
            new TypeReference<Map<String, BigDecimal>>() {
            };

        @Override
        public void onValue(
            String topicPath,
            TopicSpecification specification,
            JSON oldValue,
            JSON newValue) {
            try {
                // Use the third-party Jackson library to parse the
newValue's
                // binary representation and convert to a map
                final CBORParser parser =
                    factory.createParser(newValue.asInputStream());
                final Map<String, BigDecimal> map =
                    mapper.readValue(parser, typeReference);
                final String currency = pathToCurrency(topicPath);
                listener.onNewRates(currency, map);
            }
            catch (IOException ex) {
                ex.printStackTrace();
            }
        }

        @Override
        public void onUnsubscription(
            String topicPath,
            TopicSpecification specification,
            UnsubscribeReason reason) {

            final String currency = pathToCurrency(topicPath);
            listener.onRatesRemoved(currency);
        }
    }
}

```

Change the URL from that provided in the example to the URL of the Diffusion server.

Example: Fetch topic state

The following examples use the Unified API to fetch the current state of a topic without subscribing to the topic.

Apple

```

@import Diffusion;

@interface FetchExample (PTDiffusionFetchStreamDelegate)
<PTDiffusionFetchStreamDelegate>
@end

```

```

@implementation FetchExample {
    PTDiffusionSession* _session;
}

-(void)startWithURL:(NSURL*)url {
    NSLog(@"Connecting...");

    [PTDiffusionSession openWithURL:url
        completionHandler:^(PTDiffusionSession *session,
        NSError *error)
    {
        if (!session) {
            NSLog(@"Failed to open session: %@", error);
            return;
        }

        // At this point we now have a connected session.
        NSLog(@"Connected.");

        // Set ivar to maintain a strong reference to the session.
        _session = session;

        // Send fetch request.
        [session.topics fetchWithTopicSelectorExpression:@"**Assets//"
        delegate:self];
    }];
}

@end

@implementation FetchExample (PTDiffusionFetchStreamDelegate)

-(void)diffusionStream:(PTDiffusionStream * const)stream
    didFetchTopicPath:(NSString * const)topicPath
    content:(PTDiffusionContent * const)content {
    NSLog(@"Fetch Result: %@ = \"%@\\"", topicPath, content);
}

-(void)diffusionDidCloseStream:(PTDiffusionStream * const)stream {
    NSLog(@"Fetch stream finished.");
}

-(void)diffusionStream:(PTDiffusionStream * const)stream
    didFailWithError:(NSError * const)error {
    NSLog(@"Fetch stream failed error: %@", error);
}

@end

```

Java and Android

```

package com.pushtechology.diffusion.examples;

import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.features.Topics;
import
    com.pushtechology.diffusion.client.features.Topics.FetchContextStream;
import com.pushtechology.diffusion.client.session.Session;
import com.pushtechology.diffusion.client.topics.TopicSelector;

/**

```

```

* This is a simple example of a client that fetches the state of
topics but
* does not subscribe to them.
* <P>
* This makes use of the 'Topics' feature only.
*
* @author Push Technology Limited
* @since 5.0
*/
public final class ClientUsingFetch {

    private final Session session;
    private final Topics topics;

    /**
     * Constructor.
     */
    public ClientUsingFetch() {

        session =

Diffusion.sessions().principal("client").password("password")
                .open("ws://diffusion.example.com:80");

        topics = session.feature(Topics.class);
    }

    /**
     * Issues a fetch request for a topic or selection of topics.
     *
     * @param topicSelector a {@link TopicSelector} expression
     * @param fetchContext context string to be returned with the
fetch
     * response(s)
     * @param stream callback for fetch responses
     */
    public void fetch(
        String topicSelector,
        String fetchContext,
        FetchContextStream<String> stream) {

        topics.fetch(topicSelector, fetchContext, stream);
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }
}

```

.NET

```

using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Session;

namespace Examples {
    /// <summary>

```

```

    /// This is a simple example of a client that fetches the state
of topics but does not subscribe to them.
    ///
    /// This makes use of the <see cref="ITopics"/> feature only.
    /// </summary>
    public class ClientUsingFetch {
        private readonly ISession session;
        private readonly ITopics topics;

        public ClientUsingFetch() {
            session =
Diffusion.Sessions.Principal( "client" ).Password( "password" )
                .Open( "ws://diffusion.example.com:80" );

            topics = session.GetTopicsFeature();
        }

        /// <summary>
        /// Issues a fetch request for a topic or selection of
topics.
        /// </summary>
        /// <param name="topicSelector">A <see cref="TopicSelector"/>
expression.</param>
        /// <param name="fetchContext">The context string to be
returned with the fetch response(s).</param>
        /// <param name="stream">The callback for fetch responses.</
param>
        public void Fetch( string topicSelector, string fetchContext,
IFetchContextStream<string> stream ) {
            topics.Fetch( topicSelector, fetchContext, stream );
        }

        /// <summary>
        /// Close the session.
        /// </summary>
        public void Close() {
            session.Close();
        }
    }
}

```

C

```

/*
 * This is a sample client which connects to Diffusion and
demonstrates
 * the following features:
 *
 * 1. Fetch topic state using a user-specified topic selector.
 * 2. Connect to Diffusion with a username and password.
 * 3. Automatic retry of a connection if unable to connect at the
first
 *    attempt.
 */

#include <stdio.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"

extern void topic_message_debug();

```

```

ARG_OPTS_T arg_opts[] = {
    ARG_OPTS_HELP,
    {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
    ARG_HAS_VALUE, "ws://localhost:8080"},
    {'t', "topic_selector", "Topic selector", ARG_REQUIRED,
    ARG_HAS_VALUE, NULL},
    {'r', "retries", "Number of connection retries",
    ARG_OPTIONAL, ARG_HAS_VALUE, "3"},
    {'d', "retry_delay", "Delay (in ms) between connection
attempts", ARG_OPTIONAL, ARG_HAS_VALUE, "1000"},
    {'p', "principal", "Principal (username) for the connection",
    ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    {'c', "credentials", "Credentials (password) for the
connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    END_OF_ARG_OPTS
};

/*
 * This callback is used when the session state changes, e.g. when a
 * session
 * moves from a "connecting" to a "connected" state, or from
 * "connected" to
 * "closed".
 */
static void
on_session_state_changed(SESSION_T *session, const SESSION_STATE_T
old_state, const SESSION_STATE_T new_state)
{
    printf("Session state changed from %s (%d) to %s (%d)\n",
        session_state_as_string(old_state), old_state,
        session_state_as_string(new_state), new_state);
    if(new_state == CONNECTED_ACTIVE) {
        printf("Session ID=%s\n",
            session_id_to_string(session->id));
    }
}

/*
 * This callback is invoked when Diffusion acknowledges that it has
 * received
 * the fetch request. It does not indicate that there will be any
 * subsequent
 * messages; see on_topic_message() and on_fetch_status_message() for
 * that.
 */
static int
on_fetch(SESSION_T *session, void *context)
{
    puts("Fetch acknowledged by server");
    return HANDLER_SUCCESS;
}

/*
 * This callback is invoked when all messages for a topic selector
 * have
 * been received, or there was some kind of server-side error during
 * the
 * fetch processing.
 */
static int
on_fetch_status_message(SESSION_T *session,
                        const SVC_FETCH_STATUS_RESPONSE_T *status,

```

```

        void *context)
{
    switch(status->status_flag) {
    case DIFFUSION_TRUE:
        puts("Fetch succeeded");
        break; //exit(0);
    case DIFFUSION_FALSE:
        puts("Fetch failed");
        break; //exit(1);
    default:
        printf("Unknown fetch status: %d\n", status-
>status_flag);
        break;
    }

    return HANDLER_SUCCESS;
}

/*
 * When a fetched message is received, this callback is invoked.
 */
static int
on_topic_message(SESSION_T *session, const TOPIC_MESSAGE_T *msg)
{
    printf("Received message for topic %s\n", msg->name);
    printf("Payload: %.*s\n", (int)msg->payload->len, msg-
>payload->data);

#ifdef DEBUG
        topic_message_debug(response->payload);
#endif

    return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
    /*
     * Standard command-line parsing.
     */
    HASH_T *options = parse_cmdline(argc, argv, arg_opts);
    if(options == NULL || hash_get(options, "help") != NULL) {
        show_usage(argc, argv, arg_opts);
        return EXIT_FAILURE;
    }

    char *url = hash_get(options, "url");
    char *topic = hash_get(options, "topic_selector");
    int retries = atoi(hash_get(options, "retries"));
    long retry_delay = atol(hash_get(options, "retry_delay"));

    /*
     * A SESSION_LISTENER_T holds callbacks to inform the client
     * about changes to the state. Used here for informational
     * purposes only.
     */
    SESSION_LISTENER_T foo_listener = {
        foo_listener.on_state_changed =
&on_session_state_changed
    };

    /*

```

```

    * The client-side API can automatically keep retrying to
    * connect to the Diffusion server if it's not immediately
    * available.
    */
    RECONNECTION_STRATEGY_T *reconnection_strategy =
        make_reconnection_strategy_repeating_attempt(retries,
retry_delay);

    /*
    * Creating a session requires at least a URL. Creating a
session
    * initiates a connection with Diffusion.
    */
    SESSION_T *session;
    DIFFUSION_ERROR_T error = { 0 };
    session = session_create(url,
        hash_get(options, "principal"),
credentials_create_password(hash_get(options, "credentials")),
        &foo_listener,
reconnection_strategy, &error);
    if(session == NULL) {
        fprintf(stderr, "TEST: Failed to create session\n");
        fprintf(stderr, "ERR : %s\n", error.message);
        return EXIT_FAILURE;
    }

    /*
    * Register handlers for callbacks we're interested in
    * relating to the fetch request. In particular, we want to
    * know about the topic messages that are returned, and the
    * status message which tells us when all messages have been
    * received for the selector (or, if something went wrong.)
    */
    FETCH_PARAMS_T params = {
        .selector = topic,
        .on_topic_message = on_topic_message,
        .on_fetch = on_fetch,
        .on_status_message = on_fetch_status_message
    };

    /*
    * Issue the fetch request.
    */
    fetch(session, params);

    /*
    * Wait for 5 seconds for the results to come in.
    */
    sleep(5);

    /*
    * Clean up.
    */
    session_close(session, NULL);
    session_free(session);

    return EXIT_SUCCESS;
}

```

Change the URL from that provided in the example to the URL of the Diffusion server.

Managing topics

A client can use the TopicControl feature of the Unified API to add and remove topics at the server.

Currently all topics created using a client have a lifespan the same as the Diffusion server. The topics remain at the Diffusion server even after the client session that created them has closed unless you explicitly specify that the topic is removed with the session.

Adding topics by initial value

Required permissions: modify_topic

A client can add a topic and define its type by providing an initial value. Diffusion uses the initial value to derive the topic type and to set the initial value of the topic.

The following table lists the topic types derived from different types of provided values:

Value type	Topic type	Metadata	Initial value
JSON	JSON	Not applicable	The supplied value
Binary	Binary	Not applicable	The supplied value
Content created using a builder method	Record	The metadata of the content is derived from the records and fields in the content with the following assumptions: <ul style="list-style-type: none">Numeric values are assumed to be <code>MIntegerString</code>Numeric values that contain a decimal point (.) are assumed to be <code>MDecimalString</code> with a scale equal to the number of places after the decimal pointAll other values are assumed to be <code>MString</code>	The supplied content
Content not created using a builder method	Single value	<code>MString</code>	The supplied content as a string
Integer, Long, Short, Byte, BigInteger, AtomicInteger, AtomicLong	Single value	<code>MIntegerString</code>	A value derived from the string representation of the supplied value
BigDecimal	Single value	<code>MDecimalString</code> with scale from supplied value	A value derived from the string representation of the supplied value

Value type	Topic type	Metadata	Initial value
Double, Float	Single value	MDecimalString with scale 2	A value derived from the string representation of the supplied value Note: We do not recommend using floating point numbers. If used, the number is converted to decimal using <i>half even</i> rounding.
Other	Single value	MString	A string representation of the supplied value

The `addTopicFromValue` operation is asynchronous and calls back to notify of either successful creation of the topic or failure to create the topic. If the topic add fails at the Diffusion server, the reason for failure is returned.

A client can create topics subordinate to topics created by another client.

Note: It is not currently possible to add new topics under branches of the topic tree that have been created by internal publishers.

Adding topics with topic specifications

Required permissions: `modify_topic`

Supported platforms: JavaScript, Android, Java

To create a JSON or binary topic, you can either create the topic by defining just the topic type or use the more complex topic specification to specify other attributes of the topic.

You can use the same instance of topic specification to create many topics.

The `addTopic` operation is asynchronous and calls back to notify of either successful creation of the topic or failure to create the topic. If the topic add fails at the Diffusion server, the reason for failure is returned.

A client can create topics subordinate to topics created by another client.

Note: It is not currently possible to add new topics under branches of the topic tree that have been created by internal publishers.

Adding other topics

Required permissions: `modify_topic`

For a client to create a topic it must first define the topic details that describe the topic. Builders of topic details can be created using the TopicControl feature.

You can use the same instance of topic details to create many topics. This is recommended when many topics with the same definition are to be created, because caching optimizations occur that prevent complex definitions from being transmitted to the Diffusion server many times.

For some types of topic, setting up metadata is part of the task of describing the topic.

The client can use the TopicControl feature to supply the initial state of the topic to the Diffusion server, as content, when the topic is created.

The `addTopic` operation is asynchronous and calls back to notify of either successful creation of the topic or failure to create the topic. If the topic add fails at the Diffusion server, the reason for failure is returned. Possible reasons for failure include the following:

- The topic already exists at the Diffusion server
- The name of the supplied topic is not valid
- The supplied details are not valid. This can occur only if properties are supplied.
- A user-supplied class cannot be found or instantiated. This can occur if you try to create a routing, paged, custom, or service topic and you have defined server-side classes to instantiate the topic.
- A referenced topic cannot be found
- Permission to create the topic was denied
- An error occurred trying to initialize the newly created topic with the supplied content, possibly because it was not validly formatted

A client can create topics subordinate to topics created by another client.

Note: It is not currently possible to add new topics under branches of the topic tree that have been created by internal publishers.

Removing topics

Required permissions: `modify_topic`

A client can remove topics anywhere in the topic tree. The remove operation takes a topic selector, which enables the client to remove many topics at once.

You can also opt to remove all topics beneath a selected topic path by appending the descendant pattern qualifiers, `/` and `//`. For more information, see [Topic selectors in the Unified API](#) on page 61.

Example: Create a topic

The following examples use the TopicControl feature in the Unified API to create topics.

JavaScript

```
var diffusion = require('diffusion');

// Connect to the server. Change these options to suit your own
// environment.
// Node.js does not accept self-signed certificates by default. If
// you have
// one of these, set the environment variable
// NODE_TLS_REJECT_UNAUTHORIZED=0
// before running this example.
diffusion.connect({
  host    : 'diffusion.example.com',
  port    : 443,
  secure  : true,
  principal : 'control',
  credentials : 'password'
}).then(function(session) {
  // 1. Topics can be created with a specified topic path and
  // value. If the path contains multiple levels, any
  // intermediary topic path that do not already have topics remain
  // unchanged.
```

```

    // Create a topic with string values, and an initial value of
    "xyz".
    session.topics.add('topic/string', 'xyz');

    // Create a topic with integer values, and an initial value of
    123.
    session.topics.add('topic/integer', 123);

    // Create a topic with decimal values, with an implicit scale of
    2 and an initial value of 1.23.
    session.topics.add('topic/decimal', 1.23);

    // 2. Adding a topic returns a result, which allows us to handle
    when the operation has either
    // completed successfully or encountered an error.
    session.topics.add('topic/result', 'abc').then(function(result) {
        console.log('Added topic: ' + result.topic);
    }, function(reason) {
        console.log('Failed to add topic: ', reason);
    });

    // Adding a topic that already exists will succeed, so long as it
    has the same value type
    session.topics.add('topic/result', 'xyz').then(function(result) {
        // result.added will be false, as the topic already existed
        console.log('Added topic: ' + result.topic, result.added);
    });

    // Because the result returned from adding a topic is a promise,
    we can easily chain
    // multiple topic adds together
    session.topics.add('chain/foo',
1).then(session.topics.add('chain/bar', 2))

    .then(session.topics.add('chain/baz', 3))

    .then(session.topics.add('chain/bob', 4))
        .then(function() {
            console.log('Added all
topics');
        }, function(reason) {
            console.log('Failed to add
topic', reason);
        });

    // 3. Metadata can be used to create topics that will contain
    values of a specified format.

    // RecordContent formats data in a series of records and fields,
    similar to tabular data.
    // Each record & field is named, allowing direct lookup of
    values. Each field value has a
    // particular type (string, integer, decimal)
    var metadata = new diffusion.metadata.RecordContent();

    // Records are like rows in a table. They can have fields
    assigned, with default values.
    // You can add fields all at once like this, or individually (see
    below).
    var game = metadata.addRecord('game', 1, {
        'title' : metadata.string(),
        'round' : metadata.integer(0),

```

```

        'count' : metadata.integer(0)
    });

    // Records and fields can be set as occurring a certain number of
    // times.
    var player = metadata.addRecord('player', metadata.occurs(0, 8));

    // Add fields to a record individually.
    player.addField('name', 'Anonymous');
    player.addField('score', 0);

    // Adding the topic works just like normal.
    session.topics.add('games/some-game', metadata);

    // And the metadata can be re-used for multiple topics.
    session.topics.add('games/some-other-game', metadata);

    // 4. Using metadata, it is possible to create a topic with both
    // a metadata format, and the initial value

    // Topic values can be produced from metadata via the builder
    // interface
    var builder = metadata.builder();

    // Values must be set before a value can be created
    builder.add('game', { title : 'Planet Express!', count : 3 });

    builder.add('player', { name : 'Fry', score : 0 });
    builder.add('player', { name : 'Amy', score : 0 });
    builder.add('player', { name : 'Kif', score : 0 });

    // Build a content instance
    var content = builder.build();

    // Now that the content has been built, a topic can be added with
    // the metadata & initial value
    session.topics.add('games/yet-another-game', metadata,
    content).then(function() {
        console.log('Topic was added with metadata and content');
    });
});

```

Apple

```

#import "AddTopicExample.h"

#import Diffusion;

@implementation AddTopicExample {
    PTDiffusionSession* _session;
}

-(void)startWithURL:(NSURL*)url {

    PTDiffusionCredentials *const credentials =
        [[PTDiffusionCredentials alloc]
        initWithPassword:@"password"];

    PTDiffusionSessionConfiguration *const sessionConfiguration =
        [[PTDiffusionSessionConfiguration alloc]
        initWithPrincipal:@"control"

```

```

credentials:credentials];

    NSLog(@"Connecting...");

    [PTDiffusionSession openWithURL:url
                        configuration:sessionConfiguration
                        completionHandler:^(PTDiffusionSession *session,
NSError *error)
    {
        if (!session) {
            NSLog(@"Failed to open session: %@", error);
            return;
        }

        // At this point we now have a connected session.
        NSLog(@"Connected.");

        // Set ivar to maintain a strong reference to the session.
        _session = session;

        // Add topics.
        [self addTopicsForSession:session];
    }];
}

-(void)addTopicsForSession:(PTDiffusionSession *const)session {
    // Add a stateless topic.
    [session.topicControl addWithTopicPath:@"Example/Stateless"

type:PTDiffusionTopicType_Stateless
                                value:nil
                                completionHandler:^(NSError *const error)
    {
        if (error) {
            NSLog(@"Failed to add stateless topic. Error: %@",
error);
        } else {
            NSLog(@"Stateless topic created.");
        }
    }];

    // Add a single value topic with an initial value.
    NSData *const data = [@"Hello"
dataUsingEncoding:NSUTF8StringEncoding];
    PTDiffusionContent *const initialValue =
        [[PTDiffusionContent alloc] initWithData:data];
    [session.topicControl addWithTopicPath:@"Example/SingleValue"

type:PTDiffusionTopicType_SingleValue
                                value:initialValue
                                completionHandler:^(NSError * _Nullable
error)
    {
        if (error) {
            NSLog(@"Failed to add single value topic. Error: %@",
error);
        } else {
            NSLog(@"Single value topic created.");
        }
    }];
}
}

```

```
@end
```

Java and Android

```
import java.util.List;

import com.pushtechology.diffusion.client.Diffusion;
import
    com.pushtechology.diffusion.client.callbacks.TopicTreeHandler;
import com.pushtechology.diffusion.client.content.Content;
import
    com.pushtechology.diffusion.client.content.RecordContentBuilder;
import com.pushtechology.diffusion.client.content.metadata.MContent;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicControl;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicControl.AddCon
import
    com.pushtechology.diffusion.client.features.control.topics.TopicControl.Remove
import com.pushtechology.diffusion.client.session.Session;
import
    com.pushtechology.diffusion.client.topics.details.RecordTopicDetails;
import
    com.pushtechology.diffusion.client.topics.details.TopicDetails;

/**
 * An example of using a control client to add topics.
 * <P>
 * This uses the 'TopicControl' feature only.
 * <P>
 * To add or remove topics, the client session must have the
 * 'modify_topic'
 * permission for that branch of the topic tree.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientAddingTopics {

    private final Session session;

    private final TopicControl topicControl;

    /**
     * Constructor.
     */
    public ControlClientAddingTopics() {

        session =

Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");

        topicControl = session.feature(TopicControl.class);

    }

    /**
     * Adds a topic with type derived from the initial value.
     * <P>
     * This uses the simple convenience method for adding topics
     where the topic
```

```

    * type (and metadata, if appropriate) are derived from a
    supplied value
    * which can be any object. For example, an Integer would result
    in a single
    * value topic of type integer or a JSON object would result in a
    JSON
    * topic.
    *
    * @param topicPath full topic path
    * @param initialValue an initial value for the topic
    * @param context this will be passed back to the callback when
reporting
    * success or failure of the topic add
    * @param callback to notify result of operation
    * @param <T> the value type
    * @return the topic details used to add the topic
    */
    public <T> TopicDetails addTopic(
        String topicPath,
        T initialValue,
        String context,
        AddContextCallback<String> callback) {

        return topicControl.addTopicFromValue(
            topicPath,
            initialValue,
            context,
            callback);
    }

    /**
    * Add a record topic from a list of initial values.
    * <P>
    * This demonstrates the simplest mechanism for adding a record
topic by
    * supplying values that both the metadata and the initial values
are
    * derived from.
    *
    * @param topicPath full topic path
    * @param initialValues the initial values for the topic fields
which will
    * also be used to derive the metadata definition of the
topic
    * @param context this will be passed back to the callback when
reporting
    * success or failure of the topic add
    * @param callback to notify result of operation
    * @return the topic details used to add the topic
    */
    public TopicDetails addRecordTopic(
        String topicPath,
        List<String> initialValues,
        String context,
        AddContextCallback<String> callback) {

        return topicControl.addTopicFromValue(
            topicPath,

Diffusion.content().newBuilder(RecordContentBuilder.class)
                .putFields(initialValues).build(),
            context,
            callback);

```

```

    }

    /**
     * Adds a record topic with supplied metadata and optional
     initial content.
     * <P>
     * This example shows details being created and would be fine
     when creating
     * topics that are all different but if creating many record
     topics with the
     * same details then it is far more efficient to pre-create the
     details.
     *
     * @param topicPath the full topic path
     * @param metadata pre-created record metadata
     * @param initialValue optional initial value for the topic which
     must have
     *         been created to match the supplied metadata
     * @param context context passed back to callback when topic
     created
     * @param callback to notify result of operation
     */
    public void addRecordTopic(
        String topicPath,
        MContent metadata,
        Content initialValue,
        String context,
        AddContextCallback<String> callback) {

        final TopicDetails details =
topicControl.newDetailsBuilder(RecordTopicDetails.Builder.class)
                .metadata(metadata).build();

        topicControl.addTopic(
            topicPath,
            details,
            initialValue,
            context,
            callback);
    }

    /**
     * Remove a single topic given its path.
     *
     * @param topicPath the topic path
     * @param callback notifies result of operation
     */
    public void removeTopic(String topicPath, RemovalCallback
callback) {
        topicControl.remove(topicPath, callback);
    }

    /**
     * Remove a topic and all of its descendants.
     *
     * @param topicPath the topic path
     * @param callback notifies result of operation
     */
    public void removeTopicBranch(String topicPath, RemovalCallback
callback) {
        topicControl.remove("'" + topicPath + "'", callback);
    }

```

```

    }

    /**
     * Remove one or more topics using a topic selector expression.
     *
     * @param topicSelector the selector expression
     * @param callback notifies result of operation
     */
    public void removeTopics(String topicSelector, RemovalCallback
callback) {
        topicControl.remove(topicSelector, callback);
    }

    /**
     * Request that the topic {@code topicPath} and its descendants
be removed
     * when the session is closed (either explicitly using {@link
Session#close}
     * , or by the server). If more than one session calls this
method for the
     * same {@code topicPath}, the topics will be removed when the
last session
     * is closed.
     *
     * <p>
     * Different sessions may call this method for the same topic
path, but not
     * for topic paths above or below existing registrations on the
same branch
     * of the topic tree.
     *
     * @param topicPath the part of the topic tree to remove when the
last
     * session is closed
     */
    public void removeTopicsWithSession(String topicPath) {
        topicControl.removeTopicsWithSession(
            topicPath, new TopicTreeHandler.Default());
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }
}

```

.NET

```

using System.Collections.Generic;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Content;
using PushTechnology.ClientInterface.Client.Content.Metadata;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Topics;

namespace Examples {
    /// <summary>

```

```

    /// An example of using a control client to add topics.
    ///
    /// This uses the <see cref="ITopicControl"/> feature only.
    ///
    /// To add or remove topics, the client session must have the
    <see cref="TopicPermission.MODIFY_TOPIC"/> permission
    /// for that branch of the topic tree.
    /// </summary>
    public class ControlClientAddingTopics {
        private readonly ISession session;
        private readonly ITopicControl topicControl;

        public ControlClientAddingTopics() {
            session =
Diffusion.Sessions.Principal( "control" ).Password( "password" )
                .Open( "ws://diffusion.example.com:80" );

            topicControl = session.GetTopicControlFeature();
        }

        /// <summary>
        /// Adds a topic with the type derived from the value.
        ///
        /// This uses the simple convenience method for adding topics
where the topic type and metadata are derived
        /// from a supplied value which can be any object. For
example, an integer would result in a single value topic
        /// of type 'integer'.
        /// </summary>
        /// <typeparam name="T">The value type.</typeparam>
        /// <param name="topicPath">The full topic path.</param>
        /// <param name="initialValue">An optional initial value for
the topic.</param>
        /// <param name="context">This will be passed back to the
callback when reporting success or failure of the
        /// topic add.</param>
        /// <param name="callback">To notify the result of the
operation.</param>
        /// <returns>The topic details used to add the topic.</
returns>
        public ITopicDetails AddTopicForValue<T>( string topicPath, T
initialValue, string context,
            ITopicControlAddContextCallback<string> callback ) {
            return topicControl.AddTopicFromValue( topicPath,
initialValue, context, callback );
        }

        /// <summary>
        /// Add a record topic from a list of initial values.
        ///
        /// This demonstrates the simplest mechanism for adding a
record topic by supplying values that both the
        /// metadata and the initial values are derived from.
        /// </summary>
        /// <param name="topicPath">The full topic path.</param>
        /// <param name="initialValues">The initial values for the
topic fields which will also be used to derive the
        /// metadata definition of the topic.</param>
        /// <param name="context">This will be passed back to the
callback when reporting success or failure of the
        /// topic add.</param>
        /// <param name="callback">To notify the result of the
operation.</param>

```

```

        /// <returns></returns>
        public ITopicDetails AddRecordTopic( string topicPath,
List<string> initialValues, string context,
        ITopicControlAddContextCallback<string> callback ) {
            return topicControl.AddTopicFromValue( topicPath,

Diffusion.Content.NewBuilder<IRecordContentBuilder>().PutFields( initialValues.
                context, callback );
        }

        /// <summary>
        /// Adds a record topic with supplied metadata and optional
        initial content.
        ///
        /// This example shows details being created and would be
        fine when creating topics that are all different, but
        /// if creating many record topics with the same details,
        then it is far more efficient to pre-create the
        /// details.
        /// </summary>
        /// <param name="topicPath">The full topic path.</param>
        /// <param name="metadata">The pre-created record metadata.</
param>
        /// <param name="initialValue">The optional initial value for
        the topic which must have been created to match
        /// the supplied metadata.</param>
        /// <param name="context">The context passed back to the
        callback when the topic is created.</param>
        /// <param name="callback">To notify the result of the
        operation.</param>
        public void AddRecordTopic( string topicPath, IMContent
        metadata, IContent initialValue, string context,
        ITopicControlAddContextCallback<string> callback ) {
            var details =
        topicControl.CreateDetailsBuilder<IRecordTopicDetailsBuilder>().Metadata( metad

                topicControl.AddTopic( topicPath, details, initialValue,
        context, callback );
        }

        /// <summary>
        /// Remove a single topic given its path.
        /// </summary>
        /// <param name="topicPath">The topic path.</param>
        /// <param name="callback">Notifies the result of the
        operation.</param>
        public void RemoveTopic( string topicPath,
ITopicControlRemoveCallback callback ) {
            topicControl.RemoveTopics( ">" + topicPath, callback );
        }

        /// <summary>
        /// Remove one or more topics using a topic selector
        expression.
        /// </summary>
        /// <param name="topicSelector">The selector expression.</
param>
        /// <param name="callback">Notifies the result of the
        operation.</param>
        public void RemoveTopics( string topicSelector,
ITopicControlRemoveCallback callback ) {
            topicControl.RemoveTopics( topicSelector, callback );
        }

```

```

        /// <summary>
        /// Request that the topic path and its descendants be
removed when the session is closed (either explicitly
        /// using <see cref="ISession.Close"/>, or by the server). If
more than one session calls this method for the
        /// same topic path, the topics will be removed when the last
session is closed.
        ///
        /// Different sessions may call this method for the same
topic path, but not for topic paths above or below
        /// existing registrations on the same branch of the topic
tree.
        /// </summary>
        /// <param name="topicPath">The part of the topic tree to
remove when the last session is closed.</param>
        public void RemoveTopicsWithSession( string topicPath ) {
            topicControl.RemoveTopicsWithSession( topicPath, new
DefaultTopicTreeHandler() );
        }

        /// <summary>
        /// Close the session.
        /// </summary>
        public void Close() {
            session.Close();
        }
    }
}

```

C

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include <apr.h>
#include <apr_thread_mutex.h>
#include <apr_thread_cond.h>

#include "diffusion.h"
#include "args.h"
#include "utils.h"

/*
 * We use a mutex and a condition variable to help synchronize the
 * flow so that it becomes linear and easier to follow the core
 * logic.
 */
apr_pool_t *pool = NULL;
apr_thread_mutex_t *mutex = NULL;
apr_thread_cond_t *cond = NULL;

ARG_OPTS_T arg_opts[] = {
    ARG_OPTS_HELP,
    {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
ARG_HAS_VALUE, "ws://localhost:8080"},
    {'p', "principal", "Principal (username) for the connection",
ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    {'c', "credentials", "Credentials (password) for the
connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    END_OF_ARG_OPTS
}

```

```

};

// Various handlers which are common to all add_topic() functions.
static int
on_topic_added(SESSION_T *session, const SVC_ADD_TOPIC_RESPONSE_T
               *response, void *context)
{
    puts("on_topic_added");
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}

static int
on_topic_add_failed(SESSION_T *session, const
                   SVC_ADD_TOPIC_RESPONSE_T *response, void *context)
{
    printf("on_topic_add_failed: %d\n", response->response_code);
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}

static int
on_topic_add_discard(SESSION_T *session, void *context)
{
    puts("on_topic_add_discard");
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}

static int
on_topic_removed(SESSION_T *session, const
                 SVC_REMOVE_TOPICS_RESPONSE_T *response, void *context)
{
    puts("on_topic_removed");
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}

static int
on_topic_remove_discard(SESSION_T *session, void *context)
{
    puts("on_topic_remove_discard");
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}
/*
 *
 */
int main(int argc, char** argv)
{
    /*
     * Standard command-line parsing.
     */
}

```

```

HASH_T *options = parse_cmdline(argc, argv, arg_opts);
if(options == NULL || hash_get(options, "help") != NULL) {
    show_usage(argc, argv, arg_opts);
    return EXIT_FAILURE;
}

char *url = hash_get(options, "url");
const char *principal = hash_get(options, "principal");
CREDENTIALS_T *credentials = NULL;
const char *password = hash_get(options, "credentials");
if(password != NULL) {
    credentials = credentials_create_password(password);
}

// Setup for condition variable
apr_initialize();
apr_pool_create(&pool, NULL);
apr_thread_mutex_create(&mutex, APR_THREAD_MUTEX_UNNESTED,
pool);
apr_thread_cond_create(&cond, pool);

// Setup for session
SESSION_T *session;
DIFFUSION_ERROR_T error = { 0 };
session = session_create(url, principal, credentials, NULL,
NULL, &error);
if(session == NULL) {
    fprintf(stderr, "TEST: Failed to create session\n");
    fprintf(stderr, "ERR : %s\n", error.message);
    return EXIT_FAILURE;
}

// Common params for all add_topic() functions.
ADD_TOPIC_PARAMS_T common_params = {
    .on_topic_added = on_topic_added,
    .on_topic_add_failed = on_topic_add_failed,
    .on_discard = on_topic_add_discard
};

/*
 * Create a stateless topic.
 */
TOPIC_DETAILS_T *topic_details =
create_topic_details_stateless();
ADD_TOPIC_PARAMS_T stateless_params = common_params;
stateless_params.topic_path = "stateless";
stateless_params.details = topic_details;

apr_thread_mutex_lock(mutex);
add_topic(session, stateless_params);
apr_thread_cond_wait(cond, mutex);
apr_thread_mutex_unlock(mutex);

/*
 * Create a topic with single value string data, but with
 * containing no default data.
 */
TOPIC_DETAILS_T *string_topic_details =
create_topic_details_single_value(M_DATA_TYPE_STRING);
ADD_TOPIC_PARAMS_T string_params = common_params;
string_params.topic_path = "string";
string_params.details = string_topic_details;

```

```

    apr_thread_mutex_lock(mutex);
    add_topic(session, string_params);
    apr_thread_cond_wait(cond, mutex);
    apr_thread_mutex_unlock(mutex);

    /*
     * Create a topic with single value string data and
containing
     * some default data.
     */
    ADD_TOPIC_PARAMS_T string_data_params = common_params;
    string_data_params.topic_path = "string-data";
    string_data_params.details = string_topic_details;
    BUF_T *sample_data_buf = buf_create();
    buf_write_string(sample_data_buf, "Hello, world");
    string_data_params.content =
content_create(CONTENT_ENCODING_NONE, sample_data_buf);

    apr_thread_mutex_lock(mutex);
    add_topic(session, string_data_params);
    apr_thread_cond_wait(cond, mutex);
    apr_thread_mutex_unlock(mutex);

    /*
     * Create a topic with single value integer data, and with a
     * default value.
     */
    TOPIC_DETAILS_T *integer_topic_details =
create_topic_details_single_value(M_DATA_TYPE_INTEGER_STRING);
    integer_topic_details-
>topic_details_params.integer.default_value = 999;

    ADD_TOPIC_PARAMS_T integer_params = common_params;
    integer_params.topic_path = "integer";
    integer_params.details = integer_topic_details;

    apr_thread_mutex_lock(mutex);
    add_topic(session, integer_params);
    apr_thread_cond_wait(cond, mutex);
    apr_thread_mutex_unlock(mutex);

    /*
     * Create a topic with integer data, but using a CONTENT_T to
     * specify the initial data.
     */
    ADD_TOPIC_PARAMS_T integer_data_params = common_params;
    integer_data_params.topic_path = "integer-data";
    integer_data_params.details = integer_topic_details;
    BUF_T *integer_data_buf = buf_create();
    buf_sprintf(integer_data_buf, "%d", 123);
    integer_data_params.content =
content_create(CONTENT_ENCODING_NONE, integer_data_buf);

    apr_thread_mutex_lock(mutex);
    add_topic(session, integer_data_params);
    apr_thread_cond_wait(cond, mutex);
    apr_thread_mutex_unlock(mutex);

    /*
     * Create a topic with single value decimal data, with a
     * default value and specifying the scale (i.e. positions
     * after the decimal place).
     */

```

```

        TOPIC_DETAILS_T *decimal_topic_details =
create_topic_details_single_value(M_DATA_TYPE_DECIMAL_STRING);
        decimal_topic_details-
>topic_details_params.decimal.default_value = 123.456;
        decimal_topic_details->topic_details_params.decimal.scale =
4;

        ADD_TOPIC_PARAMS_T decimal_params = common_params;
        decimal_params.topic_path = "decimal";
        decimal_params.details = decimal_topic_details;

        apr_thread_mutex_lock(mutex);
        add_topic(session, decimal_params);
        apr_thread_cond_wait(cond, mutex);
        apr_thread_mutex_unlock(mutex);

        /*
        * Create a topic with decimal data, using a CONTENT_T to
        * specify the initial data.
        */
        ADD_TOPIC_PARAMS_T decimal_data_params = common_params;
        decimal_data_params.topic_path = "decimal-data";
        decimal_data_params.details = decimal_topic_details;
        BUF_T *decimal_data_buf = buf_create();
        buf_sprintf(decimal_data_buf, "%f", 987.654);
        decimal_data_params.content =
content_create(CONTENT_ENCODING_NONE, decimal_data_buf);

        apr_thread_mutex_lock(mutex);
        add_topic(session, decimal_data_params);
        apr_thread_cond_wait(cond, mutex);
        apr_thread_mutex_unlock(mutex);

        /*
        * Record topic data.
        *
        * The C API does not have the concept of "builders" for
        * creating record topic data, but requires you to build a
        * string containing XML that describes the structure of the
        * messages.
        */

        /*
        * First of all, this adds a topic equivalent to single-value
        * strings, but defined with XML.
        */
        BUF_T *manual_schema = buf_create();
        buf_write_string(manual_schema,
                "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=
\"yes\"?>\n");
        buf_write_string(manual_schema,
                "<field name=\"x\" type=\"string\" default=\"xyzy\"
allowsEmpty=\"true\"/>");
        TOPIC_DETAILS_T *manual_topic_details =
create_topic_details_single_value(M_DATA_TYPE_STRING);
        manual_topic_details->user_defined_schema = manual_schema;

        ADD_TOPIC_PARAMS_T string_manual_params = common_params;
        string_manual_params.topic_path = "string-manual";
        string_manual_params.details = manual_topic_details;

        apr_thread_mutex_lock(mutex);
        add_topic(session, string_manual_params);

```

```

    apr_thread_cond_wait(cond, mutex);
    apr_thread_mutex_unlock(mutex);

    /*
     * This adds a topic with a record containing multiple fields
     * of different types.
     */
    BUF_T *record_schema = buf_create();
    buf_write_string(record_schema,
        "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=
\"yes\"?>");
    buf_write_string(record_schema,
        "<message topicDataType=\"record\" name=\"MyContent
\">");
    buf_write_string(record_schema,
        "<record name=\"Record1\">");
    buf_write_string(record_schema,
        "<field type=\"string\" default=\"\" allowsEmpty=
\"true\" name=\"Field1\"/>");
    buf_write_string(record_schema,
        "<field type=\"integerString\" default=\"0\"
allowsEmpty=\"false\" name=\"Field2\"/>");
    buf_write_string(record_schema,
        "<field type=\"decimalString\" default=\"0.00\"
scale=\"2\" allowsEmpty=\"false\" name=\"Field3\"/>");
    buf_write_string(record_schema,
        "</record>");
    buf_write_string(record_schema,
        "</message>");
    TOPIC_DETAILS_T *record_topic_details =
create_topic_details_record();
    record_topic_details->user_defined_schema = record_schema;

    ADD_TOPIC_PARAMS_T record_params = common_params;
    record_params.topic_path = "record";
    record_params.details = record_topic_details;

    apr_thread_mutex_lock(mutex);
    add_topic(session, record_params);
    apr_thread_cond_wait(cond, mutex);
    apr_thread_mutex_unlock(mutex);

    /*
     * Create a topic with binary data
     */
    TOPIC_DETAILS_T *binary_topic_details =
create_topic_details_binary();
    ADD_TOPIC_PARAMS_T binary_params = common_params;
    binary_params.topic_path = "binary-data";
    binary_params.details = binary_topic_details;

    char binary_bytes[] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05 };

    BUF_T *binary_data_buf = buf_create();
    buf_write_bytes(binary_data_buf, binary_bytes,
sizeof(binary_bytes));
    binary_params.content = content_create(CONTENT_ENCODING_NONE,
binary_data_buf);

    apr_thread_mutex_lock(mutex);
    add_topic(session, binary_params);
    apr_thread_cond_wait(cond, mutex);
    apr_thread_mutex_unlock(mutex);

```

```

/*
 * We can also remove topics. First, add a couple of topics
 * and then remove their parent topic. All child topics are
 * removed with the parent.
 */
puts("Adding topics remove_me/1 and remove_me/2");

ADD_TOPIC_PARAMS_T topic_params = common_params;
topic_params.details = topic_details;
topic_params.topic_path = "remove_me/1";

apr_thread_mutex_lock(mutex);
add_topic(session, topic_params);
apr_thread_cond_wait(cond, mutex);
apr_thread_mutex_unlock(mutex);

topic_params.topic_path = "remove_me/2";
apr_thread_mutex_lock(mutex);
add_topic(session, topic_params);
apr_thread_cond_wait(cond, mutex);
apr_thread_mutex_unlock(mutex);

puts("Removing topics in 5 seconds...");
sleep(5);

REMOVE_TOPICS_PARAMS_T remove_params = {
    .on_removed = on_topic_removed,
    .on_discard = on_topic_remove_discard,
    .topic_selector = ">remove_me"
};

apr_thread_mutex_lock(mutex);
remove_topics(session, remove_params);
apr_thread_cond_wait(cond, mutex);
apr_thread_mutex_unlock(mutex);

/*
 * Close our session, and release resources and memory.
 */
session_close(session, NULL);
session_free(session);

apr_thread_mutex_destroy(mutex);
apr_thread_cond_destroy(cond);
apr_pool_destroy(pool);
apr_terminate();

return EXIT_SUCCESS;
}

```

Change the URL from that provided in the example to the URL of the Diffusion server.

Creating a metadata definition for a record topic

You can use the Unified API to specify the metadata structure that describes the byte data content of a message.

About this task

Publishing clients define the metadata structure for messages. This metadata structure can be used when defining a topic. All messages placed on the topic must conform to the metadata structure.

The Unified API for the following platforms provides builder methods that enable you to define the metadata structure:

- JavaScript
- Java
- .NET

The C Unified API does not provide builder methods, but instead takes the definition of the metadata as XML. For more information, see [C API overview](#).

The following example demonstrates how to define the metadata structure using the Java Unified API.

Procedure

1. Define the metadata structure.

- a) Import `com.pushtechnology.diffusionclient.Diffusion` and the following classes from the `com.pushtechnology.diffusion.content.metadata` package:

- `MetadataFactory`
- `MContent`
- `MRecord`
- `MField`
- `MString`
- `MIntegerString`
- `MDecimalString`
- `MCustomString`

- b) Use the `Diffusion.metadata` method to get a `MetadataFactory`.

```
private final MetadataFactory factory = Diffusion.metadata();
```

- c) Use the methods on the `MetadataFactory` to specify the content, record, and field definitions that make up the metadata structure.

For example, the following code uses a content builder to create content metadata with a single record type that can occur zero to n times.

```
public MContent createContentRepeating() {
    return
        factory.contentBuilder("Content").
            add(
                factory.record(
                    "Rec1",
                    factory.string("A"),
                    factory.string("B")),
                0,
                -1).
            build();
}
```

```
}
```

For more information, see [Java Unified API documentation](#).

2. Create a record topic and apply the metadata definition to it.

- a) Import the TopicControl feature, Session class, and RecordTopicDetails class.

```
import
    com.pushtechonology.diffusion.client.features.control.topics.TopicControl;
import com.pushtechonology.diffusion.client.session.Session;
import
    com.pushtechonology.diffusion.client.topics.details.RecordTopicDetails;
```

- b) Create a Session instance and use it to get the TopicControl feature.

```
final Session session = Diffusion.sessions().open("ws://
diffusion.example.com:80");
final TopicControl tc = session.feature(TopicControl.class);
```

- c) Get a topic builder for a record topic from the TopicControl feature.

```
final RecordTopicDetails.Builder builder =
    tc.newDetailsBuilder(RecordTopicDetails.Builder.class);
```

- d) Use the metadata method of the topic builder to create the topic definition.

```
tc.addTopic(
    TOPIC_NAME,
    builder.metadata(metadata).build(),
    new TopicControl.AddCallback.Default() {
        @Override
        public void onTopicAdded(String topic) {
            theTopic = topic;
        }
    });
```

Example: A client that creates a metadata definition and uses it when creating a topic.

```
package com.example.metadata;

import com.pushtechonology.diffusion.client.Diffusion;
import
    com.pushtechonology.diffusion.client.content.metadata.MContent;
import
    com.pushtechonology.diffusion.client.content.metadata.MDecimalString;
import
    com.pushtechonology.diffusion.client.content.metadata.MField;
import
    com.pushtechonology.diffusion.client.content.metadata.MRecord;
import
    com.pushtechonology.diffusion.client.content.metadata.MetadataFactory;
import
    com.pushtechonology.diffusion.client.topics.details.TopicType;
import
    com.pushtechonology.diffusion.client.features.control.topics.TopicControl;
import com.pushtechonology.diffusion.client.session.Session;
import
    com.pushtechonology.diffusion.client.topics.details.RecordTopicDetails;
import com.pushtechonology.diffusion.client.types.UpdateOptions;
```

```

/**
 * An example of a client creating metadata definition and using
 * it when creating a
 * topic definition.
 */
public class ControlClient {

    private final static String TOPIC_NAME = "foo/record";
    private String theTopic;
    private final MetadataFactory factory = Diffusion.metadata();

    /**
     * Creates control client instance.
     */
    public ControlClient() {

        // Create the Session
        final Session session = Diffusion.sessions()
            .open("ws://diffusion.example.com:80");

        // Add the TopicControl feature
        final TopicControl tc =
            session.feature(TopicControl.class);

        // Create metadata for the topic
        final MContent metadata = defineMetadata();

        // Get a topic builder
        final RecordTopicDetails.Builder builder =
            tc.newDetailsBuilder(RecordTopicDetails.Builder.class);

        // Create the topic with metadata
        tc.addTopic(
            TOPIC_NAME,
            builder.metadata(metadata).build(),
            new TopicControl.AddCallback.Default() {
                @Override
                public void onTopicAdded(String topic) {
                    theTopic = topic;
                }
            }
        );

    }

    /**
     * A simple example of creating content metadata with two
     * records.
     *
     * @return content metadata
     */
    public MContent defineMetadata() {
        return factory.content(
            "Content",
            createNameAndAddressRecord(),
            createMultipleRateCurrencyRecord("Exchange Rates",
2));
    }

    /**

```

```

    * Creates a simple name and address record with fixed name
single
    * multiplicity fields.
    *
    * @return record definition.
    */
    public MRecord createNameAndAddressRecord() {
        return factory.record(
            "NameAndAddress",
            factory.string("FirstName"),
            factory.string("Surname"),
            factory.string("HouseNumber"),
            factory.string("Street"),
            factory.string("Town"),
            factory.string("State"),
            factory.string("PostCode"));
    }

    /**
    * This creates a record with two fields, a string called
    "Currency" and a
    * decimal string called "Rate" with a default value of 1.00
    which repeats a
    * specified number of times.
    *
    * @param name the record name
    * @param occurs the number of occurrences of the "Rate" field
    * @return the metadata record
    */
    public MRecord createMultipleRateCurrencyRecord(String name,
int occurs) {
        return factory.recordBuilder(name).
            add(factory.string("Currency")).
            add(factory.decimal("Rate", "1.00"), occurs).
            build();
    }
}

```

Handling subscriptions to missing topics

A client can use the TopicControl feature of the Unified API to handle subscription or fetch requests for topics that do not exist.

Registering a missing topic handler and dynamically adding topics

Required permissions: modify_topic, register_handler

You can use the TopicControl feature to dynamically create topics on demand when a client tries to subscribe or fetch a topic that does not exist.

The client can register itself as a handler for missing topics for any part of the topic tree. The client is notified of attempts to subscribe to or fetch topics that are subordinate to that topic and that do not exist. This enables the client to create the topics and notify the Diffusion server that the client operation subscribe or fetch can proceed.

Note: The handler is called only when a client attempts to subscribe or fetch using a single topic path. If another type of selector is used to subscribe to or fetch the topic, the handler is not notified.

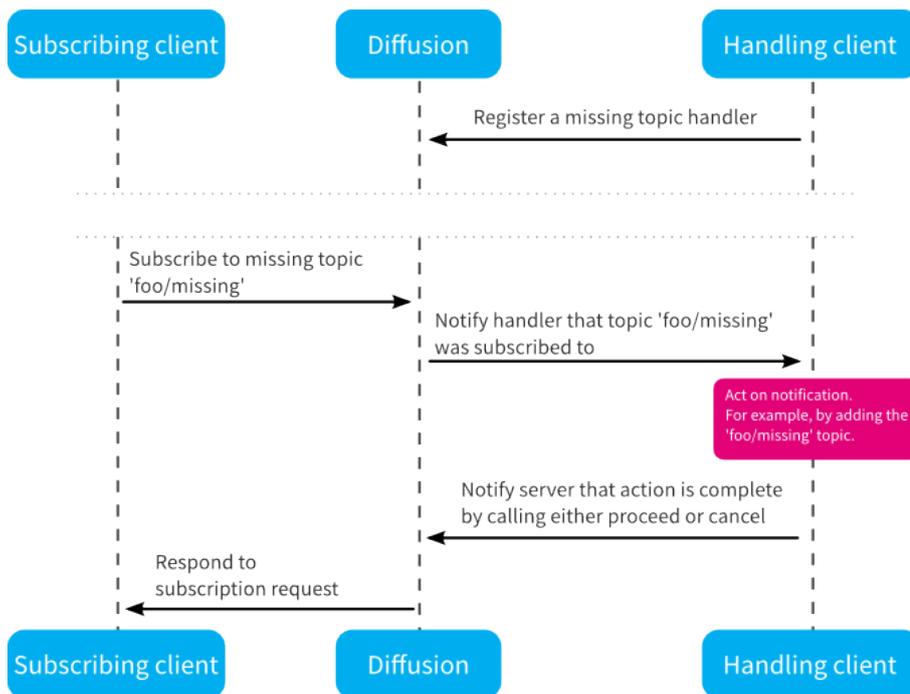


Figure 24: Flow from a subscribing client to the client that handles a missing topic subscription

The missing topic handler is removed when the registering session is closed. If the registering session loses connection, it goes into DISCONNECTED state. When in DISCONNECTED state the handler remains active but cannot pass on the notifications to the client. In this case, cancel or proceed callbacks for these notifications might not function as expected because of timeouts. If the client then closes, these notifications are discarded.

To ensure that missing topic notifications are always received by your solution, you can use multiple clients to register missing topic handlers. Ensure that if any of these clients lose connection they go straight to CLOSED state by setting the reconnection timeout to zero. When the client loses connect it closes straight away, the handler is registered, and further missing topic notifications are routed to a handler registered by another client.

Related concepts

[Using missing topic notifications with fan-out](#) on page 102

Missing topic notifications generated by subscription or fetch requests to a secondary server are propagated to missing topic handlers registered against the primary servers.

Example: Receive missing topic notifications

The following examples use the TopicControl feature in the Unified API to register a missing topic notification handler.

JavaScript

```

var diffusion = require('diffusion');

// Connect to the server. Change these options to suit your own
// environment.
// Node.js will not accept self-signed certificates by default. If
// you have
  
```

```

// one of these, set the environment variable
NODE_TLS_REJECT_UNAUTHORIZED=0
// before running this example.
diffusion.connect({
  host    : 'diffusion.example.com',
  port    : 443,
  secure  : true
}).then(function(session) {

  // Register a missing topic handler on the "example" root topic
  // Any subscriptions to missing topics along this path will invoke
  this handler
  session.topics.addMissingTopicHandler("example", {
    // Called when a handler is successfully registered
    onRegister : function(path, close) {
      console.log("Registered missing topic handler on path: " + path);
      // Once we've registered the handler, we initiate a subscription
      with the selector "?example/topic/.*"
      // This will invoke the handler.
      session.subscribe("?example/topic/.*").on('subscribe',
        function(type, path) {
          console.log("Subscribed to topic: " + path);
        });
    },
    // Called when the handler is closed
    onClose : function(path) {
      console.log("Missing topic handler on path '" + path + "' has been
        closed");
    },
    // Called if there is an error on the handler
    onError : function(path, error) {
      console.log("Error on missing topic handler");
    },
    // Called when we've received a missing topic notification on our
    registered handler path
    onMissingTopic : function(notification) {
      console.log("Received missing topic notification with selector: "
        + notification.selector);
      // Once we've received the missing topic notification initiated
      from subscribing to "?example/topic/.*",
      // we add a topic that will match the selector

      var topic = "example/topic/foo";

      session.topics.add(topic).then(function(result) {
        console.log("Topic add success: " + topic);
        // If the topic addition is successful, we proceed() with the
        session's subscription.
        // The client will now be subscribed to the topic
        notification.proceed();
      }, function(reason) {
        console.log("Topic add failed: " + reason);
        // If the topic addition fails, we cancel() the session's
        subscription request.
        notification.cancel();
      });
    }
  });
});

```

Apple

```
@import Diffusion;

@interface MissingTopicHandlerExample
    (PTDiffusionMissingTopicHandler) <PTDiffusionMissingTopicHandler>
@end

@implementation MissingTopicHandlerExample {
    PTDiffusionSession* _session;
}

-(void)startWithURL:(NSURL*)url {
    PTDiffusionCredentials *const credentials =
        [[PTDiffusionCredentials alloc]
         initWithPassword:@"password"];

    PTDiffusionSessionConfiguration *const sessionConfiguration =
        [[PTDiffusionSessionConfiguration alloc]
         initWithPrincipal:@"control"
         credentials:credentials];

    NSLog(@"Connecting...");

    [PTDiffusionSession openWithURL:url
                        configuration:sessionConfiguration
                        completionHandler:^(PTDiffusionSession *session,
                                           NSError *error)
     {
        if (!session) {
            NSLog(@"Failed to open session: %@", error);
            return;
        }

        // At this point we now have a connected session.
        NSLog(@"Connected.");

        // Set ivar to maintain a strong reference to the session.
        _session = session;

        // Register as missing topic handler for a branch of the
        topic tree.
        [self registerAsMissingTopicHandlerForSession:session];
    }];
}

-(void)registerAsMissingTopicHandlerForSession:(PTDiffusionSession
*const)session {
    [session.topicControl addMissingTopicHandler:self
                        forTopicPath:@"Example/Control
Client Handler"

completionHandler:^(PTDiffusionTopicTreeRegistration *const
registration, NSError *const error)
    {
        if (registration) {
            NSLog(@"Registered as missing topic handler.");
        } else {
            NSLog(@"Failed to register as missing topic handler.
Error: %@", error);
        }
    }];
};
```

```

}

@end

@implementation MissingTopicHandlerExample
    (PTDiffusionMissingTopicHandler)

    -(void)diffusionTopicTreeRegistration:
    (PTDiffusionTopicTreeRegistration *const)registration
        hadMissingTopicNotification:
    (PTDiffusionMissingTopicNotification *const)notification {
        NSString *const expression =
        notification.topicSelectorExpression;
        NSLog(@"Received Missing Topic Notification: %@", expression);

        // Expect a path pattern expression.
        if (![expression hasPrefix:@">"]) {
            NSLog(@"Topic selector expression is not a path pattern.");
            return;
        }

        // Extract topic path from path pattern expression.
        NSString *const topicPath = [expression substringFromIndex:1];

        // Add a stateless topic at this topic path.
        [_session.topicControl addWithTopicPath:topicPath

        type:PTDiffusionTopicType_Stateless
            value:nil
            completionHandler:^(NSError *const error)
        {
            if (error) {
                NSLog(@"Failed to add topic.");
                return;
            }

            // Topic added so allow subscriber to proceed.
            [notification proceed];
        }];
    }

@end

```

Java and Android

```

package com.pushtechology.diffusion.examples;

import com.pushtechology.diffusion.client.Diffusion;
import
    com.pushtechology.diffusion.client.features.RegisteredHandler;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicAddFailReason;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicControl;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicControl.MissingTopic;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicControl.MissingTopicReason;
import com.pushtechology.diffusion.client.session.Session;
import
    com.pushtechology.diffusion.client.topics.details.TopicDetails;
import com.pushtechology.diffusion.client.topics.details.TopicType;

```

```

/**
 * An example of registering a missing topic notification handler and
 * processing
 * notifications using a control client.
 *
 * @author Push Technology Limited
 */
public final class ControlClientHandlingMissingTopicNotification {

    // UCI features
    private final Session session;
    private final TopicControl topicControl;
    private final TopicDetails details;

    /**
     * Constructor.
     */
    public ControlClientHandlingMissingTopicNotification() {
        // Create a session
        session =
Diffusion.sessions().password("password").principal("admin").open("ws://
diffusion.example.com:8080");

        topicControl = session.feature(TopicControl.class);

        // Registers a missing topic notification on a topic path
        topicControl.addMissingTopicHandler("A", new
MissingTopicNotificationHandler());

        // For details that may be reused many times it is far more
efficient to
        // create just once - this creates a default string type
details.
        details = topicControl.newDetails(TopicType.SINGLE_VALUE);
    }

    // Private class that implements MissingTopicHandler interface
    private final class MissingTopicNotificationHandler implements
MissingTopicHandler {
        /**
         * @param topicPath
         * - the path that the handler is active for
         * @param registeredHandler
         * - allows the handler to be closed
         */
        @Override
        public void onActive(String topicPath, RegisteredHandler
registeredHandler) {
        }

        /**
         * @param topicPath
         * - the branch of the topic tree for which the
handler was
         * registered
         */
        @Override
        public void onClose(String topicPath) {
        }
    }
}

```

```

        * @param notification
        *         - the missing topic details
        */
        @Override
        public void onMissingTopic(MissingTopicNotification
notification) {
            // Create a topic and do process() in the callback
            topicControl.addTopic(notification.getTopicPath(),
details, new AddTopicCallback(notification));
        }
    }

    private final class AddTopicCallback implements
TopicControl.AddCallback {
        private final MissingTopicNotification notification;

        AddTopicCallback(MissingTopicNotification notification) {
            this.notification = notification;
        }

        @Override
        public void onDiscard() {
        }

        /**
         * @param topicPath
         *         - the topic path as supplied to the add request
         * @param reason
         *         - the reason for failure
         */
        @Override
        public void onTopicAddFailed(String topicPath,
TopicAddFailReason reason) {
            // Cancel the notification because the server have failed
to
            notification.cancel();
        }

        /**
         * @param topicPath
         *         - the full path of the topic that was added
         */
        @Override
        public void onTopicAdded(String topicPath) {
            // Proceed the notification
            notification.proceed();
        }
    }
}
}

```

.NET

```

using System.Threading.Tasks;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;

namespace Examples {
    public class ControlClientMissingTopicNotification {

```

```

private readonly ISession clientSession;
private readonly ITopicControl topicControl;

public ControlClientMissingTopicNotification() {
    clientSession =
Diffusion.Sessions.Principal( "client" ).Password( "password" )
        .Open( "ws://diffusion.example.com:80" );

    topicControl =
Diffusion.Sessions.Principal( "control" ).Password( "password" )
        .Open( "ws://
diffusion.example.com:80" ).GetTopicControlFeature();

    Subscribe( "some/path10" );
}

/// <summary>
/// Subscribes to a topic which may or may not be missing.
/// </summary>
/// <param name="topicPath">The path of the topic to
subscribe to.</param>
public async void Subscribe( string topicPath ) {
    var missingTopicHandler = new MissingTopicHandler();

    // Add the 'missing topic handler' to the topic control
object
    topicControl.AddMissingTopicHandler( topicPath,
missingTopicHandler );

    // Wait for the successful registration of the handler
var registeredHandler = await
missingTopicHandler.OnActiveCalled;

    var topics = clientSession.GetTopicsFeature();

    var topicCompletion = new TaskCompletionSource<bool>();

    // Attempt to subscribe to the topic
topics.Subscribe( topicPath, new
TopicsCompletionCallback( topicCompletion ) );

    await topicCompletion.Task;

    // Wait and see if a missing topic notification is
generated
    var request = await
missingTopicHandler.OnMissingTopicCalled;

    // Cancel the client request on the server
request.Cancel();

    // Close the registered handler
registeredHandler.Close();

    // All events in Diffusion are asynchronous, so we must
wait for the close to happen
    await missingTopicHandler.OnCloseCalled;
}

private class TopicsCompletionCallback :
ITopicsCompletionCallback {
    private readonly TaskCompletionSource<bool>
theCompletionSource;

```

```

        public
        TopicsCompletionCallback( TaskCompletionSource<bool> source ) {
            theCompletionSource = source;
        }

        /// <summary>
        /// This is called to notify that a call context was
        closed prematurely, typically due to a timeout or the
        /// session being closed. No further calls will be made
        for the context.
        /// </summary>
        public void OnDiscard() {
            theCompletionSource.SetResult( false );
        }

        /// <summary>
        /// Called to indicate that the requested operation has
        been processed by the server.
        /// </summary>
        public void OnComplete() {
            theCompletionSource.SetResult( true );
        }
    }

    /// <summary>
    /// Asynchronous helper class for handling missing topic
    notifications.
    /// </summary>
    private class MissingTopicHandler : IMissingTopicHandler {
        private readonly TaskCompletionSource<IRegisteredHandler>
        onActive =
            new TaskCompletionSource<IRegisteredHandler>();

        private readonly
        TaskCompletionSource<IMissingTopicNotification> onMissingTopic =
            new
            TaskCompletionSource<IMissingTopicNotification>();

        private readonly TaskCompletionSource<bool> onClose = new
        TaskCompletionSource<bool>();

        /// <summary>
        /// Waits for the 'OnActive' event to be called.
        /// </summary>
        public Task<IRegisteredHandler> OnActiveCalled {
            get {
                return onActive.Task;
            }
        }

        /// <summary>
        /// Waits for the 'OnMissingTopic' event to be called.
        /// </summary>
        public Task<IMissingTopicNotification>
        OnMissingTopicCalled {
            get {
                return onMissingTopic.Task;
            }
        }

        public Task OnCloseCalled {
            get {

```


C

```

/*
 * This example shows how to register a missing topic notification
 * handler and return a missing topic notification response - calling
 * missing_topic_proceed() once we've created the topic.
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

#include <apr.h>
#include <apr_thread_mutex.h>
#include <apr_thread_cond.h>

#include "diffusion.h"
#include "args.h"

ARG_OPTS_T arg_opts[] = {
    ARG_OPTS_HELP,
    {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
    ARG_HAS_VALUE, "ws://localhost:8080"},
    {'p', "principal", "Principal (username) for the connection",
    ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    {'c', "credentials", "Credentials (password) for the
    connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    {'r', "topic_root", "Topic root to process missing topic
    notifications on", ARG_OPTIONAL, ARG_HAS_VALUE, "foo"},
    END_OF_ARG_OPTS
};

static int
on_topic_added(SESSION_T *session, const SVC_ADD_TOPIC_RESPONSE_T
 *response, void *context)
{
    puts("Topic added");
    return HANDLER_SUCCESS;
}

static int
on_topic_add_failed(SESSION_T *session, const
 SVC_ADD_TOPIC_RESPONSE_T *response, void *context)
{
    puts("Topic add failed");
    printf("Reason code: %d\n", response->reason);
    return HANDLER_SUCCESS;
}

static int
on_topic_add_discard(SESSION_T *session, void *context)
{
    puts("Topic add discarded");
    return HANDLER_SUCCESS;
}

/*
 * A request has been made for a topic that doesn't exist; create it
 * and inform Diffusion that the client's subscription request can
 * proceed.
 */
static int

```

```

on_missing_topic(SESSION_T *session, const
SVC_MISSING_TOPIC_REQUEST_T *request, void *context)
{
    printf("Missing topic: %s\n", request->topic_selector);

    BUF_T *sample_data_buf = buf_create();
    buf_write_string(sample_data_buf, "Hello, world");

    // Add the missing topic.
    ADD_TOPIC_PARAMS_T topic_params = {
        .on_topic_added = on_topic_added,
        .on_topic_add_failed = on_topic_add_failed,
        .on_discard = on_topic_add_discard,
        .topic_path = strdup(request->topic_selector+1),
        .details =
create_topic_details_single_value(M_DATA_TYPE_STRING),
        .content = content_create(CONTENT_ENCODING_NONE,
sample_data_buf)
    };

    add_topic(session, topic_params);

    // Proceed with the client's subscription to the topic
missing_topic_proceed(session, (SVC_MISSING_TOPIC_REQUEST_T
*) request);

    return HANDLER_SUCCESS;
}

/*
 * Entry point for the example.
 */
int
main(int argc, char **argv)
{
    /*
     * Standard command-line parsing.
     */
    HASH_T *options = parse_cmdline(argc, argv, arg_opts);
    if(options == NULL || hash_get(options, "help") != NULL) {
        show_usage(argc, argv, arg_opts);
        return EXIT_FAILURE;
    }

    const char *url = hash_get(options, "url");
    const char *principal = hash_get(options, "principal");
    const char *topic_root = hash_get(options, "topic_root");

    CREDENTIALS_T *credentials = NULL;
    const char *password = hash_get(options, "credentials");
    if(password != NULL) {
        credentials = credentials_create_password(password);
    }

    SESSION_T *session;
    DIFFUSION_ERROR_T error = { 0 };

    session = session_create(url, principal, credentials, NULL,
NULL, &error);
    if(session != NULL) {
        printf("Session created (state=%d, id=%s)\n",
            session_state_get(session),
            session_id_to_string(session->id));
    }
}

```

```

    }
    else {
        printf("Failed to create session: %s\n",
error.message);
        free(error.message);
        return EXIT_FAILURE;
    }

    /*
     * Register the missing topic handler
     */
    MISSING_TOPIC_PARAMS_T handler = {
        .on_missing_topic = on_missing_topic,
        .topic_path = topic_root,
        .context = NULL
    };

    missing_topic_register_handler(session, handler);

    /*
     * Run for 5 minutes.
     */
    sleep(5 * 60);

    /*
     * Close session and clean up.
     */
    session_close(session, NULL);
    session_free(session);

    hash_free(options, NULL, free);

    return EXIT_SUCCESS;
}

```

Change the URL from that provided in the example to the URL of the Diffusion server.

Listening for topic events

A client can use the TopicControl feature of the Unified API to listen for events that happen on topics in a specific topic branch.

Registering a topic event listener

Required permissions: register_handler

You can use the TopicControl feature to receive a notification whenever one of the following topic events occurs:

- A topic that previously had zero subscribers gains one or more subscribers
- A topic that previously had one or more subscribers goes to zero subscribers

Note: Subscriber numbers also include indirect subscriptions, for example: subscriptions via a slave topic or routing topic; subscriptions by another Diffusion server for fan-out replication; or replication of the topic using high-availability replication.

A client can register a topic event listener against any branch of the topic tree. When a topic event occurs on one of the topics in that branch of the topic tree the listening client receives a notification.

If multiple clients register listeners against the same branch of the topic tree, all receive notifications. However, if a client registers a listener at a more specific branch of the topic tree, the most specific listener or listeners receive a notification and any less specific listeners within that same branch do not receive a notification.

For example: Client One registers a topic event listener against A, Client Two registers a topic event listener against A, and Client Three registers a topic event listener against A/B/C. If a topic event occurs on A/B, both Client One and Client Two receive notifications. If a topic event occurs on A/B/C/D, only Client Three receives a notification.

Removing topics with sessions

A client can specify that the Diffusion server removes a topic or topics after the client session closes or fails.

Required permissions: `modify_topic`

When a client registers that a branch of the topic tree be removed when its session closes, the following events occur:

1. The client registers the removal request on a branch of the topic tree and passes in a topic tree handler.
 - The removal request is registered against a topic path. This is a path that identifies a branch of the topic tree, for example `foo/bar`. The removal request is registered for the branch of the topic tree, for example the topics `foo/bar/baz` and `foo/bar/fred/qux` are included in the specified branch of the topic tree.
 - You cannot register a removal request above or below an existing removal request. For example, if a client has registered a removal request against `foo/bar/fred` another client cannot register a removal request against `foo/bar` or `foo/bar/fred/qux`.
2. The server validates the request and gives one of the following responses:
 - If the request is not valid, the Diffusion server calls the `onError` callback of the topic tree handler.

For example, a registration request is not valid if it registers against a topic branch that is above or below a branch where an existing removal request is registered.
 - If the request is valid, the Diffusion server calls the `onActive` callback of the topic tree handler and provides a `Registration` object to the client.
3. If the client wants to deregister a removal request, it can call the `onClose` method of the `Registration` object for that removal request.
4. Other clients can register removal requests against a topic that already has a removal request registered against it. For example, if one session on the Diffusion server has registered a removal request against `foo/bar/baz`, another session on the Diffusion server can also register a removal request against `foo/bar/baz`.
5. When a client session closes or fails, if it has registered removal requests, one of the following things happens:
 - If there are still open sessions that have removal requests for the same branch of the topic tree, the Diffusion server takes no action.
 - If there are no open sessions that have removal requests for that branch of the topic tree, the Diffusion server removes all topics in that branch of the topic tree.

Note: The client session must be in a closed state for a removal request to be acted upon. If a client becomes disconnected, the removal request is not acted upon until the reconnection timeout has elapsed and the client session is closed.

Remove topic requests and topic replication

If all sessions on a Diffusion server that have a removal request for a branch of the topic tree close, the topics are removed even if that topic is replicated and sessions on other Diffusion servers have removal requests registered against that part of the tree. When the topics are removed on the server, that change is replicated to all other servers that participate in replication for these topics.

Updating topics

A client can use the `TopicUpdateControl` feature to update topics.

A client can update a topic in one of the following ways:

Exclusive updating

By registering with the Diffusion server as an update source for the branch of the topic tree that contains the topic to be updated.

If a client is registered as the active update source for a branch of the topic tree, no other clients can update topics in that branch of the topic tree.

You must use an exclusive update source with a value updater to benefit from delta updating.

Non-exclusive updating

By getting a non-exclusive updater from the `TopicUpdateControl` feature. This updater can be used to update any topic that does not already have an active update source registered against it.

Registering as an exclusive update source

Required permissions: `update_topic`, `register_handler`

A client must register as an update source for a branch of the topic tree to be able to exclusively publish content to topics in that branch. This locks the branch of the topic tree and prevents other clients from publishing updates to topics in the branch.

When a client registers as an update source the following events occur:

1. The client requests to register as an update source on a branch of the topic tree.
 - The update source is registered against a topic path. This is a path that identifies a branch of the topic tree, for example `foo/bar`. The update source is registered as a source for that branch of the topic tree, for example the topics `foo/bar/baz` and `foo/bar/fred/qux` are included in the specified branch of the topic tree.
 - You cannot register an update source above or below an existing update source. For example, if a client has registered an update source against `foo/bar/fred` another client cannot register an update source against `foo/bar` or `foo/bar/fred/qux`.
 - You can register an update source against a topic owned by an existing publisher or a topic that has an update source created by the server that is used for topic failover.
2. The server validates the registration request and returns one of the following responses:
 - If the request is valid, the Diffusion server calls the `OnRegister` callback of the update source and passes a `RegisteredHandler` that you can use to deregister the update source.
 - If the request is not valid, the Diffusion server calls the `onClose` callback of the update source.

For example, a registration request is not valid if it registers against a topic branch that is above or below a branch where an existing update source is registered.
3. When the update source is registered, the Diffusion server calls one of the following callbacks:

- If the update source is the primary update source, the Diffusion server calls the `onActive` callback of the update source.
 - If another update source is already the primary source for this branch of the topic tree, the Diffusion server calls the `onStandby` callback of the update source.
4. If an update source is on standby, the update source cannot update the topics it is registered against. If the active update source for a branch of the topic tree closes or becomes inactive, a standby update source can then become active and become the primary update source for that branch of the topic tree.
 5. If an update source is active, the Diffusion server provides the update source with an `Updater`. The update source can use the `Updater` to update the topics it is registered against.
 6. If an active update source exists for a branch of the topic tree, no other clients can update topics in that branch of the topic tree.

Updating a topic non-exclusively

Required permissions: `update_topic`

To non-exclusively update topics, a client must get a non-exclusive updater from the `TopicUpdateControl` feature. This updater can be used to update any topic under the following conditions:

- The topic does not already have an active update source registered against it
- The client has the `update_topic` permission for the topic

Types of updater

Updater type	Description
Value updater	Use a value updater to update one of the following topic types: JSON, binary. In future releases, more topic types will use the value updaters. If you are using an exclusive update source, a value updater can calculate the delta of change between two values and send only that to the Diffusion server, reducing the data volume.
Updater	Use an updater to update one of the following topic types: single value, record, stateless, custom, protocol buffer, paged.

Using a value updater to stream values through topics

Required permissions: `update_topic`

A client uses a value updater to publish a value to a topic. Value updaters are typed and can only be used to update topics whose data type matches the data type of the value updater.

Value updaters can be used for exclusive or non-exclusive updating, depending on how the value updater is acquired.

When used exclusively, value updaters cache the values that are passed to them. When a value is passed to a value updater, the value updater compared that value with the previously cached value. If it is more efficient to do so, the value updater publishes a delta of changes between the previous value and the new value instead of publishing the full new value.

For non-exclusive updating, the complete value is always sent to the server and the value is not cached.

When the client uses a value updater method to publish values, it passes in the following parameters:

Topic path

The path to the topic to be updated.

If the value updater is an exclusive updater, this topic must be in the branch of the topic tree that the client is the active update source for and that the updater is associated with.

Value

The value to use to update the topic. This value is of the data type that matches the data type of the topic being updated.

Context

OPTIONAL: A context object can be passed in to the update method that provides application state information.

Callback

The server uses the callback to return the result of the update. If the update completes successfully, the Diffusion server calls the callback's `onComplete` method. Otherwise, the Diffusion server calls the callback's `onError` method.

Using an updater to publish content to topics

Required permissions: `update_topic`

A client uses an updater to publish content to topics. Updaters can be used for exclusive or non-exclusive updating, depending on how the updater is acquired. When the client uses an updater method to publish content, it passes in the following parameters:

Topic path

The path to the topic to be updated.

If the updater is an exclusive updater, this topic must be in the branch of the topic tree that the client is the active update source for and that the updater is associated with.

Content

The information about the update can be provided as either a simple `Content` object or as a more complex `Update` object.

The content that is to be published to the topic. The client must use the appropriate content type when formatting the content. If the content uses the wrong content type for the topic, it can cause an error.

Update

The information about the update can be provided as either a simple `Content` object or as a more complex `Update` object.

An update that contains the content that is to be published to the topic and other information about the update, such as its type.

Use the `Builder` methods provided in the Unified API to build your `Update` objects.

Context

OPTIONAL: A context object can be passed in to the update method that provides application state information.

Callback

The server uses the callback to return the result of the update. If the update completes successfully, the Diffusion server calls the callback's `onComplete` method. Otherwise, the Diffusion server calls the callback's `onError` method.

Related reference

[Failover of active update sources](#) on page 110

You can use failover of active update sources to ensure that when a server that is the active update source for a section of the topic tree becomes unavailable, an update source on another server is assigned to be the active update source for that section of the topic tree. Failover of active update sources is enabled for any sections of the topic tree that have topic replication enabled.

Example: Make exclusive updates to a topic

The following examples use the Unified API to register as the update source of a topic and to update that topic with content. A client that updates a topic using this method locks the topic and prevents other clients from updating the topic.

JavaScript

```
diffusion.connect({
  host    : 'diffusion.example.com',
  port    : 443,
  secure  : true,
  principal : 'control',
  credentials : 'password'
}).then(function(session) {
  // A session may establish an exclusive update source. Once
  // active, only this session may update topics at or
  // under the registration branch.

  session.topics.registerUpdateSource('exclusive/topic', {
    onRegister : function(topic, deregister) {
      // The handler provides a deregistration function to
      // remove this registration and allow other sessions to
      // update topics under the registered path.
    },
    onActive : function(topic, updater) {
      // Once active, a handler may use the provided updater to
      // update any topics at or under the registered path
      updater.update('exclusive/topic/bar',
        123).then(function() {
          // The update was successful.
        }, function(err) {
          // There was an error updating the topic
        });
    },
    onStandBy : function(topic) {
      // If there is another update source registered for the
      // same topic path, any subsequent registrations will
      // be put into a standby state. The registration is still
      // held by the server, and the 'onActive' function
      // will be called if the pre-existing registration is
      // closed at a later point in time
    },
    onClose : function(topic, err) {
      // The 'onClose' function will be called once the
      // registration is closed, either by the session being closed
      // or the 'deregister' function being called.
    }
  });
});
```

Apple

```
@import Diffusion;

@interface TopicUpdateSourceExample (PTDiffusionTopicUpdateSource)
<PTDiffusionTopicUpdateSource>
@end

@implementation TopicUpdateSourceExample {
    PTDiffusionSession* _session;
}

-(void)startWithURL:(NSURL*)url {
    PTDiffusionCredentials *const credentials =
        [[PTDiffusionCredentials alloc]
         initWithPassword:@"password"];

    PTDiffusionSessionConfiguration *const sessionConfiguration =
        [[PTDiffusionSessionConfiguration alloc]
         initWithPrincipal:@"control"
         credentials:credentials];

    NSLog(@"Connecting...");

    [PTDiffusionSession openWithURL:url
                        configuration:sessionConfiguration
                        completionHandler:^(PTDiffusionSession *session,
                                           NSError *error)
     {
        if (!session) {
            NSLog(@"Failed to open session: %@", error);
            return;
        }

        // At this point we now have a connected session.
        NSLog(@"Connected.");

        // Set ivar to maintain a strong reference to the session.
        _session = session;

        // Add topic.
        [self addTopicForSession:session];
    }];
}

static NSString *const _TopicPath = @"Example/Exclusively Updating";

-(void)addTopicForSession:(PTDiffusionSession *const)session {
    // Add a single value topic without an initial value.
    [session.topicControl addWithTopicPath:_TopicPath
                             type:PTDiffusionTopicType_SingleValue
                             value:nil
                             completionHandler:^(NSError * _Nullable
                                                  error)
        {
            if (error) {
                NSLog(@"Failed to add topic. Error: %@", error);
            } else {
                NSLog(@"Topic created.");
            }

            // Register as an exclusive update source.
        }];
}
```

```

        [self registerAsUpdateSourceForSession:session];
    }
}];
}

-(void)registerAsUpdateSourceForSession:(PTDiffusionSession
*const)session {
    [session.topicUpdateControl registerUpdateSource:self
                                forTopicPath:_TopicPath

completionHandler:^(PTDiffusionTopicTreeRegistration *const
registration, NSError *const error)
    {
        if (registration) {
            NSLog(@"Registered as an update source.");
        } else {
            NSLog(@"Failed to register as an update source. Error:
%@", error);
        }
    }];
}

-(void)updateTopicWithUpdater:(PTDiffusionTopicUpdater *const)updater
                             value:(const NSUInteger)value {
    // Prepare data to update topic with.
    NSString *const string =
        [NSString stringWithFormat:@"Update #%lu", (unsigned
long)value];
    NSData *const data = [string
dataUsingEncoding:NSUTF8StringEncoding];
    PTDiffusionContent *const content =
        [[PTDiffusionContent alloc] initWithData:data];

    // Update the topic.
    [updater updateWithTopicPath:_TopicPath
                        value:content
                        completionHandler:^(NSError *const error)
    {
        if (error) {
            NSLog(@"Failed to update topic. Error: %@", error);
        } else {
            NSLog(@"Topic updated to \"%@\\"", string);

            // Update topic after a short wait.
            dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)
(1.0 * NSEC_PER_SEC)),
                dispatch_get_main_queue(), ^
            {
                [self updateTopicWithUpdater:updater value:value +
1];
            });
        }
    }];
}

@end

@implementation TopicUpdateSourceExample
(PTDiffusionTopicUpdateSource)

-(void)diffusionTopicTreeRegistration:
(PTDiffusionTopicTreeRegistration *const)registration

```

```

        isActiveWithUpdater:(PTDiffusionTopicUpdater
*const)updater {
    NSLog(@"Registration is active.");

    // Start updating.
    [self updateTopicWithUpdater:updater value:1];
}

@end

```

Java and Android

```

import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;

import com.pushtechology.diffusion.client.Diffusion;
import
    com.pushtechology.diffusion.client.callbacks.TopicTreeHandler;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicControl;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicControl.AddCal
import
    com.pushtechology.diffusion.client.features.control.topics.TopicUpdateControl;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicUpdateControl.
import
    com.pushtechology.diffusion.client.features.control.topics.TopicUpdateControl.
import
    com.pushtechology.diffusion.client.features.control.topics.TopicUpdateControl.
import com.pushtechology.diffusion.client.session.Session;
import
    com.pushtechology.diffusion.client.topics.details.SingleValueTopicDetails;
import
    com.pushtechology.diffusion.client.topics.details.TopicDetails;

/**
 * An example of using a control client as an event feed to a topic.
 * <P>
 * This uses the 'TopicControl' feature to create a topic and the
 * 'TopicUpdateControl' feature to send updates to it.
 * <P>
 * To send updates to a topic, the client session requires the
 * 'update_topic'
 * permission for that branch of the topic tree.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientAsUpdateSource {

    private static final String TOPIC_NAME = "Feeder";

    private final Session session;
    private final TopicControl topicControl;
    private final TopicUpdateControl updateControl;
    private final UpdateCallback updateCallback;

    /**
     * Constructor.
     *

```

```

    * @param callback for updates
    */
    public ControlClientAsUpdateSource(UpdateCallback callback) {

        updateCallback = callback;

        session =

Diffusion.sessions().principal("control").password("password")
        .open("ws://diffusion.example.com:80");
        topicControl = session.feature(TopicControl.class);
        updateControl = session.feature(TopicUpdateControl.class);
    }

    /**
     * Start the feed.
     *
     * @param provider the provider of prices
     * @param scheduler a scheduler service to schedule a periodic
     feeder task
     */
    public void start(
        final PriceProvider provider,
        final ScheduledExecutorService scheduler) {

        // Set up topic details
        final SingleValueTopicDetails.Builder builder =
            topicControl.newDetailsBuilder(
                SingleValueTopicDetails.Builder.class);

        final TopicDetails details =

builder.metadata(Diffusion.metadata().decimal("Price")).build();

        // Declare a custom update source implementation. When the
        source is set
        // as active start a periodic task to poll the provider every
        second and
        // update the topic. When the source is closed, stop the
        scheduled task.
        final UpdateSource source = new UpdateSource.Default() {
            private ScheduledFuture<?> theFeeder;

            @Override
            public void onActive(String topicPath, Updater updater) {
                theFeeder =
                    scheduler.scheduleAtFixedRate(
                        new FeederTask(provider, updater),
                        1, 1, TimeUnit.SECONDS);
            }

            @Override
            public void onClose(String topicPath) {
                if (theFeeder != null) {
                    theFeeder.cancel(true);
                }
            }
        };

        // Create the topic. When the callback indicates that the
        topic has been
        // created then register the topic source for the topic and
        request

```

```

        // that it is removed when the session closes.
        topicControl.addTopic(
            TOPIC_NAME,
            details,
            new AddCallback.Default() {
                @Override
                public void onTopicAdded(String topic) {
                    topicControl.removeTopicsWithSession(
                        topic,
                        new TopicTreeHandler.Default());
                    updateControl.registerUpdateSource(topic,
source);
                }
            });
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }

    /**
     * Periodic task to poll from provider and send update to server.
     */
    private final class FeederTask implements Runnable {

        private final PriceProvider priceProvider;
        private final Updater priceUpdater;

        private FeederTask(PriceProvider provider, Updater updater) {
            priceProvider = provider;
            priceUpdater = updater;
        }

        @Override
        public void run() {
            priceUpdater.update(
                TOPIC_NAME,

Diffusion.content().newContent(priceProvider.getPrice()),
                updateCallback);
        }
    }

    /**
     * Interface of a price provider that can periodically be polled
for a
     * price.
     */
    public interface PriceProvider {
        /**
         * Get the current price.
         *
         * @return current price as a decimal string
         */
        String getPrice();
    }
}

```

.NET

```
using System.Threading;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Topics;

namespace Examples {
    /// <summary>
    /// An example of using a control client as an event feed to a
    topic.
    ///
    /// This uses the <see cref="ITopicControl"/> feature to create a
    topic and the <see cref="ITopicUpdateControl"/>
    /// feature to send updates to it.
    ///
    /// To send updates to a topic, the client session requires the
    <see cref="TopicPermission.UPDATE_TOPIC"/>
    /// permission for that branch of the topic tree.
    /// </summary>
    public class ControlClientAsUpdateSource {
        private const string TopicName = "Feeder";
        private readonly ISession session;
        private readonly ITopicControl topicControl;
        private readonly ITopicUpdateControl updateControl;
        private readonly ITopicUpdaterUpdateCallback updateCallback;

        /// <summary>
        /// Constructor.
        /// </summary>
        /// <param name="callback">The callback for updates.</param>
        public
        ControlClientAsUpdateSource( ITopicUpdaterUpdateCallback callback )
        {
            updateCallback = callback;

            session =
            Diffusion.Sessions.Principal( "control" ).Password( "password" )
                .Open( "ws://diffusion.example.com;80" );

            topicControl = session.GetTopicControlFeature();
            updateControl = session.GetTopicUpdateControlFeature();
        }

        public void Start( IPriceProvider provider ) {
            // Set up topic details
            var builder =
            topicControl.CreateDetailsBuilder<ISingleValueTopicDetailsBuilder>();
            var details =
            builder.Metadata( Diffusion.Metadata.Decimal( "Price" ) ).Build();

            // Declare a custom update source implementation. When
            the source is set as active, start a periodic task
            // to poll the provider every second and update the
            topic. When the source is closed, stop the scheduled
            // task.
            var source = new UpdateSource( provider,
            updateCallback );

            // Create the topic. When the callback indicates that the
            topic has been created, register the topic
            // source for the topic.
        }
    }
}
```

```

        topicControl.AddTopicFromValue( TopicName, details, new
AddCallback( updateControl, source ) );
    }

    public void Close() {
        // Remove our topic and close the session when done.
        topicControl.RemoveTopics( ">" + TopicName, new
RemoveCallback( session ) );
    }

    private class RemoveCallback :
TopicControlRemoveCallbackDefault {
        private readonly ISession theSession;

        public RemoveCallback( ISession session ) {
            theSession = session;
        }

        /// <summary>
        /// Notification that a call context was closed
prematurely, typically due to a timeout or the session being
        /// closed. No further calls will be made for the
context.
        /// </summary>
        public override void OnDiscard() {
            theSession.Close();
        }

        /// <summary>
        /// Topic(s) have been removed.
        /// </summary>
        public override void OnTopicsRemoved() {
            theSession.Close();
        }
    }

    private class AddCallback : TopicControlAddCallbackDefault {
        private readonly ITopicUpdateControl updateControl;
        private readonly UpdateSource updateSource;

        public AddCallback( ITopicUpdateControl updater,
UpdateSource source ) {
            updateControl = updater;
            updateSource = source;
        }

        /// <summary>
        /// Topic has been added.
        /// </summary>
        /// <param name="topicPath">The full path of the topic
that was added.</param>
        public override void OnTopicAdded( string topicPath ) {
            updateControl.RegisterUpdateSource( topicPath,
updateSource );
        }
    }

    private class UpdateSource : TopicUpdateSourceDefault {
        private readonly IPriceProvider thePriceProvider;
        private readonly ITopicUpdaterUpdateCallback
theUpdateCallback;
        private readonly CancellationTokenSource
cancellationToken = new CancellationTokenSource();
    }

```

```

        public UpdateSource( IPriceProvider provider,
ITopicUpdaterUpdateCallback callback ) {
            thePriceProvider = provider;
            theUpdateCallback = callback;
        }

        /// <summary>
        /// State notification that this source is now active for
the specified topic path, and is therefore in a
        /// valid state to send updates on topics at or below the
registered topic path.
        /// </summary>
        /// <param name="topicPath">The registration path.</
param>
        /// <param name="updater">An updater that may be used to
update topics at or below the registered path.</param>
        public override void OnActive( string topicPath,
ITopicUpdater updater ) {
            PeriodicTaskFactory.Start( () => {
                updater.Update(
                    TopicName,
Diffusion.Content.NewContent( thePriceProvider.Price ),
theUpdateCallback );
            }, 1000, cancellationToken: cancellationToken.Token );
        }

        /// <summary>
        /// Called if the handler is closed. The handler will be
closed if the
        /// session is closed after the handler has been
registered, or if the
        /// handler is unregistered using <see
 cref="IRegistration.Close">close</see>.
        ///
        /// No further calls will be made for the handler.
        /// </summary>
        /// <param name="topicPath">the branch of the topic tree
for which the handler was registered</param>
        public override void OnClose( string topicPath ) {
            cancellationToken.Cancel();
        }
    }

    public interface IPriceProvider {
        /// <summary>
        /// Get the current price as a decimal string.
        /// </summary>
        string Price {
            get;
        }
    }
}

```

C

```

/**
 * Copyright © 2014, 2016 Push Technology Ltd.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.

```

```

* You may obtain a copy of the License at
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing,
software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
* See the License for the specific language governing permissions
and
* limitations under the License.
*
* This example is written in C99. Please use an appropriate C99
capable compiler
*
* @author Push Technology Limited
* @since 5.0
*/

/*
* This example creates a simple single-value topic and periodically
updates
* the data it contains.
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

#include <apr.h>
#include <apr_thread_mutex.h>
#include <apr_thread_cond.h>

#include "diffusion.h"
#include "args.h"
#include "conversation.h"
#include "service/svc-update.h"

int active = 0;

apr_pool_t *pool = NULL;
apr_thread_mutex_t *mutex = NULL;
apr_thread_cond_t *cond = NULL;

ARG_OPTS_T arg_opts[] = {
    ARG_OPTS_HELP,
    {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
    ARG_HAS_VALUE, "ws://localhost:8080"},
    {'p', "principal", "Principal (username) for the connection",
    ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    {'c', "credentials", "Credentials (password) for the
connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    {'t', "topic", "Topic name to create and update",
    ARG_OPTIONAL, ARG_HAS_VALUE, "time"},
    {'s', "seconds", "Number of seconds to run for before
exiting", ARG_OPTIONAL, ARG_HAS_VALUE, "30"},
    END_OF_ARG_OPTS
};

/*
* Handlers for add topic feature.
*/
static int

```

```

on_topic_added(SESSION_T *session, const SVC_ADD_TOPIC_RESPONSE_T
 *response, void *context)
{
    printf("Added topic\n");
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}

static int
on_topic_add_failed(SESSION_T *session, const
 SVC_ADD_TOPIC_RESPONSE_T *response, void *context)
{
    printf("Failed to add topic (%d)\n", response-
>response_code);
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}

static int
on_topic_add_discard(SESSION_T *session, void *context)
{
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}

/*
 * Handlers for registration of update source feature
 */
static int
on_update_source_init(SESSION_T *session,
                     const CONVERSATION_ID_T *updater_id,
                     const SVC_UPDATE_REGISTRATION_RESPONSE_T
 *response,
                     void *context)
{
    char *id_str = conversation_id_to_string(*updater_id);
    printf("Topic source \"%s\" in init state\n", id_str);
    free(id_str);
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
    return HANDLER_SUCCESS;
}

static int
on_update_source_registered(SESSION_T *session,
                           const CONVERSATION_ID_T *updater_id,
                           const SVC_UPDATE_REGISTRATION_RESPONSE_T
 *response,
                           void *context)
{
    char *id_str = conversation_id_to_string(*updater_id);
    printf("Registered update source \"%s\"\n", id_str);
    free(id_str);
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);
}

```

```

        return HANDLER_SUCCESS;
    }

    static int
    on_update_source_deregistered(SESSION_T *session,
                                  const CONVERSATION_ID_T *updater_id,
                                  void *context)
    {
        char *id_str = conversation_id_to_string(*updater_id);
        printf("Deregistered update source \"%s\"\n", id_str);
        free(id_str);
        apr_thread_mutex_lock(mutex);
        apr_thread_cond_broadcast(cond);
        apr_thread_mutex_unlock(mutex);
        return HANDLER_SUCCESS;
    }

    static int
    on_update_source_active(SESSION_T *session,
                            const CONVERSATION_ID_T *updater_id,
                            const SVC_UPDATE_REGISTRATION_RESPONSE_T
                            *response,
                            void *context)
    {
        char *id_str = conversation_id_to_string(*updater_id);
        printf("Topic source \"%s\" active\n", id_str);
        free(id_str);
        active = 1;
        apr_thread_mutex_lock(mutex);
        apr_thread_cond_broadcast(cond);
        apr_thread_mutex_unlock(mutex);
        return HANDLER_SUCCESS;
    }

    static int
    on_update_source_standby(SESSION_T *session,
                             const CONVERSATION_ID_T *updater_id,
                             const SVC_UPDATE_REGISTRATION_RESPONSE_T
                             *response,
                             void *context)
    {
        char *id_str = conversation_id_to_string(*updater_id);
        printf("Topic source \"%s\" on standby\n", id_str);
        free(id_str);
        apr_thread_mutex_lock(mutex);
        apr_thread_cond_broadcast(cond);
        apr_thread_mutex_unlock(mutex);
        return HANDLER_SUCCESS;
    }

    static int
    on_update_source_closed(SESSION_T *session,
                            const CONVERSATION_ID_T *updater_id,
                            const SVC_UPDATE_REGISTRATION_RESPONSE_T
                            *response,
                            void *context)
    {
        char *id_str = conversation_id_to_string(*updater_id);
        printf("Topic source \"%s\" closed\n", id_str);
        free(id_str);
        apr_thread_mutex_lock(mutex);
        apr_thread_cond_broadcast(cond);
        apr_thread_mutex_unlock(mutex);
    }

```

```

        return HANDLER_SUCCESS;
    }

    /*
     * Handlers for update of data.
     */
    static int
    on_update_success(SESSION_T *session,
                     const CONVERSATION_ID_T *updater_id,
                     const SVC_UPDATE_RESPONSE_T *response,
                     void *context)
    {
        char *id_str = conversation_id_to_string(*updater_id);
        printf("on_update_success for updater \"%s\"\n", id_str);
        free(id_str);
        return HANDLER_SUCCESS;
    }

    static int
    on_update_failure(SESSION_T *session,
                     const CONVERSATION_ID_T *updater_id,
                     const SVC_UPDATE_RESPONSE_T *response,
                     void *context)
    {
        char *id_str = conversation_id_to_string(*updater_id);
        printf("on_update_failure for updater \"%s\"\n", id_str);
        free(id_str);
        return HANDLER_SUCCESS;
    }

    /*
     * Program entry point.
     */
    int
    main(int argc, char** argv)
    {
        /*
         * Standard command-line parsing.
         */
        const HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if(options == NULL || hash_get(options, "help") != NULL) {
            show_usage(argc, argv, arg_opts);
            return EXIT_FAILURE;
        }

        const char *url = hash_get(options, "url");
        const char *principal = hash_get(options, "principal");
        CREDENTIALS_T *credentials = NULL;
        const char *password = hash_get(options, "credentials");
        if(password != NULL) {
            credentials = credentials_create_password(password);
        }
        const char *topic_name = hash_get(options, "topic");
        const long seconds = atol(hash_get(options, "seconds"));

        /*
         * Setup for condition variable.
         */
        apr_initialize();
        apr_pool_create(&pool, NULL);
        apr_thread_mutex_create(&mutex, APR_THREAD_MUTEX_UNNESTED,
pool);
        apr_thread_cond_create(&cond, pool);
    }

```

```

/*
 * Create a session with the Diffusion server.
 */
SESSION_T *session;
DIFFUSION_ERROR_T error = { 0 };
session = session_create(url, principal, credentials, NULL,
NULL, &error);
if(session == NULL) {
    fprintf(stderr, "TEST: Failed to create session\n");
    fprintf(stderr, "ERR : %s\n", error.message);
    return EXIT_FAILURE;
}

/*
 * Create a topic holding simple string content.
 */
TOPIC_DETAILS_T *string_topic_details =
create_topic_details_single_value(M_DATA_TYPE_STRING);
const ADD_TOPIC_PARAMS_T add_topic_params = {
    .topic_path = topic_name,
    .details = string_topic_details,
    .on_topic_added = on_topic_added,
    .on_topic_add_failed = on_topic_add_failed,
    .on_discard = on_topic_add_discard,
};

apr_thread_mutex_lock(mutex);
add_topic(session, add_topic_params);
apr_thread_cond_wait(cond, mutex);
apr_thread_mutex_unlock(mutex);

topic_details_free(string_topic_details);

/*
 * Define the handlers for add_update_source()
 */
const UPDATE_SOURCE_REGISTRATION_PARAMS_T update_reg_params =
{
    .topic_path = topic_name,
    .on_init = on_update_source_init,
    .on_registered = on_update_source_registered,
    .on_active = on_update_source_active,
    .on_standby = on_update_source_standby,
    .on_close = on_update_source_closed
};

/*
 * Register an updater.
 */
apr_thread_mutex_lock(mutex);
CONVERSATION_ID_T *updater_id =
register_update_source(session, update_reg_params);
apr_thread_cond_wait(cond, mutex);
apr_thread_mutex_unlock(mutex);

/*
 * Define default parameters for an update source.
 */
UPDATE_SOURCE_PARAMS_T update_source_params_base = {
    .updater_id = updater_id,
    .topic_path = topic_name,
    .on_success = on_update_success,

```

```

        .on_failure = on_update_failure
    };

    time_t end_time = time(NULL) + seconds;

    while(time(NULL) < end_time) {

        if(active) {
            /*
             * Create an update structure containing the
current time.
             */
            BUF_T *buf = buf_create();
            const time_t time_now = time(NULL);
            buf_write_string(buf, ctime(&time_now));

            CONTENT_T *content =
content_create(CONTENT_ENCODING_NONE, buf);

            UPDATE_T *upd =
update_create(UPDATE_ACTION_REFRESH,

UPDATE_TYPE_CONTENT,

                                content);

            UPDATE_SOURCE_PARAMS_T update_source_params =
update_source_params_base;
            update_source_params.update = upd;

            /*
             * Update the topic.
             */
            update(session, update_source_params);

            content_free(content);
            update_free(upd);
            buf_free(buf);
        }

        sleep(1);
    }

    if(active) {
        UPDATE_SOURCE_DEREGISTRATION_PARAMS_T
update_dereg_params = {
            .updater_id = updater_id,
            .on_deregistered =
on_update_source_deregistered
        };

        apr_thread_mutex_lock(mutex);
        deregister_update_source(session,
update_dereg_params);
        apr_thread_cond_wait(cond, mutex);
        apr_thread_mutex_unlock(mutex);
    }

    /*
     * Close session and free resources.
     */
    session_close(session, NULL);
    session_free(session);

```

```

    conversation_id_free(updater_id);
    credentials_free(credentials);

    apr_thread_mutex_destroy(mutex);
    apr_thread_cond_destroy(cond);
    apr_pool_destroy(pool);
    apr_terminate();

    return EXIT_SUCCESS;
}

```

Change the URL from that provided in the example to the URL of the Diffusion server.

Example: Make non-exclusive updates to a topic

The following examples use the Unified API to update a topic with content. Updating a topic this way does not prevent other clients from updating the topic.

JavaScript

```

// 1. A session may update any existing topic. Update values must be
// of the same type as the topic being updated.

// Add a topic first with a string type
session.topics.add('foo', '').then(function() {
    // Update the topic
    return session.topics.update('foo', 'hello');
}).then(function() {
    // Update the topic again
    return session.topics.update('foo', 'world');
});

// 2. If using RecordContent metadata, update values are
// constructed from the metadata

// Create a new metadata instance
var meta = new diffusion.metadata.RecordContent();

meta.addRecord('record', 1, {
    'field' : meta.integer()
});

// Create a builder to set values
var builder = meta.builder();

builder.add('record', {
    field : 123
});

// Update the topic with the new value
session.topics.add('topic', '').then(function() {
    session.topics.update('topic', builder.build());
});

```

Apple

```

@import Diffusion;

@implementation TopicUpdateExample {

```

```

    PTDiffusionSession* _session;
}

-(void)startWithURL:(NSURL*)url {

    PTDiffusionCredentials *const credentials =
        [[PTDiffusionCredentials alloc]
         initWithPassword:@"password"];

    PTDiffusionSessionConfiguration *const sessionConfiguration =
        [[PTDiffusionSessionConfiguration alloc]
         initWithPrincipal:@"control"
         credentials:credentials];

    NSLog(@"Connecting...");

    [PTDiffusionSession openWithURL:url
                        configuration:sessionConfiguration
                        completionHandler:^(PTDiffusionSession *session,
                                           NSError *error)
     {
        if (!session) {
            NSLog(@"Failed to open session: %@", error);
            return;
        }

        // At this point we now have a connected session.
        NSLog(@"Connected.");

        // Set ivar to maintain a strong reference to the session.
        _session = session;

        // Add topic.
        [self addTopicForSession:session];
    }];

static NSString *const _TopicPath = @"Example/Updating";

-(void)addTopicForSession:(PTDiffusionSession *const)session {
    // Add a single value topic without an initial value.
    [session.topicControl addWithTopicPath:_TopicPath

     type:PTDiffusionTopicType_SingleValue
     value:nil
     completionHandler:^(NSError * _Nullable
                          error)
    {
        if (error) {
            NSLog(@"Failed to add topic. Error: %@", error);
        } else {
            NSLog(@"Topic created.");

            // Update topic after a short wait.
            [self updateTopicForSession:session withValue:1];
        }
    }];
}

-(void)updateTopicForSession:(PTDiffusionSession *const)session
withValue:(const NSUInteger)value {

```

```

    dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(1.0 *
NSEC_PER_SEC)),
        dispatch_get_main_queue(), ^
    {
        // Get the non-exclusive updater.
        PTDiffusionTopicUpdater *const updater =
session.topicUpdateControl.updater;

        // Prepare data to update topic with.
        NSString *const string =
        [NSString stringWithFormat:@"Update #%lu", (unsigned
long)value];
        NSData *const data = [string
dataUsingEncoding:NSUTF8StringEncoding];
        PTDiffusionContent *const content =
        [[PTDiffusionContent alloc] initWithData:data];

        // Update the topic.
        [updater updateWithTopicPath:_TopicPath
            value:content
            completionHandler:^(NSError *const error)
        {
            if (error) {
                NSLog(@"Failed to update topic. Error: %@", error);
            } else {
                NSLog(@"Topic updated to \"%@\\"", string);

                // Update topic after a short wait.
                [self updateTopicForSession:session withValue:value +
1];
            }
        }
    ]];
});
}

@end

```

Java and Android

```

import com.pushtechology.diffusion.client.Diffusion;
import
    com.pushtechology.diffusion.client.callbacks.TopicTreeHandler;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicControl;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicControl.AddCal
import
    com.pushtechology.diffusion.client.features.control.topics.TopicUpdateControl;
import
    com.pushtechology.diffusion.client.features.control.topics.TopicUpdateControl.
import com.pushtechology.diffusion.client.session.Session;
import com.pushtechology.diffusion.client.topics.details.TopicType;

/**
 * An example of using a control client to create and update a topic
 * in non
 * exclusive mode (as opposed to acting as an exclusive update
 * source). In this
 * mode other clients could update the same topic (on a last update
 * wins basis).
 * <P>
 * This uses the 'TopicControl' feature to create a topic and the

```

```

* 'TopicUpdateControl' feature to send updates to it.
* <P>
* To send updates to a topic, the client session requires the
'update_topic'
* permission for that branch of the topic tree.
*
* @author Push Technology Limited
* @since 5.3
*/
public final class ControlClientUpdatingSingleValueTopic {

    private static final String TOPIC = "MyTopic";

    private final Session session;
    private final TopicControl topicControl;
    private final TopicUpdateControl updateControl;

    /**
     * Constructor.
     */
    public ControlClientUpdatingSingleValueTopic() {

        session =
Diffusion.sessions().principal("control").password("password")
        .open("ws://diffusion.example.com:80");

        topicControl = session.feature(TopicControl.class);
        updateControl = session.feature(TopicUpdateControl.class);

        // Create the topic and request that it is removed when the
session
        // closes
        topicControl.addTopic(
            TOPIC,
            TopicType.SINGLE_VALUE,
            new AddCallback.Default() {
                @Override
                public void onTopicAdded(String topicPath) {
                    topicControl.removeTopicsWithSession(
                        TOPIC,
                        new TopicTreeHandler.Default());
                }
            }
        );
    }

    /**
     * Update the topic with a string value.
     *
     * @param value the update value
     * @param callback the update callback
     */
    public void update(String value, UpdateCallback callback) {
        updateControl.updater().update(TOPIC, value, callback);
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }
}

```

```
}
```

.NET

```
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Topics;

namespace Examples {
    /// <summary>
    /// An example of using a control client to create and update a
    /// topic in non-exclusive mode (as opposed to acting
    /// as an exclusive update source). In this mode other clients
    /// could update the same topic (on a 'last update wins'
    /// basis).
    ///
    /// This uses the <see cref="ITopicControl"/> feature to create a
    /// topic and the <see cref="ITopicUpdateControl"/>
    /// feature to send updates to it.
    ///
    /// To send updates to a topic, the client session requires the
    /// 'update_topic' permission for that branch of the
    /// topic tree.
    /// </summary>
    public class ControlClientUpdatingTopic {
        private const string Topic = "MyTopic";
        private readonly ISession session;
        private readonly ITopicControl topicControl;
        private readonly ITopicUpdateControl updateControl;

        /// <summary>
        /// Constructor.
        /// </summary>
        public ControlClientUpdatingTopic() {
            session =
Diffusion.Sessions.Principal( "control" ).Password( "password" )
                .Open( "ws://diffusion.example.com:80" );

            topicControl = session.GetTopicControlFeature();
            updateControl = session.GetTopicUpdateControlFeature();

            // Create a single-value topic.
            topicControl.AddTopicFromValue( Topic,
TopicType.SINGLE_VALUE, new TopicControlAddCallbackDefault() );
        }

        /// <summary>
        /// Update the topic with a string value.
        /// </summary>
        /// <param name="value">The update value.</param>
        /// <param name="callback">The update callback.</param>
        public void Update( string value, ITopicUpdaterUpdateCallback
callback ) {
            updateControl.Updater.Update( Topic, value, callback );
        }

        /// <summary>
        /// Close the session.
        /// </summary>
        public void Close() {
            // Remove our topic and close session when done.

```

```

        topicControl.RemoveTopics( ">" + Topic, new
RemoveCallback( session ) );
    }

    private class RemoveCallback :
TopicControlRemoveCallbackDefault {
        private readonly ISession theSession;

        public RemoveCallback( ISession session ) {
            theSession = session;
        }

        /// <summary>
        /// Notification that a call context was closed
prematurely, typically due to a timeout or the session being
        /// closed. No further calls will be made for the
context.
        /// </summary>
        public override void OnDiscard() {
            theSession.Close();
        }

        /// <summary>
        /// Topic(s) have been removed.
        /// </summary>
        public override void OnTopicsRemoved() {
            theSession.Close();
        }
    }
}
}
}

```

Change the URL from that provided in the example to the URL of the Diffusion server.

Managing subscriptions

A client can use the SubscriptionControl feature to subscribe other client sessions to topics that they have not requested subscription to themselves and also to unsubscribe clients from topics. It also enables the client to register as the handler for routing topic subscriptions.

Subscribing and unsubscribing clients

Required permissions: modify_session, select_topic permission for the topics being subscribed to

A client can subscribe client sessions that it knows about to topics that those clients have not explicitly requested. It can also unsubscribe clients from topics.

A session identifier is required to specify the client session that is to be subscribed or unsubscribed. Use the ClientControl feature to get the identifiers for connected client sessions.

The SubscriptionControl feature uses topic selectors to specify topics for subscription and unsubscription. Many topics can be specified in a single operation.

The client being subscribed to topics must have read_topic permission for the topics it is being subscribed to.

Using session properties to select clients to subscribe and unsubscribe

Required permissions: view_session, modify_session, select_topic permission for the topics being subscribed to

When managing client subscriptions, a client can specify a filter for which client sessions it subscribes to topics or unsubscribes from topics. The filter is a query expression on the values of session properties.

The managing client defines a filter and sends a subscription request with this filter to the Diffusion server. The Diffusion server evaluates the query and subscribes those currently connected client sessions whose session properties match the filter to the topic or topics.

The filter is evaluated only once. Clients that subsequently connect or clients whose properties change are do not cause the subscription request to be reevaluated. Even if these clients match the filter, they are not subscribed.

Managing all subscriptions from a separate control session

You can prevent client sessions from subscribing themselves to topics and control all subscriptions from a separate control client session that uses SubscriptionControl feature to subscribe clients to topics.

To restrict subscription capability to control sessions, configure the following permissions:

Control session:

- Grant the `modify_session` permission
- Grant the `select_topic` permission

This can either be granted for the default topic scope or more selectively to restrict the topic selectors the control session can use.

Other sessions:

- Grant `read_topic` to the appropriate topics.
- Deny the `select_topic` permission by default.

Do not assign the session a role that has the `select_topic` permission for the default topic scope. This prevents the session from subscribing to all topics using a wildcard selector.

- Optionally, grant the `select_topic` permission to specific branches of the topic tree to which the session can subscribe freely.

Acting as a routing subscription handler

Required permissions: `view_session`, `modify_session`, `register_handler`

Routing topics can be created with a server-side handler that assigns clients to real topics. However, you can omit the server-side handler such that subscriptions to routing topics are directed at a client acting as a routing subscription handler.

A client can register a routing subscription handler for a branch of the topic tree. Any subscription requests to routing topics in that branch that do not have server-side handlers are passed to the client for action.

On receipt of a routing subscription request the client can respond with a route request that specifies the path of the actual topic that the routing topic maps to for the requesting client. This subscription fails if the target topic does not already exist or if the requesting client does not have `read_topic` permission for the routing topic or target topic.

The client can complete other actions before calling back to `route`. For example, it could use the `TopicControl` feature to create the topic that the client is to map to.

Alternatively, the client can defer the routing subscription request in which case the requesting client remains unsubscribed. This is similar to denying it from an authorization point of view.

The client must reply with a route or defer for all routing requests.

Related concepts

[Topic selectors in the Unified API](#) on page 61

A topic selector identifies one or more topics. You can create a topic selector object from a pattern expression.

[Topic selectors in the Classic API \(deprecated\)](#) on page 69

A topic selector is a string that can be used by the Classic API to select more than one topic by indicating that subordinate topics are to be included or by fuzzy matching on topic names or both.

[Session properties](#) on page 267

A client session has a number of properties associated with it. Properties are key-value pairs. Both the key and the value are case sensitive.

[Session filtering](#) on page 268

Session filters enable you to query the set of connected client sessions on the Diffusion server based on their session properties.

Example: Subscribe other clients to topics

The following examples use the SubscriptionControl feature in the Unified API to subscribe other client sessions to topics.

Java and Android

```
package com.pushtechology.diffusion.examples;

import com.pushtechology.diffusion.client.Diffusion;
import
    com.pushtechology.diffusion.client.features.control.topics.SubscriptionControl
import
    com.pushtechology.diffusion.client.features.control.topics.SubscriptionControl
import com.pushtechology.diffusion.client.session.Session;
import com.pushtechology.diffusion.client.session.SessionId;

/**
 * This demonstrates using a client to subscribe and unsubscribe
 * other clients
 * to topics.
 * <P>
 * This uses the 'SubscriptionControl' feature.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientSubscriptionControl {

    private final Session session;

    private final SubscriptionControl subscriptionControl;

    /**
     * Constructor.
     */
    public ControlClientSubscriptionControl() {

        session =

        Diffusion.sessions().principal("control").password("password")
            .open("ws://diffusion.example.com:80");
```

```

        subscriptionControl =
session.feature(SubscriptionControl.class);
    }

    /**
     * Subscribe a client to topics.
     *
     * @param sessionId client to subscribe
     * @param topicSelector topic selector expression
     * @param callback for subscription result
     */
    public void subscribe(
        SessionId sessionId,
        String topicSelector,
        SubscriptionCallback callback) {

        // To subscribe a client to a topic, this client session
        // must have the 'modify_session' permission.
        subscriptionControl.subscribe(
            sessionId,
            topicSelector,
            callback);
    }

    /**
     * Unsubscribe a client from topics.
     *
     * @param sessionId client to unsubscribe
     * @param topicSelector topic selector expression
     * @param callback for unsubscription result
     */
    public void unsubscribe(
        SessionId sessionId,
        String topicSelector,
        SubscriptionCallback callback) {

        // To unsubscribe a client from a topic, this client session
        // must have the 'modify_session' permission.
        subscriptionControl.unsubscribe(
            sessionId,
            topicSelector,
            callback);
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }
}

```

.NET

```

using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;

namespace Examples {
    /// <summary>

```

```

    /// This demonstrates using a client to subscribe and unsubscribe
    other clients to topics.
    ///
    /// This uses the <see cref="ISubscriptionControl"/> feature.
    /// </summary>
    public class ControlClientSubscriptionControl {
        private readonly ISession session;
        private readonly ISubscriptionControl subscriptionControl;

        /// <summary>
        /// Constructor.
        /// </summary>
        public ControlClientSubscriptionControl() {
            session =
Diffusion.Sessions.Principal( "control" ).Password( "password" )
                .Open( "ws://diffusion.example.com:80" );

            subscriptionControl =
session.GetSubscriptionControlFeature();
        }

        /// <summary>
        /// Subscribe a client to topics.
        /// </summary>
        /// <param name="sessionId">The session id of the client to
subscribe.</param>
        /// <param name="topicSelector">The topic selector
expression.</param>
        /// <param name="callback">The callback for the subscription
result.</param>
        public void Subscribe( SessionId sessionId, string
topicSelector, ISubscriptionCallback callback ) {
            // To subscribe a client to a topic, this client session
must have the MODIFY_SESSION permission.
            subscriptionControl.Subscribe( sessionId, topicSelector,
callback );
        }

        /// <summary>
        /// Unsubscribe a client from topics.
        /// </summary>
        /// <param name="sessionId">The session id of the client to
unsubscribe.</param>
        /// <param name="topicSelector">The topic selector
expression.</param>
        /// <param name="callback">The callback for the
unsubscribe result.</param>
        public void Unsubscribe( SessionId sessionId, string
topicSelector, ISubscriptionCallback callback ) {
            subscriptionControl.Unsubscribe( sessionId,
topicSelector, callback );
        }

        /// <summary>
        /// Close the session.
        /// </summary>
        public void Close() {
            session.Close();
        }
    }
}

```

C

```

/*
 * This example waits to be notified of a client connection, and then
 * subscribes that client to a named topic.
 */

#include <stdio.h>
#include <unistd.h>

#include <apr.h>
#include <apr_thread_mutex.h>
#include <apr_thread_cond.h>

#include "diffusion.h"
#include "args.h"

ARG_OPTS_T arg_opts[] = {
    ARG_OPTS_HELP,
    {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
     ARG_HAS_VALUE, "ws://localhost:8080"},
    {'p', "principal", "Principal (username) for the connection",
     ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    {'c', "credentials", "Credentials (password) for the
     connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    {'t', "topic_selector", "Topic selector to subscribe/
     unsubscribe clients from", ARG_OPTIONAL, ARG_HAS_VALUE, ">foo"},
    END_OF_ARG_OPTS
};
HASH_T *options = NULL;

/*
 * Callback invoked when a client has been successfully subscribed to
 * a topic.
 */
static int
on_subscription_complete(SESSION_T *session, void *context)
{
    printf("Subscription complete\n");
    return HANDLER_SUCCESS;
}

/*
 * Callback invoked when a client session has been opened.
 */
static int
on_session_open(SESSION_T *session, const SESSION_PROPERTIES_EVENT_T
 *request, void *context)
{
    if(session_id_cmp(*session->id, request->session_id) == 0) {
        // It's our own session, ignore.
        return HANDLER_SUCCESS;
    }

    char *topic_selector = hash_get(options, "topic_selector");

    char *sid_str = session_id_to_string(&request->session_id);
    printf("Subscribing session %s to topic selector %s\n",
    sid_str, topic_selector);
    free(sid_str);

    /*
     * Subscribe the client session to the topic.

```

```

        */
SUBSCRIPTION_CONTROL_PARAMS_T subscribe_params = {
    .session_id = request->session_id,
    .topic_selector = topic_selector,
    .on_complete = on_subscription_complete
};
subscribe_client(session, subscribe_params);

return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
    /*
     * Standard command-line parsing.
     */
    options = parse_cmdline(argc, argv, arg_opts);
    if(options == NULL || hash_get(options, "help") != NULL) {
        show_usage(argc, argv, arg_opts);
        return EXIT_FAILURE;
    }

    const char *url = hash_get(options, "url");
    const char *principal = hash_get(options, "principal");
    CREDENTIALS_T *credentials = NULL;
    const char *password = hash_get(options, "credentials");
    if(password != NULL) {
        credentials = credentials_create_password(password);
    }

    /*
     * Create a session with Diffusion.
     */
    DIFFUSION_ERROR_T error = { 0 };
    SESSION_T *session = session_create(url, principal,
credentials, NULL, NULL, &error);
    if(session == NULL) {
        fprintf(stderr, "Failed to create session: %s\n",
error.message);
        return EXIT_FAILURE;
    }

    /*
     * Register a session properties listener, so we are notified
     * of new client connections.
     * In the callback, we will subscribe the client to topics
     * according to the topic_selector argument.
     */
    SET_T *required_properties = set_new_string(1);
    set_add(required_properties,
PROPERTIES_SELECTOR_ALL_FIXED_PROPERTIES);
    SESSION_PROPERTIES_REGISTRATION_PARAMS_T params = {
        .on_session_open = on_session_open,
        .required_properties = required_properties
    };
    session_properties_listener_register(session, params);
    set_free(required_properties);

    /*
     * Pretend to do some work.
     */
    sleep(10);

```

```

    /*
     * Close session and tidy up.
     */
    session_close(session, NULL);
    session_free(session);

    return EXIT_SUCCESS;
}

```

Change the URL from that provided in the example to the URL of the Diffusion server.

Example: Receive notifications when a client subscribes to a routing topic

The following examples use the SubscriptionControl feature in the Unified API to listen for notifications of when a client subscribes to a routing topic.

Java and Android

```

package com.pushtechology.diffusion.examples;

import com.pushtechology.diffusion.client.Diffusion;
import
    com.pushtechology.diffusion.client.features.control.topics.SubscriptionControl
import
    com.pushtechology.diffusion.client.features.control.topics.SubscriptionControl
import
    com.pushtechology.diffusion.client.features.control.topics.SubscriptionControl
import com.pushtechology.diffusion.client.session.Session;

/**
 * This demonstrates using a control client to be notified of
 * subscription
 * requests to routing topics.
 * <P>
 * This uses the 'SubscriptionControl' feature.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientSubscriptionControlRouting {

    private final Session session;

    /**
     * Constructor.
     *
     * @param routingCallback for routing subscription requests
     */
    public ControlClientSubscriptionControlRouting(
        final SubscriptionCallback routingCallback) {

        session =

Diffusion.sessions().principal("control").password("password")
        .open("ws://diffusion.example.com:80");

        final SubscriptionControl subscriptionControl =
            session.feature(SubscriptionControl.class);

```

```

        // Sets up a handler so that all subscriptions to topic a/b
are routed
        // to routing/target/topic
        // To do this, the client session requires the
'view_session',
        // 'modify_session', and 'register_handler' permissions.
        subscriptionControl.addRoutingSubscriptionHandler(
            "a/b",
            new
SubscriptionControl.RoutingSubscriptionRequest.Handler
            .Default() {
                @Override
                public void onSubscriptionRequest(
                    final RoutingSubscriptionRequest request) {

                    request.route(
                        "routing/target/topic",
                        routingCallback);
                }
            });
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }
}

```

.NET

```

using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;

namespace Examples {
    /// <summary>
    /// This demonstrates using a control client to be notified of
subscription requests to routing topics.
    ///
    /// This uses the <see cref="ISubscriptionControl"/> feature.
    /// </summary>
    public class ControlClientSubscriptionControlRouting {
        private readonly ISession session;

        /// <summary>
        /// Constructor.
        /// </summary>
        /// <param name="routingCallback">The callback for routing
subscription requests.</param>
        public
ControlClientSubscriptionControlRouting( ISubscriptionCallback
routingCallback ) {
            session =
Diffusion.Sessions.Principal( "control" ).Password( "password" )
                .Open( "ws://diffusion.example.com:80" );

            var subscriptionControl =
session.GetSubscriptionControlFeature();

```

```

        // Sets up a handler so that all subscriptions to topic
        'a/b' are routed to the routing/target topic.
        // To do this, the client session requires the
        VIEW_SESSION, MODIFY_SESSION and REGISTER_HANDLER
        // permissions.
        subscriptionControl.AddRoutingSubscriptionHandler( "a/b",
        new SubscriptionHandler( routingCallback ) );
    }

    /// <summary>
    /// Close the session.
    /// </summary>
    public void Close() {
        session.Close();
    }

    private class SubscriptionHandler :
    RoutingSubscriptionRequestHandlerDefault {
        private readonly ISubscriptionCallback
        theRoutingCallback;

        public SubscriptionHandler( ISubscriptionCallback
        callback ) {
            theRoutingCallback = callback;
        }

        /// <summary>
        /// A request to subscribe to a specific routing topic.
        /// </summary>
        /// <param name="request"></param>
        public override void
        OnSubscriptionRequest( IRoutingSubscriptionRequest request ) {
            request.Route( "routing/target/topic",
            theRoutingCallback );
        }
    }
}

```

Change the URL from that provided in the example to the URL of the Diffusion server.

Messaging to topic paths

A client can use the Messaging feature to send individual messages to a handler that has registered to receive messages on a topic path.

Sending messages to clients

Required permissions: send_to_message_handler

A client can send a message to a topic path, regardless of whether a topic is bound to that topic path. The messages are delivered to a handler that has registered to receive messages on that topic path.

The body of the message that is sent is represented as *bytes*. Any of the builder features can be used to build a message as *content*, JSON, or binary. With messaging you can also send an empty message.

When sending a message certain additional options can also be specified:

Headers

A set of string values that can be sent along with the content.

Priority

Use this to specify the priority used when queuing the message for the client at the Diffusion server.

If it is a publisher that has registered to receive a message on a topic, the message is mapped to a delta `TopicMessage`.

Listen for messages on a topic path

A client can specify a listener that receives messages sent to the client on a topic path.

Example: Send a message to a topic path

The following examples use the Unified API to send a message to a topic path. The message is received by a handler that has registered to receive messages on that topic path.

JavaScript

```
var diffusion = require('diffusion');

// Connect to the server. Change these options to suit your own
// environment.
// Node.js will not accept self-signed certificates by default. If
// you have
// one of these, set the environment variable
// NODE_TLS_REJECT_UNAUTHORIZED=0
// before running this example.
diffusion.connect({
  host    : 'diffusion.example.com',
  port    : 443,
  secure  : true,
  principal : 'control',
  credentials : 'password'
}).then(function(session) {
  // 1. Messages can be sent & received between sessions.

  // Create a stream of received messages for a specific path
  session.messages.listen('foo').on('message', function(msg) {
    console.log('Received message: ' + msg.content);
  });

  // Send a message to another session. It is the application's
  // responsibility to find the SessionID of the intended
  // recipient.
  session.messages.send('foo', 'Hello world', 'another-session-
  id');

  // 2. Messages can also be sent without a recipient, in which
  // case they will be dispatched to any Message Handlers
  // that have been registered for the same path. If multiple
  // handlers are registered to the same path, any given
  // message will only be dispatched to one handler.

  // Register the handler to receive messages at or below the given
  // path.
  session.messages.addHandler('foo', {
    onActive : function() {
      console.log('Handler registered');
    },
  },
```

```

    onClose : function() {
        console.log('Handler closed');
    },
    onMessage : function(msg) {
        console.log('Received message:' + msg.content + ' from
Session: ' + msg.session);
        if (msg.properties) {
            console.log('with properties:', msg.properties);
        }
    }
}).then(function() {
    console.log('Registered handler');
}, function(e) {
    console.log('Failed to register handler: ', e);
});

// Send a message at a lower path, without an explicit recipient
- this will be received by the Handler.
    session.messages.send('foo/bar', 'Another message');
});

```

Apple

```

#import Diffusion;

@implementation MessagingSendExample {
    PTDiffusionSession* _session;
    NSUInteger _nextValue;
}

-(void)startWithURL:(NSURL*)url {
    NSLog(@"Connecting...");

    [PTDiffusionSession openWithURL:url
        completionHandler:^(PTDiffusionSession *session,
NSError *error)
    {
        if (!session) {
            NSLog(@"Failed to open session: %@", error);
            return;
        }

        // At this point we now have a connected session.
        NSLog(@"Connected.");

        // Set ivar to maintain a strong reference to the session.
        _session = session;

        // Create a timer to send a message once a second.
        NSTimer *const timer = [NSTimer timerWithTimeInterval:1.0
            target:self

selector:@selector(sendMessage:)

userInfo:session
repeats:YES];

        [[NSRunLoop currentRunLoop] addTimer:timer
forMode:NSDefaultRunLoopMode];
    }];
}

-(void)sendMessage:(NSTimer *const)timer {

```

```

PTDiffusionSession *const session = timer.userInfo;

const NSUInteger value = _nextValue++;
NSData *const data = [[NSString stringWithFormat:@"%lu",
(long)value] dataUsingEncoding:NSUTF8StringEncoding];
PTDiffusionContent *const content = [[PTDiffusionContent alloc]
initWithData:data];

NSLog(@"Sending %lu...", (long)value);
[session.messaging sendWithTopicPath:@"foo/bar"
                    value:content
                    options:[PTDiffusionSendOptions new]
                    completionHandler:^(NSError *const error)
{
    if (error) {
        NSLog(@"Failed to send. Error: %@", error);
    } else {
        NSLog(@"Sent");
    }
}];
}

@end

```

Java and Android

```

package com.pushtechology.diffusion.examples;

import java.util.List;

import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.features.Messaging;
import
    com.pushtechology.diffusion.client.features.Messaging.SendCallback;
import
    com.pushtechology.diffusion.client.features.Messaging.SendContextCallback;
import com.pushtechology.diffusion.client.session.Session;
import com.pushtechology.diffusion.datatype.json.JSON;

/**
 * This is a simple example of a client that uses the 'Messaging'
 * feature to
 * send messages to a topic path.
 * <P>
 * To send a message on a topic path, the client session requires the
 * 'send_to_message_handler' permission.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public final class ClientSendingMessages {

    private final Session session;
    private final Messaging messaging;

    /**
     * Constructs a message sending application.
     */
    public ClientSendingMessages() {
        session =
Diffusion.sessions().principal("client").password("password")

```

```

        .open("ws://diffusion.example.com:80");
        messaging = session.feature(Messaging.class);
    }

    /**
     * Sends a simple string message to a specified topic path.
     * <P>
     * There will be no context with the message so callback will be
directed to
     * the no context callback.
     *
     * @param topicPath the topic path
     * @param message the message to send
     * @param callback notifies message sent
     */
    public void send(String topicPath, String message, SendCallback
callback) {
        messaging.send(topicPath, message, callback);
    }

    /**
     * Sends a JSON object to a specified topic path.
     * <P>
     *
     * @param topicPath the topic path
     * @param message the JSON object to send
     * @param callback notifies message sent
     */
    public void send(String topicPath, JSON message, SendCallback
callback) {
        messaging.send(topicPath, message, callback);
    }

    /**
     * Sends a simple string message to a specified topic path with
context string.
     * <P>
     * Callback will be directed to the contextual callback with the
string
     * provided.
     *
     * @param topicPath the topic path
     * @param message the message to send
     * @param context the context string to return with the callback
     * @param callback notifies message sent
     */
    public void send(
        String topicPath,
        String message,
        String context,
        SendContextCallback<String> callback) {

        messaging.send(topicPath, message, context, callback);
    }

    /**
     * Sends a string message to a specified topic path with headers.
     * <P>
     * There will be no context with the message so callback will be
directed to
     * the no context callback.
     *
     * @param topicPath the topic path

```

```

    * @param message the message to send
    * @param headers the headers to send with the message
    * @param callback notifies message sent
    */
    public void sendWithHeaders(
        String topicPath,
        String message,
        List<String> headers,
        SendCallback callback) {

        messaging.send(
            topicPath,
            Diffusion.content().newContent(message),
            messaging.sendOptionsBuilder().headers(headers).build(),
            callback);
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }
}

```

.NET

```

using System.Collections.Generic;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Session;

namespace Examples {
    /// <summary>
    /// This is a simple example of a client that uses the
    'Messaging' feature to send messages on a topic path.
    ///
    /// To send messages on a topic path, the client session requires
    the
    /// <see cref="TopicPermission.SEND_TO_MESSAGE_HANDLER"/>
    permission.
    /// </summary>
    public class ClientSendingMessages {
        private readonly ISession session;
        private readonly IMessaging messaging;

        /// <summary>
        /// Constructs a message sending application.
        /// </summary>
        public ClientSendingMessages() {
            session =
            Diffusion.Sessions.Principal( "client" ).Password( "password" )
                .Open( "ws://diffusion.example.com:80" );

            messaging = session.GetMessagingFeature();
        }

        /// <summary>
        /// Sends a simple string message to a specified topic path.
        ///

```

```

        /// There will be no context with the message so callback
        will be directed to the 'no context' callback.
        /// </summary>
        /// <param name="topicPath">The topic path.</param>
        /// <param name="message">The message to send.</param>
        /// <param name="callback">Notifies that the message was
        sent.</param>
        public void Send( string topicPath, string message,
        ISendCallback callback ) {
            messaging.Send( topicPath,
            Diffusion.Content.NewContent( message ), callback );
        }

        /// <summary>
        /// Sends a simple string message to a specified topic path
        with context string.
        ///
        /// The callback will be directed to the contextual callback
        with the string provided.
        /// </summary>
        /// <param name="topicPath"></param>
        /// <param name="message"></param>
        /// <param name="context"></param>
        /// <param name="callback"></param>
        public void Send( string topicPath, string message, string
        context, ISendContextCallback<string> callback ) {
            messaging.Send( topicPath,
            Diffusion.Content.NewContent( message ), context, callback );
        }

        /// <summary>
        /// Sends a string message to a specified topic with headers.
        ///
        /// There will be no context with the message so callback
        will be directed to the 'no context' callback.
        /// </summary>
        /// <param name="topicPath">The topic path.</param>
        /// <param name="message">The message to send.</param>
        /// <param name="headers">The headers to send with the
        message.</param>
        /// <param name="callback">Notifies that the message was
        sent.</param>
        public void SendWithHeaders( string topicPath, string
        message, List<string> headers, ISendCallback callback ) {
            messaging.Send( topicPath,
            Diffusion.Content.NewContent( message ),

            messaging.CreateSendOptionsBuilder().SetHeaders( headers ).Build(),
            callback );
        }

        /// <summary>
        /// Close the session.
        /// </summary>
        public void Close() {
            session.Close();
        }
    }
}

```

C

```

/*
 * This example shows how a message can be sent from a client to a
 * message handler via a topic path.
 *
 * See msg-handler.c for an example of how to receive these messages
 * in a control client.
 */

#include <stdio.h>
#include <unistd.h>

#include <apr.h>
#include <apr_thread_mutex.h>
#include <apr_thread_cond.h>

#include "diffusion.h"
#include "args.h"

apr_pool_t *pool = NULL;
apr_thread_mutex_t *mutex = NULL;
apr_thread_cond_t *cond = NULL;

ARG_OPTS_T arg_opts[] = {
    ARG_OPTS_HELP,
    {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
    ARG_HAS_VALUE, "ws://localhost:8080"},
    {'p', "principal", "Principal (username) for the connection",
    ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    {'c', "credentials", "Credentials (password) for the
    connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    {'t', "topic", "Topic name", ARG_REQUIRED, ARG_HAS_VALUE,
    "echo"},
    {'d', "data", "Data to send", ARG_REQUIRED, ARG_HAS_VALUE,
    NULL},
    END_OF_ARG_OPTS
};

/*
 * Callback invoked when/if a message is sent on the topic that the
 * client is writing to.
 */
static int
on_send(SESSION_T *session, void *context)
{
    printf("on_send() successful. Context=\"%s\".\n", (char
    *)context);

    /*
     * Allow main thread to continue.
     */
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);

    return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
    /*

```

```

    * Standard command-line parsing.
    */
HASH_T *options = parse_cmdline(argc, argv, arg_opts);
if(options == NULL || hash_get(options, "help") != NULL) {
    show_usage(argc, argv, arg_opts);
    return EXIT_FAILURE;
}

char *url = hash_get(options, "url");
const char *principal = hash_get(options, "principal");
CREDENTIALS_T *credentials = NULL;
const char *password = hash_get(options, "credentials");
if(password != NULL) {
    credentials = credentials_create_password(password);
}
char *topic = hash_get(options, "topic");

/*
 * Setup for condition variable.
 */
apr_initialize();
apr_pool_create(&pool, NULL);
apr_thread_mutex_create(&mutex, APR_THREAD_MUTEX_UNNESTED,
pool);
apr_thread_cond_create(&cond, pool);

/*
 * Create a session with Diffusion.
 */
SESSION_T *session = NULL;
DIFFUSION_ERROR_T error = { 0 };
session = session_create(url, principal, credentials, NULL,
NULL, &error);
if(session == NULL) {
    fprintf(stderr, "TEST: Failed to create session\n");
    fprintf(stderr, "ERR : %s\n", error.message);
    return EXIT_FAILURE;
}

/*
 * Create a payload.
 */
char *data = hash_get(options, "data");
BUF_T *payload = buf_create();
buf_write_bytes(payload, data, strlen(data));

/*
 * Build up a list of some headers to send with the message.
 */
LIST_T *headers = list_create();
list_append_last(headers, "apple");
list_append_last(headers, "train");

/*
 * Parameters for send_msg() call.
 */
SEND_MSG_PARAMS_T params = {
    .topic_path = topic,
    .payload = *payload,
    .headers = headers,
    .priority = CLIENT_SEND_PRIORITY_NORMAL,
    .on_send = on_send,
    .context = "FOO"
}

```

```

    };

    /*
     * Send the message and wait for the callback to acknowledge
     * delivery.
     */
    apr_thread_mutex_lock(mutex);
    send_msg(session, params);
    apr_thread_cond_wait(cond, mutex);
    apr_thread_mutex_unlock(mutex);

    /*
     * Politely close the client connection and tidy up.
     */
    session_close(session, NULL);
    session_free(session);

    apr_thread_mutex_destroy(mutex);
    apr_thread_cond_destroy(cond);
    apr_pool_destroy(pool);
    apr_terminate();

    return EXIT_SUCCESS;
}

```

Change the URL from that provided in the example to the URL of the Diffusion server.

Example: Send a request message to the Push Notification Bridge

The following examples use the Unified API to send a request message on a topic path to communicate with the Push Notification Bridge. The request message is in JSON and can be used to subscribe or unsubscribe from receiving push notifications when specific topics are updated.

Apple

```

/**
 * Compose a URI understood by the Push Notification Bridge from an
 * APNs device token.
 * @param deviceID APNs device token.
 * @return string in format expected by the push notification bridge.
 */
-(NSString*)formatAsURI:(NSData*)deviceID {
    return [NSString stringWithFormat:@"apns://%@", [deviceID
    base64EncodedStringWithOptions:0]];
}

/**
 * Compose and send a subscription request to the Push Notification
 * bridge
 * @param paths topic paths within the subscription request
 */
-(void)doPnSubscribe:(NSArray<NSString*> *)paths deviceToken:
(NSData*)deviceToken {
    // Compose the JSON request from Obj-C literals
    NSString *const correlation = [[NSUUID UUID] UUIDString];
    PTDiffusionTopicSelector *const selector =
    [PTDiffusionTopicSelector topicSelectorWithAnyExpression:paths];
    NSDictionary *const request =
        @{@"request": @{
            @"correlation": correlation,
            @"content": @{

```

```

        @"pnsub": @{
            @"destination": [self formatAsURI:deviceToken],
            @"topic": selector.description}
        }
    }];
    NSData *const requestData = [NSJSONSerialization
dataWithJSONObject:request options:0 error:nil];

    // Send a message to `SERVICE_TOPIC`
    [_session.messaging sendWithTopicPath:SERVICE_TOPIC
                        value:[[PTDiffusionContent alloc]
initWithData:requestData]
                        completionHandler:^(NSError * _Nullable
error)
    {
        if(error != nil) {
            NSLog(@"Send to topic %@ failed:
%@", SERVICE_TOPIC, error);
        }
    }];
}

```

Android

```

import static java.util.UUID.randomUUID;

import java.io.PrintStream;

import org.json.JSONObject;
import org.json.JSONTokener;

import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.content.Content;
import com.pushtechology.diffusion.client.features.Messaging;
import
    com.pushtechology.diffusion.client.features.Messaging.MessageStream;
import
    com.pushtechology.diffusion.client.features.Messaging.SendCallback;
import com.pushtechology.diffusion.client.session.Session;
import com.pushtechology.diffusion.client.types.ReceiveContext;

/**
 * An example of a client using the 'Messaging' feature to request
 * the Push Notification Bridge
 * subscribe to a topic and relay updates to a GCM registration ID.
 *
 * @author Push Technology Limited
 * @since 5.9
 */
public class ClientSendingPushNotificationSubscription {

    private static final PrintStream OUT = System.out;

    private final String pushServiceTopicPath;
    private final Session session;
    private final Messaging messaging;
    private final MessageStream messageStream = new
MessageStream.Default() {
        @Override
        public void onMessageReceived(String topicPath, Content
content, ReceiveContext context) {

```

```

        final JSONObject response = (JSONObject) new
JSONTokener(content.asString()).nextValue();
        final String correlation =
response.getJSONObject("response").getString("correlation");

        OUT.printf("Received response with correlation '%s': %s",
correlation, response);
    } }];

/**
 * Constructs message sending application.
 * @param pushServiceTopicPath topic path on which the Push
Notification Bridgre is taking requests.
 */
public ClientSendingPushNotificationSubscription(String
pushServiceTopicPath) {
    this.pushServiceTopicPath = pushServiceTopicPath;
    this.session =

Diffusion.sessions().principal("client").password("password")
        .open("ws://diffusion.example.com:80");
    this.messaging = session.feature(Messaging.class);
    messaging.addMessageStream(pushServiceTopicPath,
messageStream);
}

/**
 * Close the session.
 */
public void close() {
    messaging.removeMessageStream(messageStream);
    session.close();
}

/**
 * Compose & send a subscription request to the Push Notification
Bridge.
 *
 * @param subscribedTopic topic to which the bridge subscribes.
 * @param gcmRegistrationID GCM registration ID to which the
bridge relays updates.
 */
public void requestPNSubscription(String gcmRegistrationID,
String subscribedTopic) {
    // Compose the request
    final String gcmDestination = "gcm://" + gcmRegistrationID;
    final String correlation = randomUUID().toString();
    final JSONObject request =
buildSubscriptionRequest(gcmDestination, subscribedTopic,
correlation);

    // Send the request
    messaging.send(pushServiceTopicPath, request.toString(), new
SendCallback.Default());
}

/**
 * Compose a subscription request.
 * <P>
 * @param destination The {@code gcm://} or {@code apns://}
destination for any push notifications.
 * @param topic Diffusion topic subscribed-to by the Push
Notification Bridge.

```

```

    * @param correlation value embedded in the response by the
    bridge relating it back to the request.
    * @return a complete request
    */
    private static JSONObject buildSubscriptionRequest(String
destination, String topic, String correlation) {
        final JSONObject subObject = new JSONObject();
        subObject.put("destination", destination);
        subObject.put("topic", topic);

        final JSONObject contentObj = new JSONObject();
        contentObj.put("pnsub", subObject);

        final JSONObject requestObj = new JSONObject();
        requestObj.put("correlation", correlation);
        requestObj.put("content", contentObj);

        final JSONObject rootObject = new JSONObject();
        rootObject.put("request", requestObj);
        return rootObject;
    }
}

```

Related concepts

[Push notification networks](#) on page 123

Consider whether your solution will interact with push notification networks.

[Push Notification Bridge persistence plugin](#) on page 511

The Push Notification Bridge stores subscription information in memory. To persist this information past the end of the bridge process, implement a persistence plugin.

[Push Notification Bridge](#) on page 660

The Push Notification Bridge is a Diffusion client that subscribes to topics on behalf of other Diffusion clients and uses a push notification network to relay topic updates to the device where the client application is located.

Messaging to sessions

A client session can use the MessagingControl feature to send individual messages to any known client session on any topic path. It can also register a handler for messages sent from client sessions.

Sending messages to sessions

Required permissions: view_session, send_to_session

A client session can send a message to any known client session on any topic path, regardless of whether a topic is bound to that topic path. The messages are delivered to other client sessions through the Messaging feature or through the topic listener mechanism.

The client requires the session identifier of the client session it is sending to. Use the ClientControl feature to get the identifiers for connected client sessions. A client session can also send messages back to client sessions from which it receives messages through a *message handler*.

The body of the message that is sent is represented as *bytes*. Any of the builder features can be used to build a message as *content*, JSON, or binary. With messaging you can also send an empty message.

When sending a message certain additional options can also be specified:

Headers

A set of string values that can be sent along with the content.

Priority

Use this to specify the priority used when queuing the message for the client session at the Diffusion server.

Filtering message recipients using session properties

Required permissions: view_session

When sending a message, a client session can specify a filter for the recipients of that message. The filter is a query expression on the values of session properties.

The client session defines a filter and sends a message on a topic path with the filter associated. The Diffusion server evaluates the query and sends the message on to connected client sessions whose session properties match the filter.

Note: Sending messages to a set of client sessions defined by a filter is not intended for high throughput or data. If you have a lot of data to send or want to send data to a lot of client sessions, use the pub-sub capabilities of Diffusion. Subscribe the set of client sessions to a topic and publish the data as updates through that topic.

Registering message handlers

Required permissions: register_handler

A client session can add a message handler for any branch of the topic tree. This handler receives messages sent from client sessions on any topic in that branch unless overridden by a handler registered for a more specific branch.

Each client session can register only a single handler for any branch in the topic tree. To change the handler for a particular branch, the previous handler must be closed.

A message received by a message handler comprises content and message context which can include headers. The content can be empty. The client interprets the content of messages. If a topic is bound to the topic path that the message is sent on, message content is not required to match the content definition of that topic.

Requesting session properties with messages

Required permissions: register_handler,view_session

When registering a message handler, a client can specify session properties that it is interested in receiving with the handled messages.

Messages received by this message handler include the current values of the requested session properties for the client session that sent the message.

The message handler can request both fixed properties and user-defined properties of the session.

Related concepts

[Session properties](#) on page 267

A client session has a number of properties associated with it. Properties are key-value pairs. Both the key and the value are case sensitive.

[Session filtering](#) on page 268

Session filters enable you to query the set of connected client sessions on the Diffusion server based on their session properties.

Example: Handle messages and send messages to sessions

The following examples use the MessagingControl feature in the Unified API to handle messages sent to topic paths and to send messages to one or more clients.

JavaScript

```
var diffusion = require('diffusion');

// Connect to the server. Change these options to suit your own
// environment.
// Node.js will not accept self-signed certificates by default. If
// you have
// one of these, set the environment variable
// NODE_TLS_REJECT_UNAUTHORIZED=0
// before running this example.
diffusion.connect({
  host    : 'diffusion.example.com',
  port    : 443,
  secure  : true,
  principal : 'control',
  credentials : 'password'
}).then(function(session) {
  // 1. Messages can be sent & received between sessions.

  // Create a stream of received messages for a specific path
  session.messages.listen('foo').on('message', function(msg) {
    console.log('Received message: ' + msg.content);
  });

  // Send a message to another session. It is the application's
  // responsibility to find the SessionID of the intended
  // recipient.
  session.messages.send('foo', 'Hello world', 'another-session-
  id');

  // 2. Messages can also be sent without a recipient, in which
  // case they will be dispatched to any Message Handlers
  // that have been registered for the same path. If multiple
  // handlers are registered to the same path, any given
  // message will only be dispatched to one handler.

  // Register the handler to receive messages at or below the given
  // path.
  session.messages.addHandler('foo', {
    onActive : function() {
      console.log('Handler registered');
    },
    onClose  : function() {
      console.log('Handler closed');
    },
    onMessage : function(msg) {
      console.log('Received message:' + msg.content + ' from
      Session: ' + msg.session);
      if (msg.properties) {
        console.log('with properties:', msg.properties);
      }
    }
  });
});
```

```

    }
  }).then(function() {
    console.log('Registered handler');
  }, function(e) {
    console.log('Failed to register handler: ', e);
  });

  // Send a message at a lower path, without an explicit recipient
  - this will be received by the Handler.
  session.messages.send('foo/bar', 'Another message');
});

```

Java and Android

```

import org.json.JSONException;
import org.json.JSONObject;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.content.Content;
import com.pushtechology.diffusion.client.features.Messaging;
import
  com.pushtechology.diffusion.client.features.control.topics.MessagingControl;
import
  com.pushtechology.diffusion.client.features.control.topics.MessagingControl.Me
import
  com.pushtechology.diffusion.client.features.control.topics.MessagingControl.Se
import com.pushtechology.diffusion.client.session.Session;
import com.pushtechology.diffusion.client.session.SessionId;
import com.pushtechology.diffusion.client.types.ReceiveContext;
import com.pushtechology.diffusion.datatype.json.JSON;

/**
 * This is an example of a control client using the
 * 'MessagingControl' feature
 * to receive messages from clients and also send messages to
 * clients.
 * <P>
 * It is a trivial example that simply responds to all messages on a
 * particular
 * branch of the topic tree by echoing them back to the client
 * exactly as they
 * are complete with headers.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientReceivingMessages {

  private final Session echoingSession;
  private final Session sendingSession;
  private final MessagingControl echoingSessionMessagingControl;
  private final MessagingControl sendingSessionMessagingControl;
  private final SendCallback sendCallback;

  private static final Logger LOG =

  LoggerFactory.getLogger(ControlClientReceivingMessages.class);

  /**
   * Constructor.

```

```

    *
    * @param callback for result of sends
    */
    public ControlClientReceivingMessages(SendCallback callback) {

        sendCallback = callback;

        echoingSession =

Diffusion.sessions().principal("control").password("password")
        .open("ws://diffusion.example.com:80");

        sendingSession =

Diffusion.sessions().principal("control").password("password")
        .open("ws://diffusion.example.com:80");

        echoingSessionMessagingControl =
echoingSession.feature(MessagingControl.class);
        sendingSessionMessagingControl =
sendingSession.feature(MessagingControl.class);

        // Register to receive all messages sent by clients on the
"foo" branch
        // To do this, the client session must have the
'register_handler' permission.
        echoingSessionMessagingControl.addMessageHandler("foo", new
EchoHandler());
    }

    /**
     * Close the session.
     */
    public void close() {
        echoingSession.close();
        sendingSession.close();
    }

    /**
     * Handler that echoes messages back to the originating client
complete with
     * original headers.
     */
    private class EchoHandler extends MessageHandler.Default {
        @Override
        public void onMessage(
            SessionId sessionId,
            String topicPath,
            Content content,
            ReceiveContext context) {

            try {
                final JSONObject jsonObject = new
JSONObject(content.asString());
                final String value = (String)
jsonObject.get("hello");
                LOG.info("JSON content with key: 'hello' and value:
'{}'", value);
            }
            catch (JSONException e) {
                //Non-JSON message so just carry on and echo the
message
            }
        }
    }

```

```

        // To send a message to a client, this client session
must have // the 'view_session' and 'send_to_session' permissions.
        echoingSessionMessagingControl.send(
            sessionId,
            topicPath,
            content,
            echoingSessionMessagingControl.sendOptionsBuilder()
                .headers(context.getHeaderList())
                .build(),
            sendCallback);
    }
}

/**
 * Add a message stream to observe echoed messages.
 *
 * @param stream stream to be added
 */
public void
addSendingSessionMessageStream(Messaging.MessageStream stream) {

    sendingSession.feature(Messaging.class).addMessageStream("foo",
stream);
}

/**
 * Sends messages "hello:world" and "{\"hello\":\"world\"}".
 */
public void sendHelloWorld() {
    final Content helloWorldContent =
Diffusion.content().newContent("hello:world");
    final JSON helloWorldJson =
Diffusion.dataTypes().json().fromJsonString("{\"hello\":\"world
\"}");

    //To do this, the client session must have the 'view_session'
and 'send_to_session' permissions.

    sendingSessionMessagingControl.send(echoingSession.getSessionId(),
"foo", helloWorldContent, sendCallback);

    sendingSessionMessagingControl.send(echoingSession.getSessionId(),
"foo", helloWorldJson, sendCallback);
}
}

```

.NET

```

using System.Linq;
using PushTechnology.ClientInterface.Client.Content;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Types;

namespace Examples {
    /// <summary>

```

```

    /// This is an example of a control client using the <see
    cref="IMessagingControl"/> feature to receive messages
    /// from clients and also send messages to clients.
    ///
    /// It is a trivial example that simply responds to all messages
    on a particular branch of the topic tree by
    /// echoing them back to the client exactly as they are, complete
    with headers.
    /// </summary>
    public class ControlClientReceivingMessages {
        private readonly ISession session;

        /// <summary>
        /// Constructor.
        /// </summary>
        /// <param name="callback">The callback to receive the result
of message sending.</param>
        public ControlClientReceivingMessages( ISendCallback
callback ) {
            session =
Diffusion.Sessions.Principal( "control" ).Password( "password" )
                .Open( "ws://diffusion.example.com:80" );

            var messagingControl =
session.GetMessagingControlFeature();

            // Register to receive all messages sent by clients on
the "foo" branch.
            // To do this, the client session must have the
REGISTER_HANDLER permission.
            messagingControl.AddMessageHandler( "foo", new
EchoHandler( messagingControl, callback ) );
        }

        /// <summary>
        /// Close the session.
        /// </summary>
        public void Close() {
            session.Close();
        }

        private class EchoHandler : MessageHandlerDefault {
            private readonly IMessagingControl theMessagingControl;
            private readonly ISendCallback theSendCallback;

            public EchoHandler( IMessagingControl messagingControl,
ISendCallback sendCallback ) {
                theMessagingControl = messagingControl;
                theSendCallback = sendCallback;
            }

            /// <summary>
            /// Receives content sent from a session via a topic.
            /// </summary>
            /// <param name="sessionId">Identifies the client session
that sent the content.</param>
            /// <param name="topicPath">The path of the topic that
the content was sent on.</param>
            /// <param name="content">The content sent by the
client.</param>
            /// <param name="context">The context associated with the
content.</param>

```

```

        public override void OnMessage( SessionId sessionId,
string topicPath, IContent content,
        IReceiveContext context ) {
            theMessagingControl.Send( sessionId, topicPath,
content,
theMessagingControl.CreateSendOptionsBuilder().SetHeaders( context.HeadersList,
        theSendCallback );
        }
    }

    private class MessageHandlerDefault :
TopicTreeHandlerDefault, IMessageHandler {
        /// <summary>
        /// Receives content sent from a session via a topic.
        /// </summary>
        /// <param name="sessionId">Identifies the client session
that sent the content.</param>
        /// <param name="topicPath">The path of the topic that
the content was sent on.</param>
        /// <param name="content">The content sent by the
client.</param>
        /// <param name="context">The context associated with the
content.</param>
        public virtual void OnMessage( SessionId sessionId,
string topicPath, IContent content,
        IReceiveContext context ) {
        }
    }
}

```

C - Receive

```

/*
 * This example shows how to receive messages, rather than topic
 * updates, as part of MessagingControl.
 *
 * You may register a handler against an endpoint, which will
 * become the only destination for messages to that endpoint (where
 * the control client which is considered "active" is determined by
 * the server).
 *
 * See send-msg.c for an example of how to send messages to an
 * endpoint from a client.
 */

#include <stdio.h>
#include <unistd.h>

#include "diffusion.h"
#include "conversation.h"
#include "args.h"

ARG_OPTS_T arg_opts[] = {
    ARG_OPTS_HELP,
    {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
ARG_HAS_VALUE, "ws://localhost:8080"},
    {'p', "principal", "Principal (username) for the connection",
ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    {'c', "credentials", "Credentials (password) for the
connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},

```

```

        {'t', "topic", "Topic name", ARG_REQUIRED, ARG_HAS_VALUE,
"echo"},
        END_OF_ARG_OPTS
};

/*
 * Function to be called when the message receiver has been
registered.
 */
int
on_registered(SESSION_T *session, void *context)
{
    printf("on_registered()\n");
    return HANDLER_SUCCESS;
}

/*
 * Function called on receipt of a message from a client.
 *
 * We print the following information:
 * 1. The topic path on which the message was received.
 * 2. A hexdump of the message content.
 * 3. The headers associated with the message.
 * 4. The session properties that were requested when the handler
was
 * registered.
 * 5. The user context, as a string.
 */
int
on_msg(SESSION_T *session, const SVC_SEND_RECEIVER_CLIENT_REQUEST_T
*request, void *context)
{
    printf("Received message on topic path %s\n", request-
>topic_path);
    hexdump_buf(request->content->data);
    printf("Headers:\n");
    if(request->send_options.headers->first == NULL) {
        printf(" No headers\n");
    }
    else {
        for(LIST_NODE_T *node = request-
>send_options.headers->first;
            node != NULL;
            node = node->next) {
            printf(" Header: %s\n", (char *)node->data);
        }
    }

    printf("Session properties:\n");
    char **keys = hash_keys(request->session_properties);
    if(keys == NULL || *keys == NULL) {
        printf(" No properties\n");
    }
    else {
        for(char **k = keys; *k != NULL; k++) {
            char *v = hash_get(request-
>session_properties, *k);
            printf(" %s=%s\n", *k, v);
        }
    }
    free(keys);

    if(context != NULL) {

```

```

        printf("Context: %s\n", (char *)context);
    }

    return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
    /*
     * Standard command-line parsing.
     */
    HASH_T *options = parse_cmdline(argc, argv, arg_opts);
    if(options == NULL || hash_get(options, "help") != NULL) {
        show_usage(argc, argv, arg_opts);
        return EXIT_FAILURE;
    }

    char *url = hash_get(options, "url");
    const char *principal = hash_get(options, "principal");
    CREDENTIALS_T *credentials = NULL;
    const char *password = hash_get(options, "credentials");
    if(password != NULL) {
        credentials = credentials_create_password(password);
    }
    char *topic = hash_get(options, "topic");

    /*
     * Create a session with Diffusion.
     */
    SESSION_T *session = NULL;
    DIFFUSION_ERROR_T error = { 0 };
    session = session_create(url, principal, credentials, NULL,
    NULL, &error);
    if(session == NULL) {
        fprintf(stderr, "TEST: Failed to create session\n");
        fprintf(stderr, "ERR : %s\n", error.message);
        return EXIT_FAILURE;
    }

    char *session_id = session_id_to_string(session->id);
    printf("Session created, id=%s\n", session_id);
    free(session_id);

    /*
     * Register a message handler, and for each message ask for
     * the $Principal property to be provided.
     */
    LIST_T *requested_properties = list_create();
    list_append_last(requested_properties, "$Principal");

    MSG_RECEIVER_REGISTRATION_PARAMS_T params = {
        .on_registered = on_registered,
        .topic_path = topic,
        .on_message = on_msg,
        .session_properties = requested_properties
    };
    list_free(requested_properties, free);

    register_msg_handler(session, params);

    /*
     * Accept messages for a while, then deregister.

```

```

        */
        sleep(30);

        deregister_msg_handler(session, params);

        /*
         * Close session and clean up.
         */
        session_close(session, NULL);
        session_free(session);

        list_free(requested_properties, NULL);

        return EXIT_SUCCESS;
}

```

C - Send

```

/*
 * This example shows how a message can be sent to another client via
 * a topic endpoint. The session ID of the target client must be
 * known.
 *
 * See msg-listener.c for an example of how to receive these messages
 * in a client.
 */

#include <stdio.h>
#include <unistd.h>

#include <apr.h>
#include <apr_thread_mutex.h>
#include <apr_thread_cond.h>

#include "diffusion.h"
#include "args.h"

apr_pool_t *pool = NULL;
apr_thread_mutex_t *mutex = NULL;
apr_thread_cond_t *cond = NULL;

ARG_OPTS_T arg_opts[] = {
    ARG_OPTS_HELP,
    {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
     ARG_HAS_VALUE, "ws://localhost:8080"},
    {'p', "principal", "Principal (username) for the connection",
     ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    {'c', "credentials", "Credentials (password) for the
     connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    {'t', "topic", "Topic name", ARG_REQUIRED, ARG_HAS_VALUE,
     "echo"},
    {'s', "session_id", "Session id", ARG_REQUIRED,
     ARG_HAS_VALUE, NULL},
    {'d', "data", "Data to send", ARG_REQUIRED, ARG_HAS_VALUE,
     NULL},
    END_OF_ARG_OPTS
};

/*
 * Callback invoked when/if a message is published on the topic that
 * the
 * client is writing to.

```

```

*/
static int
on_send(SESSION_T *session, void *context)
{
    printf("on_send() successful. Context=\"%s\".\n", (char
    *)context);

    /*
     * Allow main thread to continue.
     */
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);

    return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
    /*
     * Standard command-line parsing.
     */
    HASH_T *options = parse_cmdline(argc, argv, arg_opts);
    if(options == NULL || hash_get(options, "help") != NULL) {
        show_usage(argc, argv, arg_opts);
        return EXIT_FAILURE;
    }

    char *url = hash_get(options, "url");

    const char *principal = hash_get(options, "principal");
    CREDENTIALS_T *credentials = NULL;

    const char *password = hash_get(options, "credentials");
    if(password != NULL) {
        credentials = credentials_create_password(password);
    }

    char *topic = hash_get(options, "topic");
    char *session_id_str = hash_get(options, "session_id");

    /*
     * Setup for condition variable.
     */
    apr_initialize();
    apr_pool_create(&pool, NULL);
    apr_thread_mutex_create(&mutex, APR_THREAD_MUTEX_UNNESTED,
pool);
    apr_thread_cond_create(&cond, pool);

    /*
     * Create a session with Diffusion.
     */
    SESSION_T *session = NULL;
    DIFFUSION_ERROR_T error = { 0 };
    session = session_create(url, principal, credentials, NULL,
NULL, &error);
    if(session == NULL) {
        fprintf(stderr, "TEST: Failed to create session\n");
        fprintf(stderr, "ERR : %s\n", error.message);
        return EXIT_FAILURE;
    }
}

```

```

    /*
     * Create a payload.
     */
    char *data = hash_get(options, "data");
    BUF_T *payload = buf_create();
    buf_write_bytes(payload, data, strlen(data));
    CONTENT_T *content = content_create(CONTENT_ENCODING_NONE,
payload);
    buf_free(payload);

    /*
     * Build up some headers to send with the message.
     */
    LIST_T *headers = list_create();
    list_append_last(headers, "apple");
    list_append_last(headers, "train");

    /*
     * Parameters for send_msg_to_session() call.
     */
    SESSION_ID_T *session_id =
session_id_create_from_string(session_id_str);

    SEND_MSG_TO_SESSION_PARAMS_T params = {
        .topic_path = topic,
        .session_id = *session_id,
        .content = *content,
        .options.headers = headers,
        .options.priority = CLIENT_SEND_PRIORITY_NORMAL,
        .on_send = on_send,
        .context = "FOO"
    };

    /*
     * Send the message and wait for the callback to acknowledge
     * delivery.
     */
    apr_thread_mutex_lock(mutex);
    send_msg_to_session(session, params);
    apr_thread_cond_wait(cond, mutex);
    apr_thread_mutex_unlock(mutex);

    /*
     * Politely close the client connection and clean up.
     */
    session_id_free(session_id);
    session_close(session, NULL);
    session_free(session);

    apr_thread_mutex_destroy(mutex);
    apr_thread_cond_destroy(cond);
    apr_pool_destroy(pool);
    apr_terminate();

    return EXIT_SUCCESS;
}

```

Change the URL from that provided in the example to the URL of the Diffusion server.

Example: Use session property filters with messaging

The following examples use the MessagingControl feature in the Unified API to request session properties with messages sent to topic paths and to send messages to one or more clients depending on the values of their session properties.

JavaScript

```
var diffusion = require('../..../js-uci-client/src/diffusion');

// Connect to the server. Change these options to suit your own
// environment.
// Node.js will not accept self-signed certificates by default. If
// you have
// one of these, set the environment variable
// NODE_TLS_REJECT_UNAUTHORIZED=0
// before running this example.
diffusion.connect({
  host      : 'diffusion.example.com',
  port      : 443,
  secure    : true,
  principal : 'control',
  credentials : 'password'
}).then(function(session) {

  // Create a listener for a stream of messages on a specific path.
  session.messages.listen('foo').on('message', function(msg) {
    console.log('Received message: ' + msg.content);
  });

  // Send a message to another session listening on 'foo' by way of
  // session properties.
  session.messages.send('foo', 'Hello world', '$Principal is
  "control"');
});
```

Java and Android

```
package com.pushtechology.diffusion.examples;

import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.content.Content;
import
  com.pushtechology.diffusion.client.features.control.topics.MessagingControl;
import
  com.pushtechology.diffusion.client.features.control.topics.MessagingControl.Me
import
  com.pushtechology.diffusion.client.features.control.topics.MessagingControl.Se
import com.pushtechology.diffusion.client.session.Session;
import com.pushtechology.diffusion.client.session.SessionId;
import com.pushtechology.diffusion.client.types.ReceiveContext;

/**
 * This is an example of a control client using the
 * 'MessagingControl' feature
 * to send messages to clients using message filters. It also
 * demonstrates the
 * ability to register a message handler with an interest in session
 * property
 * values.
 *
 */
```

```

* @author Push Technology Limited
* @since 5.5
*/
public final class ControlClientUsingFiltersAndProperties {

    private final Session session;
    private final MessagingControl messagingControl;
    private final SendToFilterCallback sendToFilterCallback;

    /**
     * Constructor.
     *
     * @param callback for result of sends
     */
    public
ControlClientUsingFiltersAndProperties(SendToFilterCallback
callback) {

        sendToFilterCallback = callback;

        session =

Diffusion.sessions().principal("control").password("password")
        .open("ws://diffusion.example.com:80");
        messagingControl = session.feature(MessagingControl.class);

        // Register to receive all messages sent by clients on the
"foo" branch
        // and include the "JobTitle" session property value with
each message.
        // To do this, the client session must have the
'register_handler'
        // permission.
        messagingControl.addMessageHandler(
            "foo", new BroadcastHandler(), "JobTitle");
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }

    /**
     * Handler that will pass any message to all sessions that have a
"JobTitle"
     * property set to "Staff" if, and only if it comes from a
session that has
     * a "JobTitle" set to "Manager".
     */
    private class BroadcastHandler extends MessageHandler.Default {
        @Override
        public void onMessage(
            SessionId sessionId,
            String topicPath,
            Content content,
            ReceiveContext context) {

            if
("Manager".equals(context.getSessionProperties().get("JobTitle"))) {
                messagingControl.sendToFilter(
                    "JobTitle is 'Staff'",

```

```

        topicPath,
        content,
        messagingControl.sendOptionsBuilder()
            .headers(context.getHeaderList())
            .build(),
        sendToFilterCallback);
    }
}
}
}
}

```

.NET

```

using System.Linq;
using PushTechnology.ClientInterface.Client.Content;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Types;

namespace Examples {
    /// <summary>
    /// This is an example of a control client using the
    /// 'MessagingControl' feature to send messages to clients using
    /// message filters. It also demonstrates the ability to register
    a message handler with an interest in session
    /// property values.
    /// </summary>
    public class ControlClientUsingFiltersAndProperties {
        private readonly ISession theSession;
        private readonly IMessagingControl theMessagingControl;
        private readonly ISendToFilterCallback
theSendToFilterCallback;

        public
ControlClientUsingFiltersAndProperties( ISendToFilterCallback
callback ) {
            theSendToFilterCallback = callback;

            theSession =
Diffusion.Sessions.Principal( "control" ).Password( "password" )
                .Open( "ws://diffusion.example.com:80" );

            theMessagingControl =
theSession.GetMessagingControlFeature();

            // Register and receive all messages sent by clients on
the "foo" branch and include the "JobTitle" session
            // property value with each message. To do this, the
client session must have the "register_handler"
            // permission.
            theMessagingControl.AddMessageHandler(
                "foo",
                new BroadcastHandler( theMessagingControl,
theSendToFilterCallback ),
                "JobTitle" );
        }

        public void Close() {

```

```

        theSession.Close();
    }

    private class BroadcastHandler : IMessageHandler {
        private readonly IMessagingControl theMessagingControl;
        private readonly ISendToFilterCallback
theSendToFilterCallback;

        /// <summary>
        /// Constructor.
        /// </summary>
        /// <param name="messagingControl">The messaging control
object.</param>
        /// <param name="callback">The filter callback.</param>
        public BroadcastHandler( IMessagingControl
messagingControl, ISendToFilterCallback callback ) {
            theMessagingControl = messagingControl;
            theSendToFilterCallback = callback;
        }

        /// <summary>
        /// Called when the handler has been successfully
registered with the server.
        ///
        /// A session can register a single handler of each type
for a given branch of the topic tree. If there is
        /// already a handler registered for the topic path the
operation will fail, <c>registeredHandler</c> will
        /// be closed, and the session error handler will be
notified. To change the handler, first close the
        /// previous handler.
        /// </summary>
        /// <param name="topicPath">The path that the handler is
active for.</param>
        /// <param name="registeredHandler">Allows the handler to
be closed.</param>
        public void OnActive( string topicPath,
IRegisteredHandler registeredHandler ) {
        }

        /// <summary>
        /// Called if the handler is closed. This happens if the
call to register the handler fails, or the handler
        /// is unregistered.
        /// </summary>
        /// <param name="topicPath">The branch of the topic tree
for which the handler was registered.</param>
        public void OnClose( string topicPath ) {
        }

        /// <summary>
        /// Receives content sent from a session via a topic.
        /// </summary>
        /// <param name="sessionId">Identifies the client session
that sent the content.</param>
        /// <param name="topicPath">The path of the topic that
the content was sent on.</param>
        /// <param name="content">The content sent by the
client.</param>
        /// <param name="context">The context associated with the
content.</param>
        public void OnMessage( SessionId sessionId, string
topicPath, IContent content, IReceiveContext context ) {

```



```

{
    printf("on_send() successful. Context=\"%s\".\n", (char
*)context);
    printf("Sent message to %d clients\n", response->sent_count);

    if(response->error_reports != NULL && response-
>error_reports->first != NULL) {
        LIST_NODE_T *node = response->error_reports->first;
        while(node != NULL) {
            ERROR_REPORT_T *err = (ERROR_REPORT_T *)node-
>data;
            printf("Error: %s at line %d, column %d\n",
err->message, err->line, err->column);
            node = node->next;
        }
    }
    else {
        printf("No errors reported\n");
    }

    /*
    * Allow main thread to continue.
    */
    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);

    return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
    /*
    * Standard command-line parsing.
    */
    HASH_T *options = parse_cmdline(argc, argv, arg_opts);
    if(options == NULL || hash_get(options, "help") != NULL) {
        show_usage(argc, argv, arg_opts);
        return EXIT_FAILURE;
    }

    char *url = hash_get(options, "url");

    const char *principal = hash_get(options, "principal");
    CREDENTIALS_T *credentials = NULL;

    const char *password = hash_get(options, "credentials");
    if(password != NULL) {
        credentials = credentials_create_password(password);
    }

    char *topic = hash_get(options, "topic");
    char *filter = hash_get(options, "filter");

    /*
    * Setup for condition variable.
    */
    apr_initialize();
    apr_pool_create(&pool, NULL);
    apr_thread_mutex_create(&mutex, APR_THREAD_MUTEX_UNNESTED,
pool);
    apr_thread_cond_create(&cond, pool);

```

```

/*
 * Create a session with Diffusion.
 */
SESSION_T *session = NULL;
DIFFUSION_ERROR_T error = { 0 };
session = session_create(url, principal, credentials, NULL,
NULL, &error);
if(session == NULL) {
    fprintf(stderr, "TEST: Failed to create session\n");
    fprintf(stderr, "ERR : %s\n", error.message);
    return EXIT_FAILURE;
}

/*
 * Create a payload.
 */
char *data = hash_get(options, "data");
BUF_T *payload = buf_create();
buf_write_bytes(payload, data, strlen(data));
CONTENT_T *content = content_create(CONTENT_ENCODING_NONE,
payload);
buf_free(payload);

/*
 * Build up some headers to send with the message.
 */
LIST_T *headers = list_create();
list_append_last(headers, "apple");
list_append_last(headers, "train");

/*
 * Parameters for send_msg_to_session() call.
 */
SEND_MSG_TO_FILTER_PARAMS_T params = {
    .topic_path = topic,
    .filter = filter,
    .content = *content,
    .options.headers = headers,
    .options.priority = CLIENT_SEND_PRIORITY_NORMAL,
    .on_send = on_send,
    .context = "FOO"
};

/*
 * Send the message and wait for the callback to acknowledge
delivery.
 */
apr_thread_mutex_lock(mutex);
send_msg_to_filter(session, params);
apr_thread_cond_wait(cond, mutex);
apr_thread_mutex_unlock(mutex);

/*
 * Close session and clean up.
 */
session_close(session, NULL);
session_free(session);

apr_thread_mutex_destroy(mutex);
apr_thread_cond_destroy(cond);
apr_pool_destroy(pool);
apr_terminate();

```

```
}        return EXIT_SUCCESS;
```

Change the URL from that provided in the example to the URL of the Diffusion server.

Authenticating clients

A client can use the `AuthenticationControl` feature to authenticate other client sessions.

Registering a control authentication handler

Required permissions: `authenticate`, `register_handler`

A client can register an authentication handler that can be called when a client connects to the Diffusion server or changes the principal and credentials it is connected with.

The authentication handler can decide whether a client's authentication request is allowed or denied, or the authentication handler can abstain from the decision. In which case the next configured authentication handler is called.

If the authentication handler allows a client's authentication request, it can assign roles to that client's session.

For more information about authentication and role-based security, see [Authentication](#) on page 140.

Related concepts

[Configuring authentication handlers](#) on page 561

Authentication handlers and the order that the Diffusion server calls them in are configured in the `Server.xml` configuration file.

Example: Register an authentication handler

The following examples use the Diffusion Unified API to register a control authentication handler with the Diffusion server. The examples also include a simple or empty authentication handler.

The name by which the control authentication handler is registered must be configured in the `Server.xml` configuration file of the Diffusion server for the control authentication handler to be called to handle authentication requests.

Java and Android

```
package com.pushtechology.diffusion.examples;

import java.nio.charset.Charset;
import java.util.Arrays;
import java.util.EnumSet;

import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.details.SessionDetails;
import
    com.pushtechology.diffusion.client.details.SessionDetails.DetailType;
import com.pushtechology.diffusion.client.features.ServerHandler;
import
    com.pushtechology.diffusion.client.features.control.clients.AuthenticationCont
import
    com.pushtechology.diffusion.client.features.control.clients.AuthenticationCont
```

```

import com.pushtechology.diffusion.client.session.Session;
import com.pushtechology.diffusion.client.types.Credentials;

/**
 * This demonstrates the use of a control client to authenticate
 * client
 * connections.
 * <P>
 * This uses the 'AuthenticationControl' feature.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientIdentityChecks {

    private final Session session;

    /**
     * Constructor.
     */
    public ControlClientIdentityChecks() {

        session =

Diffusion.sessions().principal("control").password("password")
        .open("ws://diffusion.example.com:80");

        final AuthenticationControl authenticationControl =
            session.feature(AuthenticationControl.class);

        // To register the authentication handler, this client
        session must
        // have the 'authenticate' and 'register_handler'
        permissions.
        authenticationControl.setAuthenticationHandler(
            "example-handler",
            EnumSet.allOf(DetailType.class),
            new Handler());
    }

    /**
     * Authentication handler.
     */
    private static class Handler extends ServerHandler.Default
        implements ControlAuthenticationHandler {
        @Override
        public void authenticate(
            final String principal,
            final Credentials credentials,
            final SessionDetails sessionDetails,
            final Callback callback) {

            final byte[] passwordBytes =
                "password".getBytes(Charset.forName("UTF-8"));

            if ("admin".equals(principal) &&
                credentials.getType() ==
Credentials.Type.PLAIN_PASSWORD &&
                Arrays.equals(credentials.toBytes(), passwordBytes))
            {
                callback.allow();
            }
            else {

```

```

        callback.deny();
    }
}

/**
 * Close the session.
 */
public void close() {
    session.close();
}
}

```

.NET

```

using System;
using System.Linq;
using System.Threading;
using PushTechnology.ClientInterface.Client.Details;
using PushTechnology.ClientInterface.Client.Factories;

namespace Examples {
    /// <summary>
    /// This is a control client which registers an authentication
    handler with a server.
    /// </summary>
    public class ControlAuthenticationClient {
        /// <summary>
        /// Main entry point.
        /// </summary>
        public static void Run() {
            var session =
                Diffusion.Sessions.Principal( "auth" ).Password( "auth_secret" )
                    .Open( "ws://diffusion.example.com:80" );

            session.GetAuthenticationControlFeature().SetAuthenticationHandler( "control-
client-auth-handler-example",

                Enum.GetValues( typeof( DetailType ) ).OfType<DetailType>().ToList(),
                    new ExampleControlAuthenticationHandler() );

            while ( true ) {
                Thread.Sleep( 60000 );
            }
        }
    }
}

```

C

```

/*
 * Diffusion can be configured to delegate authentication requests to
 * an external handler. This program provides an authentication
 * handler to demonstrate this feature. A detailed description of
 * security and authentication handlers can be found in the Diffusion
 * user manual.
 *
 * Authentication handlers are registered with a name, which is
 * typically specified in
 * Server.xml
 */

```

```

* Two handler names are provided by default;
* before-system-handler and after-system-handler, and additional
* handlers may be specified for Diffusion through the Server.xml
file
* and an accompanying Java class that implements the
* AuthenticationHandler interface.
*
* This control authentication handler connects to Diffusion and
attempts
* to register itself with a user-supplied name, which should match
the name
* configured in Server.xml.
*
* The default behavior is to install as the "before-system-handler",
* which means that it will intercept authentication requests before
* Diffusion has a chance to act on them.
*
* It will:
* <ul>
* <li>Deny all anonymous connections</li>
* <li>Allow connections where the principal and credentials (i.e.,
username and password) match some hardcoded values</li>
* <li>Abstain from all other decisions, thereby letting Diffusion
and other authentication handlers decide what to do.</li>
* </ul>
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"
#include "conversation.h"

struct user_credentials_s {
    const char *username;
    const char *password;
};

/*
 * Username/password pairs that this handler accepts.
 */
static const struct user_credentials_s USERS[] = {
    { "fish", "chips" },
    { "ham", "eggs" },
    { NULL, NULL }
};

ARG_OPTS_T arg_opts[] = {
    ARG_OPTS_HELP,
    {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
ARG_HAS_VALUE, "ws://localhost:8080"},
    {'n', "name", "Name under which to register the
authentication handler", ARG_OPTIONAL, ARG_HAS_VALUE, "before-
system-handler"},
    {'p', "principal", "Principal (username) for the connection",
ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    {'c', "credentials", "Credentials (password) for the
connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
    END_OF_ARG_OPTS
};

```

```

/*
 * When the authentication service has been registered, this function
 * will be
 * called.
 */
static int
on_registration(SESSION_T *session, void *context)
{
    printf("Registered authentication handler\n");
    return HANDLER_SUCCESS;
}

/*
 * When the authentication service has be deregistered, this function
 * will be
 * called.
 */
static int
on_deregistration(SESSION_T *session, void *context)
{
    printf("Deregistered authentication handler\n");
    return HANDLER_SUCCESS;
}

/*
 * This is the function that is called when authentication has been
 * delegated
 * from Diffusion.
 *
 * The response may return one of three values via the response
 * parameter:
 * ALLOW: The user is authenticated.
 * ALLOW_WITH_RESULT: The user is authenticated, and additional roles
 * are
 * to be applied to the user.
 * DENY: The user is NOT authenticated.
 * ABSTAIN: Allow another handler to make the decision.
 *
 * The handler should return HANDLER_SUCCESS in all cases, unless an
 * actual
 * error occurs during the authentication process (in which case,
 * HANDLER_FAILURE is appropriate).
 */
static int
on_authentication(SESSION_T *session,
                  const SVC_AUTHENTICATION_REQUEST_T *request,
                  SVC_AUTHENTICATION_RESPONSE_T *response,
                  void *context)
{
    // No credentials, or not password type. We're not an
    authority for
    // this type of authentication so abstain in case some other
    registered
    // authentication handler can deal with the request.
    if(request->credentials == NULL) {
        printf("No credentials specified, abstaining\n");
        response->value = AUTHENTICATION_ABSTAIN;
        return HANDLER_SUCCESS;
    }
    if(request->credentials->type != PLAIN_PASSWORD) {
        printf("Credentials are not PLAIN_PASSWORD,
abstaining\n");
        response->value = AUTHENTICATION_ABSTAIN;
    }
}

```

```

        return HANDLER_SUCCESS;
    }

    printf("principal = %s\n", request->principal);
    printf("credentials = %*s\n",
        (int)request->credentials->data->len,
        request->credentials->data->data);

    if(request->principal == NULL || strlen(request->principal)
== 0) {
        printf("Denying anonymous connection (no
principal)\n");
        response->value = AUTHENTICATION_DENY; // Deny anon
connections
        return HANDLER_SUCCESS;
    }

    char *password = malloc(request->credentials->data->len + 1);
    memmove(password, request->credentials->data->data, request-
>credentials->data->len);
    password[request->credentials->data->len] = '\0';

    int auth_decided = 0;
    int i = 0;
    while(USERS[i].username != NULL) {

        printf("Checking username %s vs %s\n", request-
>principal, USERS[i].username);
        printf("        and password %s vs %s\n", password,
USERS[i].password);

        if(strcmp(USERS[i].username, request->principal) == 0
&&
        strcmp(USERS[i].password, password) == 0) {

            puts("Allowed");
            response->value = AUTHENTICATION_ALLOW;
            auth_decided = 1;
            break;
        }
        i++;
    }

    free(password);

    if(auth_decided == 0) {
        puts("Abstained");
        response->value = AUTHENTICATION_ABSTAIN;
    }

    return HANDLER_SUCCESS;
}

int
main(int argc, char** argv)
{
    HASH_T *options = parse_cmdline(argc, argv, arg_opts);
    if (options == NULL || hash_get(options, "help") != NULL) {
        show_usage(argc, argv, arg_opts);
        return EXIT_FAILURE;
    }
}

```

```

char *url = hash_get(options, "url");
char *name = hash_get(options, "name");
char *principal = hash_get(options, "principal");
char *credentials = hash_get(options, "credentials");

/*
 * Create a session with Diffusion.
 */
puts("Creating session");
DIFFUSION_ERROR_T error = { 0 };
SESSION_T *session = session_create(url,
                                   principal,
                                   credentials != NULL ?
credentials_create_password(credentials) : NULL,
                                   NULL, NULL,
                                   &error);

if (session == NULL) {
    fprintf(stderr, "TEST: Failed to create session\n");
    fprintf(stderr, "ERR : %s\n", error.message);
    return EXIT_FAILURE;
}

/*
 * Provide a set (via a hash map containing keys and NULL
 * values) to indicate what information about the connecting
 * client that we'd like Diffusion to send us.
 */
HASH_T *detail_set = hash_new(5);
char buf[2];
sprintf(buf, "%d", SESSION_DETAIL_SUMMARY);
hash_add(detail_set, strdup(buf), NULL);
sprintf(buf, "%d", SESSION_DETAIL_LOCATION);
hash_add(detail_set, strdup(buf), NULL);
sprintf(buf, "%d", SESSION_DETAIL_CONNECTOR_NAME);
hash_add(detail_set, strdup(buf), NULL);

/*
 * Register the authentication handler.
 */
AUTHENTICATION_REGISTRATION_PARAMS_T auth_registration_params
= {
    .name = name,
    .detail_set = detail_set,
    .on_registration = on_registration,
    .authentication_handlers.on_authentication =
on_authentication
};

puts("Sending registration request");
SVC_AUTHENTICATION_REGISTER_REQUEST_T *reg_request =
authentication_register(session,
auth_registration_params);

/*
 * Wait a while before moving on to deregistration.
 */
sleep(30);

AUTHENTICATION_DEREGISTRATION_PARAMS_T
auth_deregistration_params = {
    .on_deregistration = on_deregistration,
    .original_request = reg_request
};

```

```

    /*
     * Deregister the authentication handler.
     */
    printf("Deregistering authentication handler\n");
    authentication_deregister(session,
    auth_deregistration_params);

    session_close(session, NULL);
    session_free(session);

    return EXIT_SUCCESS;
}

```

Change the URL from that provided in the example to the URL of the Diffusion server.

Related concepts

[Configuring authentication handlers](#) on page 561

Authentication handlers and the order that the Diffusion server calls them in are configured in the `Server.xml` configuration file.

Developing a control authentication handler

Implement the `ControlAuthenticationHandler` interface to create a control authentication handler.

About this task

A control authentication handler can be implemented in any language where the Diffusion Unified API includes the `AuthenticationControl` feature.

For more information, see .

This example demonstrates how to implement a control authentication handler in Java.

Note: Where `c.p.d` is used in package names, it indicates `com.pushtechology.diffusion`.

Procedure

1. Edit the `etc/Server.xml` configuration file to include a name that the control authentication handler can register with.

Include the `control-authentication-handler` element in the list of authentication handlers. The order of the list defines the order in which the authentication handlers are called. The value of the `handler-name` attribute is the name that your control authentication handler registers as. For example:

```

<security>
  <authentication-handlers>
    <!-- Include a local authentication handler that can
    authenticate the control client -->
    <authentication-handler class="com.example.LocalHandler" />

    <!-- Register your control authentication handler -->
    <control-authentication-handler handler-name="before-system-
    handler" />

  </authentication-handlers>

```

```
</security>
```

The client that registers your control authentication handler must first authenticate with the Diffusion server. Configure a local authentication handler that allows the client to connect.

2. Start the Diffusion server.
 - On UNIX[®]-based systems, run the `diffusion.sh` command in the `diffusion_installation_dir/bin` directory.
 - On Windows systems, run the `diffusion.bat` command in the `diffusion_installation_dir\bin` directory.
3. Create a Java class that implements `ControlAuthenticationHandler`.

```
package com.example.client;

import com.pushtechology.diffusion.client.details.SessionDetails;
import
    com.pushtechology.diffusion.client.features.control.clients.AuthenticationC
import com.pushtechology.diffusion.client.types.Credentials;

public class ExampleControlAuthenticationHandler implements
    ControlAuthenticationHandler{

    public void authenticate(String principal, Credentials
        credentials,
            SessionDetails sessionDetails, Callback callback) {

        // Logic to make the authentication decision.

        // Authentication decision
        callback.abstain();

        // callback.deny();
        // callback.allow();

    }

    @Override
    public void onActive(RegisteredHandler handler) {

    }

    @Override
    public void onClose() {

    }

}
```

- a) Ensure that you import `Credentials` from the `c.p.d.client.types` package, not the `c.p.d.api` package.
- b) Implement the `authenticate` method.
- c) Use the `allow`, `deny`, or `abstain` method on the `Callback` object to respond with the authentication decision.
- d) You can override the `onActive` and `onClose` to include actions the control authentication handler performs when the client opens its connection to the Diffusion server and when the client closes its session with the Diffusion server.

For example, when the client session becomes active, the control authentication handler uses the `onActive` method to open a connection to a database. When the client session is closed, the control authentication handler uses the `onClose` method to close the connection to the database.

4. Create a simple client that registers your control authentication handler with the Diffusion server.

```
package com.example.client;

import com.example.client.ExampleControlAuthenticationHandler;
import com.pushtechology.diffusion.client.Diffusion;
import
    com.pushtechology.diffusion.client.details.SessionDetails.DetailType;
import
    com.pushtechology.diffusion.client.features.control.clients.AuthenticationControl;
import com.pushtechology.diffusion.client.session.Session;
import com.pushtechology.diffusion.client.session.SessionFactory;

import java.util.EnumSet;

public class ExampleControlClient {

    public static void main(String[] args) {

        final Session session;

        // Create the client session
        SessionFactory sf = Diffusion.sessions();
        session = sf.principal("ControlClient1")
            .passwordCredentials("Passw0rd")
            .open("ws://diffusion.example.com:80");

        // Get the AuthenticationControl feature
        AuthenticationControl authControl =
        session.feature(AuthenticationControl.class);

        // Use the AuthenticationControl feature to register your
        control authentication
        // handler with the name that you configured in Server.xml

        authControl.setAuthenticationHandler("before-system-
        handler",
            EnumSet.allOf(DetailType.class), new
            ExampleControlAuthenticationHandler());
    }
}
```

a) Create a session.

Change the URL from that provided in the example to the URL of the Diffusion server.

b) Use the session to get the `AuthenticationControl` feature.

c) Use the `AuthenticationControl` feature to register your control authentication handler, `ExampleControlAuthenticationHandler`, using the name that you configured in the `etc/Server.xml` configuration file, `before-system-handler`.

5. Start your client.

It connects to the Diffusion server and registers the control authentication handler with the name `before-system-handler`.

Results

When a client authenticates, the Diffusion server forwards the authentication request to the authentication handler you have registered. Your authentication handler can ALLOW, DENY, or ABSTAIN from the authentication decision. If your authentication handler returns an ALLOW or DENY decision, this decision is used as the response to the authenticating client. If your authentication handler returns an ABSTAIN decision, the Diffusion server forwards the authentication request to the next authentication handler. For more information, see [Authentication](#) on page 140.

Related concepts

[User-written authentication handlers](#) on page 143

You can implement authentication handlers that authenticate clients that connect to the Diffusion server or perform an action that requires authentication.

[Authentication](#) on page 140

You can implement and register handlers to authenticate clients when the clients try to perform operations that require authentication.

Related tasks

[Developing a local authentication handler](#) on page 508

Implement the `AuthenticationHandler` interface to create a local authentication handler.

[Developing a composite authentication handler](#) on page 510

Extend the `CompositeAuthenticationHandler` class to combine the decisions from multiple authentication handlers.

[Developing a composite control authentication handler](#) on page 405

Extend the `CompositeControlAuthenticationHandler` class to combine the decisions from multiple control authentication handlers.

Developing a composite control authentication handler

Extend the `CompositeControlAuthenticationHandler` class to combine the decisions from multiple control authentication handlers.

About this task

Using a composite control authentication handler reduces the number of messages that are sent between the Diffusion server and the client to perform authentication.

This example describes how to use a composite control authentication handler as part of a client remote from the Diffusion server.

Procedure

1. Edit the `etc/Server.xml` configuration file to point to your composite control authentication handler.

Include the `control-authentication-handler` element in the list of authentication handlers. The order of the list defines the order in which the authentication handlers are called. The value of the `handler-name` attribute is the name that your composite control authentication handler registers as. For example:

```
<security>
  <authentication-handlers>
    <!-- Include a local authentication handler that can
    authenticate the control client -->
    <authentication-handler class="com.example.LocalHandler" />
```

```

    <!-- Register your composite control authentication handler -->
    <control-authentication-handler handler-name="example-
composite-control-authentication-handler" />

    </authentication-handlers>
</security>

```

The client that registers your control authentication handler must first authenticate with the Diffusion server. Configure a local authentication handler that allows the client to connect.

2. Start the Diffusion server.
 - On UNIX-based systems, run the `diffusion.sh` command in the `diffusion_installation_dir/bin` directory.
 - On Windows systems, run the `diffusion.bat` command in the `diffusion_installation_dir\bin` directory.
3. Create the individual control authentication handlers that your composite control authentication handler calls.

You can follow steps in the task [Developing a control authentication handler](#) on page 402.

In this example, the individual control authentication handlers are referred to as `HandlerOne`, `HandlerTwo`, and `HandlerThree`.

4. Extend the `CompositeControlAuthenticationHandler` class.

```

package com.example.client;

import com.example.client.HandlerOne;
import com.example.client.HandlerTwo;
import com.example.client.HandlerThree;

import
    com.pushtechology.diffusion.client.features.control.clients.CompositeControl

public class ExampleHandler extends
    CompositeControlAuthenticationHandler {

    public ExampleHandler() {
        super(new HandlerOne(), new HandlerTwo(), new
HandlerThree());
    }

}

```

- a) Import your individual control authentication handlers.
 - b) Create a no-argument constructor that calls the super class constructor with a list of your individual handlers.
5. Create a simple client that registers your composite control authentication handler with the Diffusion server.

You can follow steps in the task [Developing a control authentication handler](#) on page 402.

Ensure that you register your composite control authentication handler, `ExampleHandler`, using the name that you configured in the `etc/Server.xml` configuration file, `example-composite-control-authentication-handler`.

6. Start your client.

It connects to the Diffusion server and registers the composite control authentication handler.

Results

When the client session starts, the composite control authentication handler calls the `onActive` methods of the individual control authentication handlers in the order in which they are passed in to the composite handler.

When the composite control authentication handler is called, it calls the individual control authentication handlers that are passed to it as parameters in the order they are passed in.

- If an individual handler responds with `ALLOW`, the composite handler responds with that decision to the Diffusion server and a list of any roles to assign to the authenticated principal.
- If an individual handler responds with `DENY`, the composite handler responds with that decision to the Diffusion server.
- If an individual handler responds with `ABSTAIN`, the composite handler calls the next individual handler in the list.
- If all individual handlers respond with `ABSTAIN`, the composite handler responds to the Diffusion server with an `ABSTAIN` decision.

When the client session closes, the composite control authentication handler calls the `onClose` methods of the individual control authentication handlers in the order in which they are passed in to the composite handler.

Related concepts

[User-written authentication handlers](#) on page 143

You can implement authentication handlers that authenticate clients that connect to the Diffusion server or perform an action that requires authentication.

[Authentication](#) on page 140

You can implement and register handlers to authenticate clients when the clients try to perform operations that require authentication.

Related tasks

[Developing a local authentication handler](#) on page 508

Implement the `AuthenticationHandler` interface to create a local authentication handler.

[Developing a composite authentication handler](#) on page 510

Extend the `CompositeAuthenticationHandler` class to combine the decisions from multiple authentication handlers.

[Developing a control authentication handler](#) on page 402

Implement the `ControlAuthenticationHandler` interface to create a control authentication handler.

Updating the system authentication store

A client can use the `SystemAuthenticationControl` feature to update the system authentication store. The information in the system authentication store is used by the system authentication handler to authenticate users and assign roles to them.

Querying the store

Required permissions: `view_security`

The client can get a snapshot of the current information in the system authentication store. This information is returned as an object model.

The password is passed in as plain text, but is stored in the system authentication store as a secure hash.

Removing a principal

Railroad diagram



Backus-Naur form

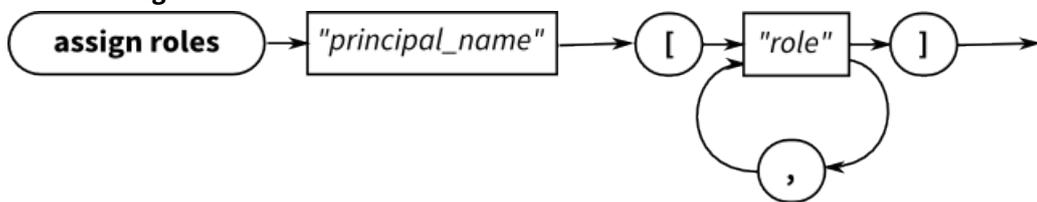
`remove principal "principal_name "`

Example

```
remove principal "user25"
```

Assigning roles to a principal

Railroad diagram



Backus-Naur form

`assign roles "principal_name" '[' "role" [, "role"]]'`

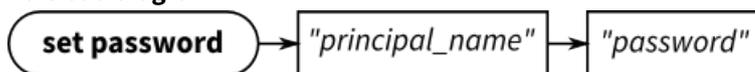
Example

```
assign roles "agent77" ["CLIENT", "CLIENT_CONTROL"]
```

When you use this command to assign roles to a principal, it overwrites any existing roles assigned to that principal. Ensure that all the roles you want the principal to have are listed in the command.

Setting the password for a principal

Railroad diagram



Backus-Naur form

`set password "principal_name" "password "`

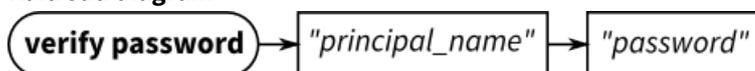
Example

```
set password "user1" "passw0rd"
```

The password is passed in as plain text, but is stored in the system authentication store as a secure hash.

Verifying the password for a principal

Railroad diagram



Backus-Naur form

```
verify password " principal_name " " password "
```

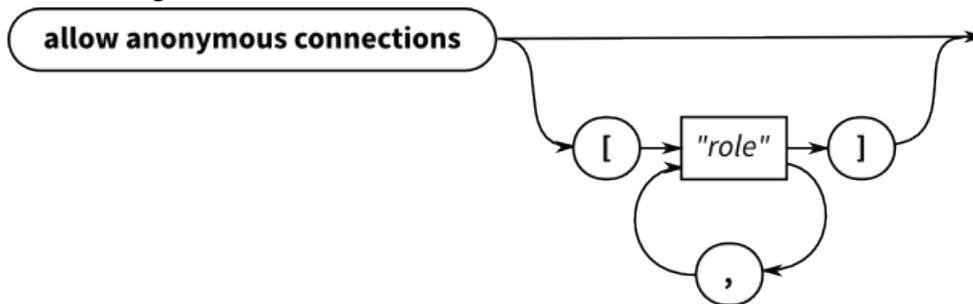
Example

```
verify password "user1" "passw0rd"
```

The password is passed in as plain text, but is stored in the system authentication store as a secure hash.

Allowing anonymous connections

Railroad diagram



Backus-Naur form

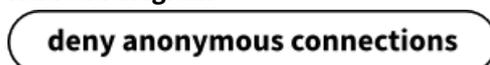
```
allow anonymous connections ['["role" [ , "role" ] ]']
```

Example

```
allow anonymous connections [ "CLIENT" ]
```

Denying anonymous connections

Railroad diagram



Backus-Naur form

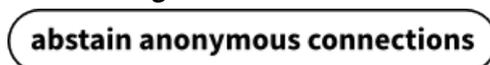
```
deny anonymous connections
```

Example

```
deny anonymous connections
```

Abstaining from providing a decision about anonymous connections

Railroad diagram



Backus-Naur form

```
abstain anonymous connections
```

Example

```
abstain anonymous connections
```

Example: Update the system authentication store

The following examples use the SystemAuthenticationControl feature in the Unified API to update the system authentication store.

JavaScript

Note: Only steps 4 and 5 deal with the system authentication store.

```
// Session security allows you to change the principal that a session
// is authenticated as. It also allows users to
// query and update server-side security and authentication stores,
// which control users, roles and permissions.
// This enables you to manage the capabilities that any logged in
// user will have access to.

// Connect to Diffusion with control client credentials
diffusion.connect({
  host    : 'diffusion.example.com',
  port    : 443,
  secure  : true,
  principal : 'control',
  credentials : 'password'
}).then(function(session) {

  // 1. A session change their principal by re-authenticating
  session.security.changePrincipal('admin',
  'password').then(function() {
    console.log('Authenticated as admin');
  });

  // 2. The security configuration provides details about roles and
  // their assigned permissions
  session.security.getSecurityConfiguration().then(function(config)
  {
    console.log('Roles for anonymous sessions: ',
    config.anonymous);
    console.log('Roles for named sessions: ', config.named);
    console.log('Available roles: ', config.roles);
  }, function(error) {
    console.log('Unable to fetch security configuration', error);
  });

  // 3. Changes to the security configuration are done with a
  // SecurityScriptBuilder
  var securityScriptBuilder =
  session.security.securityScriptBuilder();

  // Set the permissions for a particular role - global and topic-
  // scoped
  // Each method on a script builder returns a new builder
  var setPermissionScript =
  securityScriptBuilder.setGlobalPermissions('SUPERUSER',
  ['REGISTER_HANDLER'])

  .setTopicPermissions('SUPERUSER', '/foo', ['UPDATE_TOPIC'])
  .build();

  // Update the server-side store with the generated script
```

```

session.security.updateSecurityStore(setPermissionScript).then(function()
{
    console.log('Security configuration updated successfully');
}, function(error) {
    console.log('Failed to update security configuration: ',
error);
});

// 4. The system authentication configuration lists all users &
roles

session.security.getSystemAuthenticationConfiguration().then(function(config)
{
    console.log('System principals: ', config.principals);
    console.log('Anonymous sessions: ', config.anonymous);
}, function(error) {
    console.log('Unable to fetch system authentication
configuration', error);
});

// 5. Changes to the system authentication config are done with a
SystemAuthenticationScriptBuilder
var authenticationScriptBuilder =
session.security.authenticationScriptBuilder();

// Add a new user and set password & roles.
var addUserScript =
authenticationScriptBuilder.addPrincipal('Superman',
'correcthorsebatterystapler')

.assignRoles('Superman', ['SUPERUSER'])

.build();

// Update the system authentication store

session.security.updateAuthenticationStore(addUserScript).then(function()
{
    console.log('Updated system authentication config');
}, function(error) {
    console.log('Failed to update system authentication: ',
error);
});
});

```

Java and Android

```

package com.pushtechnology.diffusion.examples;

import java.util.HashSet;
import java.util.Set;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.pushtechnology.diffusion.client.Diffusion;
import com.pushtechnology.diffusion.client.callbacks.ErrorReason;
import
    com.pushtechnology.diffusion.client.features.control.clients.SystemAuthenticati
import
    com.pushtechnology.diffusion.client.features.control.clients.SystemAuthenticati

```

```

import
    com.pushtechology.diffusion.client.features.control.clients.SystemAuthenticati
import
    com.pushtechology.diffusion.client.features.control.clients.SystemAuthenticati
import
    com.pushtechology.diffusion.client.features.control.clients.SystemAuthenticati
import
    com.pushtechology.diffusion.client.features.control.clients.SecurityStoreFeatu
import com.pushtechology.diffusion.client.session.Session;

/**
 * An example of using a control client to alter the system
 authentication
 configuration.
 * <P>
 * This uses the {@link SystemAuthenticationControl} feature only.
 *
 * @author Push Technology Limited
 * @since 5.2
 */
public class ControlClientChangingSystemAuthentication {

    private static final Logger LOG =
        LoggerFactory.getLogger(
            ControlClientChangingSystemAuthentication.class);

    private final SystemAuthenticationControl
systemAuthenticationControl;

    /**
     * Constructor.
     */
    public ControlClientChangingSystemAuthentication() {

        final Session session = Diffusion.sessions()
            // Authenticate with a user that has the VIEW_SECURITY
and
            // MODIFY_SECURITY permissions.
            .principal("admin").password("password")
            // Use a secure channel because we're transferring
sensitive
            // information.
            .open("wss://diffusion.example.com:80");

        systemAuthenticationControl =
            session.feature(SystemAuthenticationControl.class);
    }

    /**
     * For all system users, update the assigned roles to replace the
     * "SUPERUSER" role and with "ADMINISTRATOR".
     *
     * @param callback result callback
     */
    public void changeSuperUsersToAdministrators(UpdateStoreCallback
callback) {

        systemAuthenticationControl.getSystemAuthentication(
            new ChangeSuperUsersToAdministrators(callback));
    }

    private final class ChangeSuperUsersToAdministrators
        implements ConfigurationCallback {

```

```

        private final UpdateStoreCallback callback;

        ChangeSuperUsersToAdministrators(UpdateStoreCallback
callback) {
            this.callback = callback;
        }

        @Override
        public void onReply(SystemAuthenticationConfiguration
configuration) {

            ScriptBuilder builder =
                systemAuthenticationControl.scriptBuilder();

            // For all system users ...
            for (SystemPrincipal principal :
configuration.getPrincipals()) {

                final Set<String> assignedRoles =
principal.getAssignedRoles();

                // ... that have the SUPERUSER assigned role ...
                if (assignedRoles.contains("SUPERUSER")) {
                    final Set<String> newRoles = new
HashSet<>(assignedRoles);
                    newRoles.remove("SUPERUSER");
                    newRoles.add("ADMINISTRATOR");

                    // ... add a command to the script that updates
the user's
                    // assigned roles, replacing SUPERUSER with
"ADMINISTRATOR".
                    builder =
newRoles);
                    builder.assignRoles(principal.getName(),
                }
            }

            final String script = builder.script();

            LOG.info(
                "Sending the following script to the server:\n{}",
                script);

            systemAuthenticationControl.updateStore(
                script,
                callback);
        }

        @Override
        public void onError(ErrorReason errorReason) {
            // This might fail if the session lacks the required
permissions.
            callback.onError(errorReason);
        }
    }

    /**
     * Close the session.
     */
    public void close() {
        systemAuthenticationControl.getSession().close();
    }
}

```

```
}  
}
```

.NET

```
using System.Collections.Generic;  
using System.Linq;  
using PushTechnology.ClientInterface.Client.Callbacks;  
using PushTechnology.ClientInterface.Client.Factories;  
using PushTechnology.ClientInterface.Client.Features.Control.Clients;  
using PushTechnology.ClientInterface.Client.Types;  
using IUpdateStoreCallback =  
    PushTechnology.ClientInterface.Client.Features.Control.Clients.SecurityControl.  
  
namespace Examples {  
    public class ControlClientChangingSystemAuthentication {  
        private readonly ISystemAuthenticationControl  
        theSystemAuthenticationControl;  
  
        public ControlClientChangingSystemAuthentication() {  
            // Authenticate with a user that has the VIEW_SECURITY  
            and MODIFY_SECURITY permissions  
            var session =  
                Diffusion.Sessions.Principal( "control" ).Password( "password" )  
            // Use a secure channel because we're transferring  
            sensitive information.  
            .Open( "wss://localhost:8443" );  
  
            theSystemAuthenticationControl =  
                session.GetSystemAuthenticationControlFeature();  
        }  
  
        /// <summary>  
        /// For all system users, update the assigned roles to  
        replace the 'SUPERUSER' role and with 'ADMINISTRATOR'.  
        /// </summary>  
        /// <param name="callback"></param>  
        public void  
        ChangeSuperUsersToAdministrators( IUpdateStoreCallback callback ) {  
            theSystemAuthenticationControl.GetSystemAuthentication(  
                new  
                ChangeSuperusersToAdministrators( theSystemAuthenticationControl,  
                callback ) );  
        }  
  
        private class InternalUpdateStoreCallback :  
        IUpdateStoreCallback {  
            /// <summary>  
            /// The script was applied successfully.  
            /// </summary>  
            public void OnSuccess() {  
            }  
  
            /// <summary>  
            /// The script was rejected. No changes were made to the  
            system authentication store.  
            /// </summary>  
            /// <param name="errors">The details of why the script  
            was rejected.</param>  
            public void OnRejected( ICollection<IErrorReport>  
            errors ) {  
            }  
        }  
    }  
}
```

```

        /// <summary>
        /// Notification of a contextual error related to this
callback. This is analogous to an exception being
        /// raised. Situations in which <code>OnError</code> is
called include the session being closed, a
        /// communication timeout, or a problem with the provided
parameters. No further calls will be made to this
        /// callback.
        /// </summary>
        /// <param name="errorReason">errorReason a value
representing the error; this can be one of constants
        /// defined in <see cref="ErrorReason" />, or a feature-
specific reason.</param>
        public void OnError( ErrorReason errorReason ) {
        }

        private class ChangeSuperusersToAdministrators :
IConfigurationCallback {
            private readonly ISystemAuthenticationControl
theSystemAuthenticationControl;
            private readonly IUpdateStoreCallback theCallback;

            public
ChangeSuperusersToAdministrators( ISystemAuthenticationControl
systemAuthenticationControl,
                IUpdateStoreCallback callback ) {
                theSystemAuthenticationControl =
systemAuthenticationControl;
                theCallback = callback;
            }

            /// <summary>
            /// The configuration callback reply.
            /// </summary>
            /// <param name="systemAuthenticationConfiguration">The
system authentication configuration stored on the
            /// server.</param>
            public void OnReply( ISystemAuthenticationConfiguration
systemAuthenticationConfiguration ) {
                var builder =
theSystemAuthenticationControl.ScriptBuilder();

                // For all system users...
                foreach ( var principal in
systemAuthenticationConfiguration.Principals ) {
                    var assignedRoles = principal.AssignedRoles;

                    // ...that have the 'SUPERUSER' assigned role...
                    if ( !assignedRoles.Contains( "SUPERUSER" ) )
                        continue;

                    var newRoles = new
HashSet<string>( assignedRoles );

                    newRoles.Remove( "SUPERUSER" );
                    newRoles.Add( "ADMINISTRATOR" );

                    // ...and add a command to the script that
updates the user's assigned roles, replacing 'SUPERUSER'
                    // with 'ADMINISTRATOR'.

```



```

/*
 * This callback is invoked when the system authentication store is
 * received, and prints the contents of the store.
 */
int
on_get_system_authentication_store(SESSION_T *session,
                                   const
                                   SYSTEM_AUTHENTICATION_STORE_T store,
                                   void *context)
{
    puts("on_get_system_authentication_store()");

    printf("Got %ld principals\n", store.system_principals-
>size);

    char **names = get_principal_names(store);
    for(char **name = names; *name != NULL; name++) {
        printf("Principal: %s\n", *name);

        char **roles = get_roles_for_principal(store, *name);
        for(char **role = roles; *role != NULL; role++) {
            printf("  |- Role: %s\n", *role);
        }
        free(roles);
    }
    free(names);

    switch(store.anonymous_connection_action) {
    case ANONYMOUS_CONNECTION_ACTION_ALLOW:
        puts("Allow anonymous connections");
        break;
    case ANONYMOUS_CONNECTION_ACTION_DENY:
        puts("Deny anonymous connections");
        break;
    case ANONYMOUS_CONNECTION_ACTION_ABSTAIN:
        puts("Abstain from making anonymous connection
decision");
        break;
    }

    puts("Anonymous connection roles:");
    char **roles = get_anonymous_roles(store);
    for(char **role = roles; *role != NULL; role++) {
        printf("  |- Role: %s\n", *role);
    }
    free(roles);

    apr_thread_mutex_lock(mutex);
    apr_thread_cond_broadcast(cond);
    apr_thread_mutex_unlock(mutex);

    return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
    /*
     * Standard command-line parsing.
     */
    const HASH_T *options = parse_cmdline(argc, argv, arg_opts);
    if(options == NULL || hash_get(options, "help") != NULL) {
        show_usage(argc, argv, arg_opts);
    }
}

```

```

        return EXIT_FAILURE;
    }

    const char *url = hash_get(options, "url");
    const char *principal = hash_get(options, "principal");
    CREDENTIALS_T *credentials = NULL;
    const char *password = hash_get(options, "credentials");
    if(password != NULL) {
        credentials = credentials_create_password(password);
    }

    /*
     * Setup for condition variable
     */
    apr_initialize();
    apr_pool_create(&pool, NULL);
    apr_thread_mutex_create(&mutex, APR_THREAD_MUTEX_UNNESTED,
pool);
    apr_thread_cond_create(&cond, pool);

    /*
     * Create a session with Diffusion.
     */
    SESSION_T *session;
    DIFFUSION_ERROR_T error = { 0 };
    session = session_create(url, principal, credentials, NULL,
NULL, &error);
    if(session == NULL) {
        fprintf(stderr, "TEST: Failed to create session\n");
        fprintf(stderr, "ERR : %s\n", error.message);
        return EXIT_FAILURE;
    }

    /*
     * Request the system authentication store.
     */
    const GET_SYSTEM_AUTHENTICATION_STORE_PARAMS_T params = {
        .on_get = on_get_system_authentication_store
    };

    apr_thread_mutex_lock(mutex);

    get_system_authentication_store(session, params);

    apr_thread_cond_wait(cond, mutex);
    apr_thread_mutex_unlock(mutex);

    /*
     * Close the session and tidy up.
     */
    session_close(session, NULL);
    session_free(session);

    apr_thread_mutex_destroy(mutex);
    apr_thread_cond_destroy(cond);
    apr_pool_destroy(pool);
    apr_terminate();

    return EXIT_SUCCESS;
}

```

Change the URL from that provided in the example to the URL of the Diffusion server.

Updating the security store

A client can use the SecurityControl feature to update the security store. The information in the security store is used by the Diffusion server to define the permissions assigned to roles and the roles assigned to anonymous sessions and named sessions.

Querying the store

Required permissions: view_security

The client can get a snapshot of the current information in the security store. This information is returned as an object model.

Updating the store

Required permissions: modify_security

The client can use a command script to update the security store. The command script is a string that contains a command on each line. These commands are applied to the current state of the security store.

The update is transactional. Unless all of the commands in the script can be applied, none of them are.

Using a script builder

You can use a script builder to create the command script used to update the security store. Use the script builder to create commands for the following actions:

- Set the global permissions assigned to a named role
- Set the default topic permissions assigned to a named role
- Set the topic permissions associated with a specific topic path assigned to a named role
 - This can include explicitly setting a role to have no permissions at a topic path.
- Remove the topic permissions associated with a specific topic path assigned to a named role
- Set the roles included in a named role
- Set the roles assigned to sessions authenticated with a named principal
- Set the roles assigned to anonymous sessions

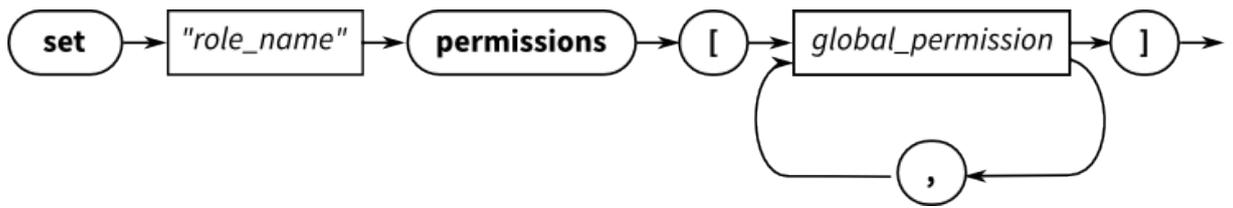
DSL syntax: security store

The scripts that you can use with the SecurityControl feature to update the security store are formatted according to a domain-specific language (DSL). You can use the script builders provided in the APIs to create a script to update the security store. However, if you want to create the script by some other method, ensure that it conforms to the DSL.

The following sections each describe the syntax for a single line of the script file.

Assigning global permissions to a role

Railroad diagram



Backus-Naur form

`set "role_name" permissions ['[' global_permission [, global_permission] ']]`

Example

```
set "ADMINISTRATOR" permissions [CONTROL_SERVER, VIEW_SERVER,
VIEW_SECURITY, MODIFY_SECURITY]
set "CLIENT_CONTROL" permissions [VIEW_SESSION, MODIFY_SESSION,
REGISTER_HANDLER]
```

Assigning default topic permissions to a role

Railroad diagram



Backus-Naur form

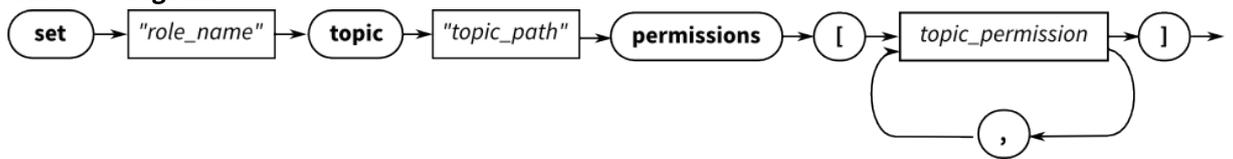
`set "role_name" default topic permissions ['[' topic_permission [, topic_permission] ']]`

Example

```
set "CLIENT" default topic permissions [READ_TOPIC ,
SEND_TO_MESSAGE_HANDLER]
```

Assigning topic permissions associated with a specific topic path to a role

Railroad diagram



Backus-Naur form

`set "role_name" topic "topic_path" permissions ['[' topic_permission [, topic_permission] ']]`

Example

```
set "CLIENT" topic "foo/bar" permissions [READ_TOPIC,
SEND_TO_MESSAGE_HANDLER]
set "ADMINISTRATOR" topic "foo" permissions [ MODIFY_TOPIC ]
set "CLIENT_CONTROL" topic "foo" permissions [ ]
```

Removing all topic permissions associated with a specific topic path to a role

Railroad diagram



Backus-Naur form

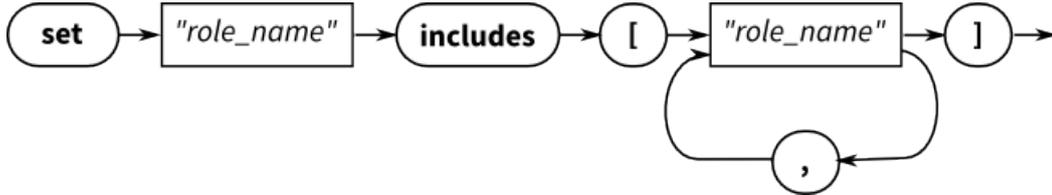
remove "role_name" permissions for topic "topic_path"

Example

```
remove "CLIENT" permissions for topic "foo/bar"
```

Including roles within another role

Railroad diagram



Backus-Naur form

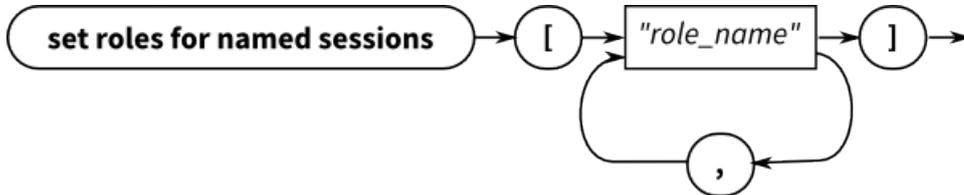
set "role_name" includes ['["role_name" [, "role_name"]']']

Example

```
set "ADMINISTRATOR" includes ["CLIENT_CONTROL" , "TOPIC_CONTROL"]
set "CLIENT_CONTROL" includes ["CLIENT"]
```

Assigning roles to a named session

Railroad diagram



Backus-Naur form

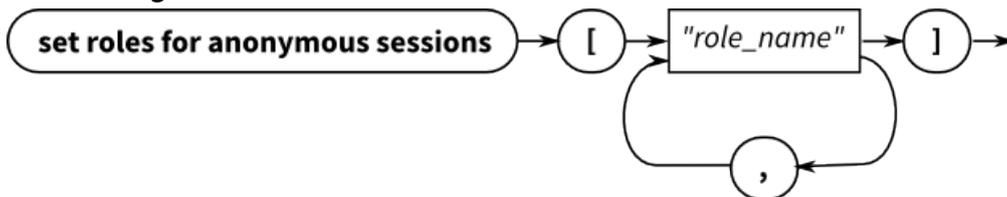
set roles for named sessions ['["role_name" [, "role_name"]']']

Example

```
set roles for named sessions ["CLIENT"]
```

Assigning roles to an anonymous session

Railroad diagram



Backus-Naur form

set roles for anonymous sessions ['["role_name" [, "role_name"]']']

Example

```
set roles for anonymous sessions ["CLIENT"]
```

Example: Update the security store

The following examples use the SecurityControl feature in the Unified API to update the security store.

JavaScript

Note: Only steps 2 and 3 deal with the security store.

```
// Session security allows you to change the principal that a session
// is authenticated as. It also allows users to
// query and update server-side security and authentication stores,
// which control users, roles and permissions.
// This enables you to manage the capabilities that any logged in
// user will have access to.

// Connect to Diffusion with control client credentials
diffusion.connect({
  host    : 'diffusion.example.com',
  port    : 443,
  secure  : true,
  principal : 'control',
  credentials : 'password'
}).then(function(session) {

  // 1. A session change their principal by re-authenticating
  session.security.changePrincipal('admin',
  'password').then(function() {
    console.log('Authenticated as admin');
  });

  // 2. The security configuration provides details about roles and
  // their assigned permissions
  session.security.getSecurityConfiguration().then(function(config)
  {
    console.log('Roles for anonymous sessions: ',
    config.anonymous);
    console.log('Roles for named sessions: ', config.named);
    console.log('Available roles: ', config.roles);
  }, function(error) {
    console.log('Unable to fetch security configuration', error);
  });

  // 3. Changes to the security configuration are done with a
  // SecurityScriptBuilder
  var securityScriptBuilder =
  session.security.securityScriptBuilder();

  // Set the permissions for a particular role - global and topic-
  // scoped
  // Each method on a script builder returns a new builder
  var setPermissionScript =
  securityScriptBuilder.setGlobalPermissions('SUPERUSER',
  ['REGISTER_HANDLER'])

  .setTopicPermissions('SUPERUSER', '/foo', ['UPDATE_TOPIC'])
  .build();
```

```

    // Update the server-side store with the generated script
    session.security.updateSecurityStore(setPermissionScript).then(function()
    {
        console.log('Security configuration updated successfully');
    }, function(error) {
        console.log('Failed to update security configuration: ',
        error);
    });

    // 4. The system authentication configuration lists all users &
    roles

    session.security.getSystemAuthenticationConfiguration().then(function(config)
    {
        console.log('System principals: ', config.principals);
        console.log('Anonymous sessions: ', config.anonymous);
    }, function(error) {
        console.log('Unable to fetch system authentication
        configuration', error);
    });

    // 5. Changes to the system authentication config are done with a
    SystemAuthenticationScriptBuilder
    var authenticationScriptBuilder =
    session.security.authenticationScriptBuilder();

    // Add a new user and set password & roles.
    var addUserScript =
    authenticationScriptBuilder.addPrincipal('Superman',
    'correcthorsebatterystapler')

    .assignRoles('Superman', ['SUPERUSER'])
    .build();

    // Update the system authentication store

    session.security.updateAuthenticationStore(addUserScript).then(function()
    {
        console.log('Updated system authentication config');
    }, function(error) {
        console.log('Failed to update system authentication: ',
        error);
    });
});

```

Java and Android

```

package com.pushtechology.diffusion.examples;

import java.util.Collections;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.pushtechology.diffusion.client.Diffusion;
import com.pushtechology.diffusion.client.callbacks.ErrorReason;

```

```

import
    com.pushtechnology.diffusion.client.features.control.clients.SecurityControl;
import
    com.pushtechnology.diffusion.client.features.control.clients.SecurityControl.Co
import
    com.pushtechnology.diffusion.client.features.control.clients.SecurityControl.Ro
import
    com.pushtechnology.diffusion.client.features.control.clients.SecurityControl.Sc
import
    com.pushtechnology.diffusion.client.features.control.clients.SecurityControl.Se
import
    com.pushtechnology.diffusion.client.features.control.clients.SecurityStoreFeatu
import com.pushtechnology.diffusion.client.session.Session;
import com.pushtechnology.diffusion.client.types.GlobalPermission;
import com.pushtechnology.diffusion.client.types.TopicPermission;

/**
 * An example of using a control client to alter the security
 * configuration.
 * <P>
 * This uses the {@link SecurityControl} feature only.
 *
 * @author Push Technology Limited
 * @since 5.3
 */
public class ControlClientChangingSecurity {

    private static final Logger LOG =
        LoggerFactory.getLogger(
            ControlClientChangingSecurity.class);

    private final SecurityControl securityControl;

    /**
     * Constructor.
     */
    public ControlClientChangingSecurity() {

        final Session session = Diffusion.sessions()
            // Authenticate with a user that has the VIEW_SECURITY
and
            // MODIFY_SECURITY permissions.
            .principal("admin").password("password")
            // Use a secure channel because we're transferring
sensitive
            // information.
            .open("wss://diffusion.example.com:80");

        securityControl = session.feature(SecurityControl.class);
    }

    /**
     * This will update the security store to ensure that all roles
     start with a
     * capital letter (note that this does not address changing the
     use of the
     * roles in the system authentication store).
     *
     * @param callback result callback
     */
    public void capitalizeRoles(UpdateStoreCallback callback) {
        securityControl.getSecurity(new CapitalizeRoles(callback));
    }
}

```

```

private final class CapitalizeRoles implements
ConfigurationCallback {

    private final UpdateStoreCallback callback;

    CapitalizeRoles(UpdateStoreCallback callback) {
        this.callback = callback;
    }

    @Override
    public void onReply(SecurityConfiguration configuration) {

        ScriptBuilder builder =
            securityControl.scriptBuilder();

        builder = builder.setRolesForAnonymousSessions(
capitalize(configuration.getRolesForAnonymousSessions()));

        builder = builder.setRolesForNamedSessions(
capitalize(configuration.getRolesForNamedSessions()));

        for (Role role : configuration.getRoles()) {

            final String oldName = role.getName();
            final String newName = capitalize(oldName);

            // Only if new name is different
            if (!oldName.equals(newName)) {

                // Global Permissions
                final Set<GlobalPermission> globalPermissions =
                    role.getGlobalPermissions();
                if (!globalPermissions.isEmpty()) {
                    // Remove global permissions for old role
                    builder =
                        builder.setGlobalPermissions(
                            oldName,

Collections.<GlobalPermission>emptySet());
                    // Set global permissions for new role
                    builder =
                        builder.setGlobalPermissions(
                            newName,
                            role.getGlobalPermissions());
                }

                final Set<TopicPermission>
defaultTopicPermissions =
                    role.getDefaultTopicPermissions();
                if (!defaultTopicPermissions.isEmpty()) {
                    // Remove default topic permissions for old
role
                    builder =
                        builder.setDefaultTopicPermissions(
                            oldName,

Collections.<TopicPermission>emptySet());
                    // Set default topic permissions for new role
                    builder =
                        builder.setDefaultTopicPermissions(

```

```

        newName,
        role.getDefaultTopicPermissions());
    }

    final Map<String, Set<TopicPermission>>
topicPermissions =
        role.getTopicPermissions();

    if (!topicPermissions.isEmpty()) {
        for (Map.Entry<String, Set<TopicPermission>>
entry : topicPermissions
            .entrySet()) {
            final String topicPath = entry.getKey();
            // Remove old topic permissions
            builder =
                builder.removeTopicPermissions(
                    oldName,
                    topicPath);
            // Set new topic permissions
            builder =
                builder.setTopicPermissions(
                    newName,
                    topicPath,
                    entry.getValue());
        }
    }

    final Set<String> oldIncludedRoles =
role.getIncludedRoles();
    if (!oldIncludedRoles.isEmpty()) {

        if (!oldName.equals(newName)) {
            // Remove old included roles
            builder =
                builder.setRoleIncludes(
                    oldName,
                    Collections.<String>emptySet());
        }

        // This is done even if role name did not change
as it is

        // possible that roles included may have
        final Set<String> newIncludedRoles =
            capitalize(oldIncludedRoles);
        builder =
            builder.setRoleIncludes(
                newName,
                newIncludedRoles);
    }

}

final String script = builder.script();

LOG.info(
    "Sending the following script to the server:\n{}",
    script);

securityControl.updateStore(

```

```

        script,
        callback);
    }

    private Set<String> capitalize(Set<String> roles) {
        final Set<String> newSet = new TreeSet<>();
        for (String role : roles) {
            newSet.add(capitalize(role));
        }
        return newSet;
    }

    private String capitalize(String role) {
        return Character.toUpperCase(role.charAt(0)) +
        role.substring(1);
    }

    @Override
    public void onError(ErrorReason errorReason) {
        // This might fail if the session lacks the required
        permissions.
        callback.onError(errorReason);
    }
}

/**
 * Close the session.
 */
public void close() {
    securityControl.getSession().close();
}
}

```

.NET

```

using System.Collections.Generic;
using System.Linq;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Factories;
using
    PushTechnology.ClientInterface.Client.Features.Control.Clients.SecurityControl;
using PushTechnology.ClientInterface.Client.Types;

namespace Examples {
    /// <summary>
    /// An example of using a control client to alter the security
    configuration.
    ///
    /// This uses the <see cref="ISecurityControl"/> feature only.
    /// </summary>
    public class ControlClientChangingSecurity {
        private readonly ISecurityControl securityControl;

        public ControlClientChangingSecurity() {
            // Authenticate with a user that has the VIEW_SECURITY
            and MODIFY_SECURITY permissions.
            var session =
            Diffusion.Sessions.Principal( "admin" ).Password( "password" )
            // Use a secure channel because we're transferring
            sensitive information.
            .Open( "wss://diffusion.example.com:8443" );
        }
    }
}

```

```

        securityControl = session.GetSecurityControlFeature();
    }

    public void DoCapitalizeRoles( IUpdateStoreCallback
callback ) {
        securityControl.GetSecurity( new
CapitalizeRoles( securityControl, callback ) );
    }

    private class CapitalizeRoles : IConfigurationCallback {
        private readonly ISecurityControl theSecurityControl;
        private readonly IUpdateStoreCallback theCallback;

        /// <summary>
        /// Constructor.
        /// </summary>
        /// <param name="securityControl">The security control
object.</param>
        /// <param name="callback">The callback object.</param>
        public CapitalizeRoles( ISecurityControl securityControl,
IUpdateStoreCallback callback ) {
            theSecurityControl = securityControl;
            theCallback = callback;
        }

        /// <summary>
        /// Notification of a contextual error related to this
callback. This is analogous to an exception being
        /// raised. Situations in which <code>OnError</code> is
called include the session being closed, a
        /// communication timeout, or a problem with the provided
parameters. No further calls will be made to this
        /// callback.
        /// </summary>
        /// <param name="errorReason">errorReason a value
representing the error; this can be one of constants
        /// defined in <see cref="ErrorReason" />, or a feature-
specific reason.</param>
        public void OnError( ErrorReason errorReason ) {
            // This might fail if the session lacks the required
permissions.
            theCallback.OnError( errorReason );
        }

        /// <summary>
        /// This is called to return the requested security
configuration.
        /// </summary>
        /// <param name="configuration">The snapshot of
information from the security store.</param>
        public void OnReply( ISecurityConfiguration
configuration ) {
            var builder = theSecurityControl.ScriptBuilder();

            builder = builder.SetRolesForAnonymousSessions(
Capitalize( configuration.RolesForAnonymousSessions ) );

            builder = builder.SetRolesForNamedSessions(
Capitalize( configuration.RolesForNamedSessions ) );

            foreach ( var role in configuration.Roles ) {

```

```

        var oldName = role.Name;
        var newName = Capitalize( oldName );

        // Only if new name is different
        if ( !oldName.Equals( newName ) ) {
            // Global permissions
            var globalPermissions =
role.GlobalPermissions;

            if ( globalPermissions.Count > 0 ) {
                // Remove global permissions for old role
                builder =
builder.SetGlobalPermissions( oldName, new
List<GlobalPermission>( ) );

                // Set global permissions for new role
                builder =
builder.SetGlobalPermissions( newName,
new
List<GlobalPermission>( role.GlobalPermissions ) );
            }

            var defaultTopicPermissions =
role.DefaultTopicPermissions;

            if ( defaultTopicPermissions.Count > 0 ) {
                // Remove default topic permissions for
old role
                builder =
builder.SetDefaultTopicPermissions( oldName, new
List<TopicPermission>( ) );

                // Set default topic permissions for new
role
                builder =
builder.SetDefaultTopicPermissions( newName,
new
List<TopicPermission>( role.DefaultTopicPermissions ) );
            }

            var topicPermissions = role.TopicPermissions;

            if ( topicPermissions.Count > 0 ) {
                foreach ( var entry in topicPermissions )
{
                    var topicPath = entry.Key;

                    // Remove old topic permissions
                    builder =
builder.RemoveTopicPermissions( oldName, topicPath );

                    // Set new topic permissions
                    builder =
builder.SetTopicPermissions( newName, topicPath, entry.Value );
                }
            }
        }

        var oldIncludedRoles = role.IncludedRoles;

        if ( oldIncludedRoles.Count > 0 ) {
            // Remove old included roles

```



```

        // The action to take on a client session open
notification
    })
    .on('onSessionUpdate', function(event) {
        // The action to take on a client session update
notification
    })
    .on('onSessionClose', function(event) {
        // The action to take on a client session close
notification
    });
}, function(err) {
    console.log('An error has occurred:', err);
});

```

Java and Android

```

ClientControl clientControl = session.feature(ClientControl.class);

clientControl.setSessionPropertiesListener(
    new ClientControl.SessionPropertiesListener.Default() {
        @Override
        public void onSessionOpen(){
            // The action to take on a client session open
notification
        }
        @Override
        public void onSessionEvent(){
            // The action to take on a client session update
notification
        }
        @Override
        public void onSessionClose(){
            // The action to take on a client session close
notification
        }
    },
    // The session properties to receive
    "$Country", "$Department");

```

.NET

```

var _clientControl = session.GetClientControlFeature();

// Set up a listener to receive notification of all sessions
_clientControl.SetSessionPropertiesListener( propertyListener,
"$Country", "Department" );

```

C

```

/*
 * Register a session properties listener.
 *
 * Requests all "fixed" properties, i.e. those defined by
 * Diffusion rather than user-defined properties.
 */
SET_T *required_properties = set_new_string(5);
set_add(required_properties,
    PROPERTIES_SELECTOR_ALL_FIXED_PROPERTIES);

// Set the parameters to callbacks previously defined
SESSION_PROPERTIES_REGISTRATION_PARAMS_T params = {

```

```

        .on_registered = on_registered,
        .on_registration_error = on_registration_error,
        .on_session_open = on_session_open,
        .on_session_close = on_session_close,
        .on_session_update = on_session_update,
        .on_session_error = on_session_error,
        .required_properties = required_properties
    };
    session_properties_listener_register(session, params);

```

When registering this listener, specify which session properties to receive for each client session:

JavaScript

```

// Receive all fixed properties
session.clients.setSessionPropertiesListener(diffusion.clients.PropertyKeys.ALL_
    listener)
    .then(function() {

        });
// OR
// Receive all user-defined properties
session.clients.setSessionPropertiesListener(diffusion.clients.PropertyKeys.ALL_
    listener)
    .then(function() {

        });
// OR
// Receive all properties
session.clients.setSessionPropertiesListener([diffusion.clients.PropertyKeys.ALL_
    diffusion.clients.PropertyKeys.ALL_USER_PROPERTIES], listener)
    .then(function() {

        });

```

Java and Android

```

// Define individual session properties to receive
clientControl.setSessionPropertiesListener(
    new ClientControl.SessionPropertiesListener.Default() {
        // Define callbacks
    },
    "$Country", "$Department");
// OR
// Receive all fixed properties
clientControl.setSessionPropertiesListener(
    new ClientControl.SessionPropertiesListener.Default() {
        // Define callbacks
    },
    Session.ALL_FIXED_PROPERTIES);
// OR
// Receive all user-defined properties
clientControl.setSessionPropertiesListener(
    new ClientControl.SessionPropertiesListener.Default() {
        // Define callbacks
    },
    Session.ALL_USER_PROPERTIES);

```

.NET

```

// Define individual session properties to receive

```

```

_clientControl.SetSessionPropertiesListener(propertiesListener,
"$Country", "Department" );
// OR
// Receive all fixed properties
_clientControl.SetSessionPropertiesListener(propertiesListener,
SessionControlConstants.AllFixedProperties );
// OR
// Receive all user-defined properties
_clientControl.SetSessionPropertiesListener(propertiesListener,
SessionControlConstants.AllUserProperties );

```

C

```

// Receive all fixed properties
SET_T *required_properties = set_new_string(5);
set_add(required_properties,
PROPERTIES_SELECTOR_ALL_FIXED_PROPERTIES);

SESSION_PROPERTIES_REGISTRATION_PARAMS_T params = {
    //Other parameters
    .required_properties = required_properties
};
// OR
// Receive all user-defined properties
SET_T *required_properties = set_new_string(5);
set_add(required_properties,
PROPERTIES_SELECTOR_ALL_USER_PROPERTIES);

SESSION_PROPERTIES_REGISTRATION_PARAMS_T params = {
    //Other parameters
    .required_properties = required_properties
};
// OR
// Receive all properties
SET_T *required_properties = set_new_string(5);
set_add(required_properties,
PROPERTIES_SELECTOR_ALL_FIXED_PROPERTIES);
set_add(required_properties,
PROPERTIES_SELECTOR_ALL_USER_PROPERTIES);

SESSION_PROPERTIES_REGISTRATION_PARAMS_T params = {
    //Other parameters
    .required_properties = required_properties
};

```

When the listening client first registers a listener, it receives a notification for every client session that is currently open. When subsequent client sessions open, the listening client receives a notification for those clients.

When the listening client is notified of a session event, it receives the requested session properties as a map of keys and values.

When the listening client is notified of a session closing, it also receives the reason that the session was closed. If the client session becomes disconnected from the Diffusion server, the listener might not receive notification of session close immediately. If reconnection is configured for the client, when the client disconnects, its session goes into reconnecting state for the configured time (the default is 60 seconds) before going into a closed state.

Getting details of specific clients

Required permissions: view_session

A client can make an asynchronous request the session properties of any client session from the Diffusion server, providing the requesting client knows the session ID of the target client.

JavaScript

```
// Get fixed session properties
session.clients.getSessionProperties(sessionID,
diffusion.clients.PropertyKeys.ALL_FIXED_PROPERTIES)
    .then(function{

    });
```

Java and Android

```
// Get fixed session properties
ClientControl clientControl = session.feature(ClientControl.class);
clientControl.getSessionProperties(sessionID,
    Session.ALL_FIXED_PROPERTIES, sessionPropertiesCallback);
```

.NET

```
var _clientControl = session.GetClientControlFeature();
_clientControl.GetSessionProperties( sessionID,
    SessionControlConstants.AllFixedProperties, sessionPropertiesCallback
);
```

C

```
GET_SESSION_PROPERTIES_PARAMS_T params = {
    .session_id = session_id,
    .required_properties = properties,
    .on_session_properties = on_session_properties
};

get_session_properties(session, params);
```

Closing client sessions

Required permissions: view_session, modify_session

A client can close any client session, providing the requesting client knows the session ID of the target client.

Java and Android

```
ClientControl clientControl = session.feature(ClientControl.class);
clientControl.close(sessionID, callback);
```

.NET

```
var _clientControl = session.GetClientControlFeature();
_clientControl.Close( sessionID, callback );
```

Related concepts

[Session properties](#) on page 267

A client session has a number of properties associated with it. Properties are key-value pairs. Both the key and the value are case sensitive.

[Session filtering](#) on page 268

Session filters enable you to query the set of connected client sessions on the Diffusion server based on their session properties.

Handling client queues

Each client session has a queue on the Diffusion server. Messages to be sent to the client are queued here. You can monitor the state of these queues and set client queue behavior.

Receiving notifications of client queue events

Required permissions: view_session, register_handler

A client can register a handler that is notified when outbound client queues at the Diffusion server reach pre-configured thresholds.

Java and Android

```
ClientControl clientControl = session.feature(ClientControl.class);
clientControl.setQueueEventHandler(
    new ClientControl.QueueEventHandler.Default {

        @Override
        public void onUpperThresholdCrossed(
            final SessionId client,
            final MessageQueuePolicy policy) {

            // The action to perform when the queue upper threshold
            is crossed.
        }

        @Override
        public void onLowerThresholdCrossed(
            final SessionId client,
            final MessageQueuePolicy policy) {

            // The action to perform when the queue lower threshold
            is crossed.
        }
    }
);
```

Handling client queue events

Required permissions: view_session, modify_session

A client can respond to a client queue getting full by setting conflation on for the client. Conflation must be configured at the Diffusion server to have an effect.

A client is also able to set throttling on for specific clients, which also sets conflation. Using throttling without conflation can result in client queues overflowing.

Always use throttling and conflation in conjunction with a well-designed conflation strategy configured at the Diffusion server. For more information, see [Conflation](#) on page 93 and [Configuring conflation](#) on page 557.

Java and Android

```
ClientControl clientControl = session.feature(ClientControl.class);
```

```
clientControl.setThrottled(client, MESSAGE_INTERVAL, 1000,
    clientCallback);
```

.NET

```
var clientControl = session.GetClientControlFeature();
clientControl.SetThrottled( client, ThrottlerType.MESSAGE_INTERVAL,
    10, theClientCallback );
```

Flow control

A client application rapidly making thousands of calls to the Diffusion server might overflow the internal queues for the client, which results in that client session being closed. Flow control automatically protects against these queues overflowing by progressively delaying messages from the client to the Diffusion server.

Supported platforms: Android, Java, .NET, and C

The flow control mechanism is a feature of the Diffusion client libraries that works automatically to protect the following internal queues for an individual client from overflowing:

- The outbound queue on the client where messages are queued to be sent to the Diffusion server
- The queue on the Diffusion server where responses to service requests are queued.

If these queues overflow, the client session is terminated.

Flow control is intended to benefit those clients that send a lot of data to the Diffusion server – for example, updates to publish to topics. Usually, these clients are those located in the back-end of your solution that perform control functions.

When is flow control enabled?

The client determines whether to enable flow control and the amount of delay to introduce into the client processing based on a calculated value called back pressure. Back pressure is calculated using the following criteria:

- Depth of the outbound client queue
- The number of pending responses to service requests
- Whether the current active thread is a callback thread

Back pressure can have a value between 0.0 and 1.0. 1.0 is the maximum amount of back pressure. The method used to calculate back pressure might be subject to change in future releases.

Flow control introduces sleeps into the client processing. The length of these sleeps depends on the value of the back pressure. The maximum amount of delay introduced into client processing by flow control is 100 ms. The amount of delay introduced by flow control might be subject to change in future releases.

The flow control behavior of a client cannot be configured.

How to tell that flow control is enabled

When flow control is enabled for a Android, Java, or .NET client, the client logs messages at DEBUG level. The client logs each time a delay is introduced. The log message has the following form:

```
2016-09-26 11:15:48,344 DEBUG [PushConnectorPool-thread-18]
c.p.d.f.SleepingFlowControl(apply) - pressure=1.0 => sleep for 100
ms
```

The log message includes the current back pressure and the length of delay introduced.

The C client does not log its flow control behavior.

Actions to take

Diffusion clients can occasionally become flow controlled in response to very heavy load or unusual network conditions. However, if your clients are constantly being flow controlled, your Diffusion solution might not be correctly configured for the traffic load.

Consider taking the following actions:

- In your client design, ensure that if you have many requests to make to the Diffusion server that these requests are made from an application thread instead of a callback thread. Less flow control is applied when the active thread is a callback thread. For more information, see [Best practice for developing clients](#) on page 173.
- Ensure that your Diffusion server can handle the incoming messages from the clients. The default memory configuration might be causing the JVM running the Diffusion server to spend a lot of time in GC. For more information about tuning your JVM, see [Memory considerations](#).
- Increase the maximum queue size on the connector your client uses. This can be configured for individual connectors in the `Connectors.xml` configuration file or as a default value for all connectors in the `Server.xml` configuration file. For more information, see [Connectors.xml](#) on page 583 and [Server.xml](#) on page 563.

Logging from the client

Ensuring that your Diffusion client logs messages to inform of events and errors can be a valuable tool in developing and maintaining your clients.

Logging in JavaScript

The JavaScript client library logs messages to the console.

Log levels

Events are logged at different levels of severity. The log levels, ordered from most severe to least severe, are as follows:

Table 32: Log levels

Level	Description
error	Events that indicate a failure.
warn	Events that indicate a problem with operation.
info	Significant events.
debug	Verbose logging. Not usually enabled for production.
trace	High-volume logging of interest only to Push Technology Support. Push Technology Support may occasionally ask you to enable this log level to diagnose issues.

Configuring logging in the JavaScript client

You can use the JavaScript Unified API to enable and configure logging at runtime.

```
diffusion.log( level )
```

To disable logging at runtime, set the level to `silent`.

```
diffusion.log( 'silent' )
```

Note: Do not enable logging in your production clients. Use logging only during development of your clients.

Logging in Apple

The Apple client logs messages to the Apple system log facility.

Log levels

Events are logged at different levels of severity. The log levels, ordered from most severe to least severe, are as follows:

Table 33: Log levels

Level	Description
ERROR	Events that indicate a failure.
WARN	Events that indicate a problem with operation.
INFO	Significant events.
DEBUG	Verbose logging. Not usually enabled for production.
TRACE	High-volume logging of interest only to Push Technology Support. Push Technology Support may occasionally ask you to enable this log level to diagnose issues.

Configuring logging in the Apple client

You can use the Apple Unified API to enable and configure logging at runtime.

```
PTDiffusionLogging *const l = [PTDiffusionLogging logging];  
  
// Enable logging in the client library  
l.enabled = YES;  
  
// Change the level that the client logs at  
l.level = [PTDiffusionLoggingLevel trace];
```

Note: Do not enable logging in your production clients. Use logging only during development of your clients.

Logging in Android

The Android client uses `slf4j-android-logger` to log messages to the Android logging system.

Log levels

Events are logged at different levels of severity. The log levels, ordered from most severe to least severe, are as follows:

Table 34: Log levels

Level	Description
ERROR	Events that indicate a failure.
WARN	Events that indicate a problem with operation.
INFO	Significant events.
DEBUG	Verbose logging. Not usually enabled for production.
TRACE	High-volume logging of interest only to Push Technology Support. Push Technology Support may occasionally ask you to enable this log level to diagnose issues.

Configuring logging in the Android client

The Android JAR, `diffusion-android-x.x.x.jar`, contains a properties file, `logger.properties`. Edit this properties file to configure logging in the Diffusion Android client.

The default `logger.properties` file contains the following properties:

```
de.psdev.slf4j.android.logger.logTag=DiffusionAndroidClient
de.psdev.slf4j.android.logger.defaultLogLevel=INFO
```

For more information about `slf4j-android-logger`, see <https://github.com/PSDev/slf4j-android-logger>

Logging in Java

The Java client uses SLF4J to log messages. Provide a bindings library to implement the SLF4J API and log out the messages.

Log levels

Events are logged at different levels of severity. The log levels, ordered from most severe to least severe, are as follows:

Table 35: Log levels

Level	Description
ERROR	Events that indicate a failure.
WARN	Events that indicate a problem with operation.
INFO	Significant events.
DEBUG	Verbose logging. Not usually enabled for production.

Level	Description
TRACE	High-volume logging of interest only to Push Technology Support. Push Technology Support may occasionally ask you to enable this log level to diagnose issues.

Configuring logging in the Java client

The Java JAR, `diffusion-client-x.x.x.jar`, uses the SLF4J API to log messages. It does not include an implementation that outputs the log messages.

Many SLF4J implementations are available.

1. Choose your preferred SLF4J implementation.
2. Ensure that the bindings JAR is on the classpath of your Java client.
3. Configure the SLF4J implementation to provide the logging behavior you require.

Log4j2

Log4j2 is a third-party SLF4J implementation provided by the Apache Software Foundation. For more information, see <http://logging.apache.org/log4j/2.x/>.

You can configure your Java clients to use log4j2 by completing the following steps:

1. Get the log4j2 bindings libraries.

The JAR files can be downloaded from <https://logging.apache.org/log4j/2.0/download.html>.

The log4j2 JAR files are also located in the `lib/thirdparty` directory of the Diffusion installation.

2. Ensure that the `log4j-api.jar` and `log4j-core.jar` files are on the client classpath.
3. Create a configuration file and ensure that is present on the client classpath.

The following example `log4j2.xml` file outputs the log messages to a rolling set of files:

```
<Configuration status="warn" name="DiffusionClient">
  <Properties>
    <Property name="my.log.dir">../logs</Property>

    <!-- The log directory can be overridden using the
    system property 'my.log.dir'. -->
    <Property name="log.dir">${sd:my.log.dir}</Property>

    <Property name="pattern">%date{yyyy-MM-dd HH:mm:ss.SSS} |
%level | %thread | %marker | %replace{%msg}{\|}{ } | %logger%n%xEx
    </Property>
  </Properties>

  <Appenders>

    <RollingRandomAccessFile name="file" immediateFlush="false"
    fileName="${log.dir}/client.log"
    filePattern="${log.dir}/${date:yyyy-MM}/client-%d{MM-
dd-yyyy}-%i.log.gz">

      <PatternLayout pattern="${pattern}" />

    <Policies>
      <OnStartupTriggeringPolicy />
      <TimeBasedTriggeringPolicy />
      <SizeBasedTriggeringPolicy size="250 MB" />
    </Policies>
  </RollingRandomAccessFile>
</Appenders>
</Configuration>
```

```
        </Policies>

        <DefaultRolloverStrategy max="20" />
    </RollingRandomAccessFile>
</Appenders>

<Loggers>
    <AsyncRoot level="info" includeLocation="false">
        <AppenderRef ref="file" />
    </AsyncRoot>
</Loggers>
</Configuration>
```

For more information about configuring log4j2, see <https://logging.apache.org/log4j/2.0/manual/configuration.html>.

Logging in .NET

The .NET API produces logging information. The logging facility uses NLog.

NLog is included in the .NET client assembly. For more information about logging with NLog, see <https://github.com/NLog/NLog/wiki/>.

Logging basics

The NLog `Logger` class acts as the source of the log messages. In general, use one logger per class and pass in the name of the class as the logger name.

To log a message at a certain level use the write methods provided by the `Logger` class.

The following log levels are provided:

- Fatal
- Error
- Warn
- Info
- Debug
- Trace

These levels are listed in order from most severe to least severe.

Configuring the log output

You must configure NLog to output the log messages produced by your application to a target or targets. NLog provides a large number of targets for your output, including File and Console. NLog also enables you to create your own targets.

In addition to specifying targets for the log output, use NLog configuration to define rules that specify to which target log messages with particular levels or logger names are directed.

You can configure NLog in the following ways:

- Using a configuration file, `NLog.config`, that is located in the same directory as your client application.

For more information, see <https://github.com/NLog/NLog/wiki/Configuration-file>.

- Using the Configuration API to configure NLog in your application code.

For more information, see <https://github.com/NLog/NLog/wiki/Configuration-API>.

Logging in C

The C Unified API provides no integrated logging feature. You can log out from your client code using your preferred method or framework.

DEPRECATED: Classic API

The Classic API is our legacy API and will be deprecated in a forthcoming version of Diffusion.

Clients that use the APIs documented in this section are still supported for version 5.9, but for all future development use the new Unified API.

DEPRECATED: Java Client Classic API

The client Classic API provides the ability to connect to a Diffusion server as an external client from within any Java application.

For full API documentation, see [Java Classic API documentation](#)

How to use the Java client API

There is a single class called `ExternalClientConnection` which can be instantiated with the required connection details and used to make an actual connection.

The connection class is of the generic type `ServerConnection` and as such, once a connection is made, any notifications or messages from the server are passed through the `ServerConnectionListener` interface.

The topic or topics to subscribe to can be specified when connecting or at any time after connection.

The `ServerConnectionListener` specified will receive all messages for all topics. However, any number of additional topic listeners can be specified and messages for different topics routed to different listeners as required.

The API permits the following types of connection to be specified by using the `ServerDetails` (see connection package) specified when configuring the connection object:

Table 36: Connection types

TCP	For a standard connection over TCP/IP. This must connect to a standard client connector.
SSL	For a secure TCP/IP connection over SSL. This must connect to a client connector with SSL enabled
HTTP	For a connection using HTTP
HTTP/SSL	For a connection using HTTP over SSL.

By specifying more than one `ServerDetails`, fallback connections can be specified. If the first connection does not succeed, the second is tried, and so on.

For a detailed description of the API see the issued API documentation (in docs directory).

Authorization credentials

If authorization credentials are required by the Diffusion server, these are set at the `ConnectionDetails` level and used for all `ServerDetails`. Credentials can be set in a `ServerDetails` by creating a `Credentials` object and using `setCredentials` before connecting.

Credentials can also be sent to the server after connection using the method `sendCredentials` in `ExternalClientConnection`. In this case the credentials can be rejected by the server, in which case this is notified on the `serverRejectedCredentials` method of each `ServerConnectionListener`.

Certificates

Diffusion Java clients use certificates to validate the security of their connection to the Diffusion server. The client validates the certificate sent by the Diffusion server against the set of certificates trusted by the .

If the certificate sent by the Diffusion server cannot be validated against any certificates in the set trusted by the , you must set up a trust store for the client and add the appropriate certificates to that trust store.

Diffusion is authenticated using the certificates provided by your certificate authority for the domain you host the Diffusion server on.

1. Obtain the appropriate intermediate certificate from the certificate authority.
2. Use `keytool` to create a trust store for your client that includes this certificate.

For more information, see <https://docs.oracle.com/cd/E19509-01/820-3503/ggfska/index.html>

3. Use system properties to add the trust store to your client.

For example:

```
System.setProperty("javax.net.ssl.trustStore", "truststore_name");
```

Or at the command line:

```
-Djavax.net.ssl.keyStore=path_to_truststore
```

Reconnection

If a client unexpectedly loses connection, it can try to reconnect using the `reconnect` method. If the server has specified reconnection timeout (`keep-alive`) for the connector, the client can pick up the same session as before and receive all messages that were queued for it whilst disconnected. The topic state (that is, which topics the client is subscribed to) is also re-established on reconnection. If unable to reconnect, a new connection is established with the same topic set as used on the original connection. Successful reconnection or connection is notified on the normal `serverConnected` method and you can determine which has occurred using the `ServerConnection.isReconnected()` method.

There is no guarantee that messages in transit at the time of the disconnection will be redelivered. However, all messages marked as requiring acknowledgment by the server are delivered.

Failover

The Java client supports autofailover. For more information, see [Client failover](#) on page 1065.

Special features

Paged topic data handling

Where paged topic data is in use at the server there are features within the client API which simplify the handling of messages to and from such a topic.

Service topic data handling

Where service topic data is in use at the server there are features within the client API which simplify the handling of messages to and from such a topic.

Example: Simple client class

The following example shows a simple client class which sends a message containing “Hello” to the server and logs all messages it receives, until it receives a message from the server (Publisher) asking it to stop. It tries to connect through TCP first but if that fails it tries HTTP.

```
public class ClientApplication implements ServerConnectionListener
{
    private static final Logger LOG =
        LoggerFactory.getLogger(ClientApplication.class);

    private ExternalClientConnection theConnection;

    public ClientApplication() throws APIException {
        // Create Connection
        theConnection=
            new ExternalClientConnection(
                this,
                "ws://diffusion.example.com:80",
                "http://diffusion.example.com:80");
        // Connect, subscribing to a single topic
        theConnection.connect("MyTopic");
        // Send a message
        TopicMessage message =
            theConnection.createDeltaMessage("MyTopic");
        message.put("Hello");
        theConnection.send(message);
    }

    public void messageFromServer(
        ServerConnection serverConnection,
        TopicMessage message) {
        LOG.info("Message Received : {}",message);
        try {
            if (message.asString().equals("STOP")) {
                theConnection.close();
            }
        }
        catch(Exception ex) {
            ex.printStackTrace();
        }
    }

    public void serverConnected(ServerConnection serverConnection)
    {
        LOG.info("Connected to Server : {}",serverConnection);
    }

    public void serverTopicStatusChanged(
        ServerConnection serverConnection,
        String topicName,
        TopicStatus status) {
        LOG.info(
```

```

        "Topic {} at {} status changed to {}",
        topicName, serverConnection, status);
    }

    public void serverRejectedCredentials(
        ServerConnection serverConnection,
        Credentials credentials) {
        LOG.info("Server Rejected Credentials :
        {}", serverConnection);
    }

    public void serverDisconnected(ServerConnection
    serverConnection) {
        LOG.info("Disconnected from Server :
        {}", serverConnection);
    }
}

```

DEPRECATED: .NET Classic API

The .NET API comprises a number of packages.

For full API documentation, see [.NET Classic API documentation](#)

.NET Client Classic API

The ExternalClient API provides the ability to connect to a Diffusion server as an external client from within any .NET application.

There is a single class called `ExternalClient` which can be instantiated with the required connection details and used to make an actual connection.

The topic or topics to subscribe to can be specified when connecting or at any time after connection.

When a connection object is instantiated, subscribe to the `InboundMessageReceived` delegate, which receives all messages for all topics.

The API permits the following types of connection to be specified by using the `ServerDetails` specified when configuring the connection object:

Table 37: Types of connection that can be specified from the .NET client

TCP	For a standard connection over DPT. This connects to the External Client Connector.
TCPSSL	For a secure TCP/IP connection over DPTS. This connects to the External Client Connector.
HTTP	For a connection using HTTP protocol
HTTPSSL	For a secure connection using HTTP protocol
WEBSOCKET	For a connection using WebSocket protocol
WEBSOCKETSSL	For a secure connection using WebSocket protocol

For a detailed description of the API, see the issued documentation (in docs directory).

The following example shows a simple client class which sends a message containing 'Hello' to the server until it receives a message from the server (Publisher) asking it to stop.

```
public class ClientApplication : IServerConnectionListener,
ITopicListener
{
    #region Fields

    private readonly
    PushTechnology.DiffusionExternalClient.ExternalClient theClient;

    #endregion // Fields

    #region Constructor

    public ClientApplication()
    {
        var connectionDetails =
        ConnectionFactory.CreateConnectionDetails( "ws://
diffusion.example.com:80", "http://diffusion.example.com:80" );

        connectionDetails.Topics = new TopicSet("MyTopic");

        theClient = new
        PushTechnology.DiffusionExternalClient.ExternalClient(connectionDetails);

        // Add a topic listener - we are listening to all messages for
        this example, but individual topics can
        // also be used as selectors
        theClient.AddGlobalTopicListener( this );

        // Now connect - this is an asynchronous process, so we have to
        wait until ServerConnected is invoked
        theClient.Connect();
    }

    #endregion // Constructor

    #region Implementation of IServerConnectionListener

    /// <summary>
    /// Notification of connection.
    ///
    /// This is called when a connection to a server is established.
    /// </summary>
    /// <param name="connector">The server connector.</param>
    public void ServerConnected( IDiffusionClientConnector connector )
    {
        Console.WriteLine( "Connected to server: " + connector );

        // Send a message as we are now connected
        ITopicMessage message = theClient.CreateDeltaMessage( "MyTopic" );

        // Populate the message
        message.Put( "Hello" );

        // Send the message to the Diffusion server
        theClient.SendMessage( message );
    }

    /// <summary>
```

```

    /// Notification that the status for a topic that was subscribed to
    has changed.
    /// </summary>
    /// <param name="connector">The connector.</param>
    /// <param name="topicName">The name of the topic on which the
    status has changed.</param>
    /// <param name="statusType">The topic status change type.</param>
    public void ServerTopicStatusChanged( IDiffusionClientConnector
connector, string topicName, TopicStatusChangeType statusType )
    {
        Console.WriteLine(
            string.Format( "Topic status for '{0}' changed to '{1}'.",
topicName, statusType ));
    }

    /// <summary>
    /// Notification of rejected credentials from the server.
    /// </summary>
    /// <param name="connector"></param>
    /// <param name="credentials"></param>
    public void ServerRejectedCredentials( IDiffusionClientConnector
connector, V4Credentials credentials )
    {
        Console.WriteLine( "Server rejected credentials." );
    }

    /// <summary>
    /// Notification of disconnection.
    ///
    /// The reason for the disconnection can be established by checking
    the state of the connection
    /// using IDiffusionClientConnector.State.
    /// </summary>
    /// <param name="connector">The server connector.</param>
    /// <param name="args">The arguments which can be interrogated for
    the state and details of a server closure.</param>
    public void ServerDisconnected( IDiffusionClientConnector
connector, ServerClosedEventArgs args )
    {
        Console.WriteLine( "Disconnected from server." );
    }

#endregion

#region Implementation of ITopicListener

    /// <summary>
    /// Handles a message received from an IMessageSource.
    ///
    /// This handles an incoming message from a specified source.
    /// </summary>
    /// <param name="source">The message source.</param>
    /// <param name="message">The message.</param>
    public bool HandleTopicMessage( IMessageSource source,
ITopicMessage message )
    {
        if ( message.AsString().Equals( "STOP" ) )
        {
            theClient.Disconnect();
        }

        return false;
    }
}

```

```
#endregion  
}
```

Connection events

Events that are invoked when a connection to the Diffusion server is established, fails, or is lost are invoked synchronously.

The following connection events are invoked synchronously by the .NET client library:

- `DiffusionServerConnected` is invoked when the client library successfully establishes a connection to the Diffusion server
- `DiffusionServerConnectionFailed` is invoked when the client library is unable to establish a connection to the Diffusion server
- `DiffusionServerDisconnected` is invoked when an established connection to the Diffusion server is lost

Because these events are invoked synchronously, do not perform any long-running or blocking operations on these event threads. If you want to make another connection attempt after one of these events is invoked, create another thread to perform this task.

DEPRECATED: JavaScript Classic API

The JavaScript API provides web developers with a simple means of connecting to and interacting with a Diffusion server from within a web browser. The API takes care to select the most appropriate underlying transport from those available.

For the list of supported web browsers, see [Browser support](#) on page 56.

Using the JavaScript Classic API

The JavaScript client library is located in the `clients/js` directory of the Diffusion installation. Two versions are provided, an uncompressed file, and a minimized file with the extension `min.js`.

For full API documentation, see [JavaScript Classic API documentation](#)

To enable the Diffusion client for production use, host the `diffusion-js-classic.js` client library on a dedicated web server and load the client library into your web page:

```
<script type="text/javascript" src="library_location/diffusion-js-classic.js"></script>
```

Dependent transport files

The JavaScript client depends on other files located in the same directory: `clients/js/diffusion-flash.swf` and `clients/js/diffusion-silverlight.xap`. These files provide Flash and Silverlight transport capabilities, respectively. Removing these files prevents the JavaScript client from using these transports.

You can configure the JavaScript client to point to specific versions of both the Flash and the Silverlight transports. This is available through the connection details.

```
var connectionDetails = new DiffusionClientConnectionDetails();  
  
connectionDetails.libPath = "/lib/js/diffusion";  
connectionDetails.libFlashPath = "diffusion-flash.swf";  
connectionDetails.libSilverlightPath = "diffusion-silverlight.xap";
```

Connection details

The `DiffusionClientConnectionDetails` object has over 20 attributes that change the way that the client behaves. Any attributes that are not set are provided with default values when used.

You can provide an anonymous object instead of instantiating a new `DiffusionClientConnectionDetails` object.

Connecting

A Diffusion client is a singleton with global scope. It can be called from anywhere. To connect to Diffusion, call the `connect` method. The method takes two parameters, first is the connection details and the optional second object is the client credentials. The following example is using an anonymous connection object:

```
DiffusionClient.connect({
  debug : true,
  onCallbackFunction : function(isConnected) {
    console.log("Diffusion client is connected: " + isConnected);
  }
})
```

If the client connection fails, the JavaScript client attempts to connect through other protocols. This is called protocol cascading.

Credentials

Credentials can either be supplied on the `connect` method or set separately using the `DiffusionClient.setCredentials(...)` method. These credentials are used for all transports that are used to connect to Diffusion. The `DiffusionClientCredentials` object is a simple one of `username` and `password` attributes.

```
// Connect with supplied credentials
DiffusionClient.connect({...}, {
  username : "foo",
  password : "bar"
});
```

```
// Connect, and send credentials later
DiffusionClient.connect({
  onCallbackFunction : function() {
    DiffusionClient.sendCredentials({
      username : "foo",
      password : "bar"
    });
  }
});
```

If authentication of the client connection fails, the JavaScript client attempts to protocol cascade and to connect through a different protocol with the same credentials. Use the `onConnectionRejectFunction` to close the client connection and prevent this from happening.

Events

The connection details have attributes that are listeners for certain events. If these are set in the connection object, they are called when these events happen.

Table 38: JavaScript functions called on events

Function	Description
<code>onDataFunction</code>	This function is responsible for handling messages from Diffusion. This function is called with an argument of <code>WebClientMessage</code> . This function is called even if there is a topic listener in place for a topic.
<code>onBeforeUnloadFunction</code>	This function is called when the user closes the browser or navigates away from the page.
<code>onCallbackFunction</code>	This function is called when Diffusion has connected, or exhausted all transports and cannot connect. This function is called with a boolean argument.
<code>onInvalidClientFunction</code>	This function is called when an invalid Diffusion operation is called, for instance if <code>Diffusion.subscribe</code> is called before <code>Diffusion.connect</code>
<code>onCascadeFunction</code>	This function is called when the client cascades transports. The function is called with an argument of the <code>{String}</code> transport name or <code>NONE</code> if all transport are exhausted.
<code>onPingFunction</code>	This function is called with an argument of <code>PingMessage</code> when the ping response has been returned from the server.
<code>onAbortFunction</code>	This function is called when the Diffusion server terminates the client connection (or the connection has been banned).
<code>onLostConnectionFunction</code>	This function is called when the client loses connection with the Diffusion server
<code>onConnectionRejectFunction</code>	This function is called when the client connection is rejected by the Diffusion server because of incorrect credentials. Use this function to close the connection when the authentication fails and prevent the client attempting to connect over a different protocol with the same incorrect credentials.
<code>onMessageNotAcknowledgedFunction</code>	This function is called when a message that is requested as <code>Acknowledge</code> did not respond in time.
<code>onServerRejectedCredentialsFunction</code>	This function is called after a <code>DiffusionClient.sendCredentials</code> and the server rejected the credentials.
<code>onTopicStatusFunction</code>	This function is called if the status of a subscribed topic changes.

Receiving messages

The `onDataFunction` with a class called `WebClientMessage` contains only one message even if the messages sent from the Diffusion server are batched. If this is the case, this method is repeatedly called until all of the messages are exhausted. The `WebClientMessage` class wraps the message sent from the Diffusion server with utility methods like `isInitialTopicLoad()` and `getTopic`. See the jsdoc for the full list of utility methods.

Sending messages

There are two ways of sending messages to the Diffusion server:

- The `DiffusionClient.send(topic, message)` method
- The `sendTopicMessage` method

If user headers are required, it is best to use the `TopicMessage` class. The following example shows how to send a message using the `TopicMessage` class.

```
var topicMessage = new TopicMessage("Echo", "This is a message");
topicMessage.addUserHeader("Header1");
topicMessage.addUserHeader("Header2");

DiffusionClient.sendTopicMessage(topicMessage);
```

Subscribing and unsubscribing

To subscribe and unsubscribe use the `DiffusionClient.subscribe(topic)` and `DiffusionClient.unsubscribe(topic)` methods respectively. The parameter can be a topic, a topic selector, or a comma-delimited list of topics

Topic listeners

During the lifetime of the connection, it might be required to have modular components that are notified about topic messages. These are topic listeners. A topic listener calls a supplied function with a `WebClientMessage` object when the topic of the message matches the pattern supplied. It is also worth noting that the `onDataMessage` function is called as well as the topic listener function. You can have many topic listeners on the same topic pattern if required. For example, if you want to be notified about a particular topic, you can use the following example code:

```
DiffusionClient.addTopicListener("^Logs$", onDataTradeEvent);
```

Note:

The characters `^` `$` are regular expression characters. For more information, see [Regular expression](#). The preceding code example means that the listener is only interested in receiving the message event if the topic is Logs. If the following example code is used, any topic name that has Logs in it matches.

```
var listenerRef = DiffusionClient.addTopicListener("Logs",
onLogEvent, this);
```

Retain the listener reference if you want to remove the topic listener at a later date.

Failover

The JavaScript client does not support autofailover. You can still implement this using the `onLostConnectionFunction`.

Special Features

Paged topic data handling

Where paged topic data is in use at the server there are features within the client API which simplify the handling of messages to and from such a topic.

Reconnecting with the JavaScript Classic API

The JavaScript Classic API supports reconnection. If you have reconnection enabled and you lose your connection to a server, you can reestablish it, using the same client ID and with the client subscriptions preserved.

The JavaScript API listens for pings from the server and raises an event if the connection to the server is lost.

To enable the liveness monitor, set the `enableLivenessMonitor` parameter to true inside the client connections details. For example:

```
var connectionDetails = { ...
    enableLivenessMonitor : true,
    ... };

DiffusionClient.connect(connectionDetails);
```

The Diffusion server sends out pings at regular intervals. The length of this interval is configured at the server by using the `system-ping-frequency` element in the `Connectors.xml` configuration file.

The liveness monitor in the ActionScript client library listens for the pings from the server and uses them to estimate the ping interval. The liveness monitor takes an average of the time between the pings it receives to estimate the ping interval. It revises this estimation each time it receives a ping, until it has received ten pings. After ten pings the liveness monitor has obtained the estimated ping interval that it uses for the rest of the client session.

If the liveness monitor does not receive a ping within a time interval equal to twice the length of the estimated ping interval, it considers the connection lost and raises a connection lost event.

Connection lost events can be raised by the liveness monitor or triggered by other events, such as an unexpectedly closed connection.

You can implement an event listener in your client that listens for a connection lost event and reacts to it by using the `reconnect()` method to reestablish the connection.

Reconnection example

To reconnect after you lose connection, you must use the `reconnect()` method. You cannot reconnect an aborted client.

The following code shows how to setup an event listener for connection events, if the connection has been lost how to reconnect and how to tell if you have successfully reconnected the client.

```
var connectionDetails = {
    onCallbackFunction : function( isConnected, isReconnect ) {
        if( !isConnected && isReconnect ) {
            DiffusionClient.reconnect();
        }
    },
    onLostConnectionFunction : function() {
        DiffusionClient.reconnect();
    }
};
```

```
DiffusionClient.connect(connectionDetails);
```

For more information, see [JavaScript Classic API documentation](#).

Service topic data in JavaScript Classic API

The JavaScript API provides a basic interface for using service topics.

The API consists of a service topic handler to process responses and using the generic `DiffusionClient.command(...)` method to send service requests.

The common sequence to follow is:

1. Add a topic listener, to capture the service topic load message
2. Subscribe to the service topic
3. With the ITL from the service topic create a service topic handler
4. Remove the topic listener
5. Send command messages to the service
6. Process any response in the function passed to the handler

To create a handler using the

`DiffusionClient.createServiceTopicHandler(TopicMessage, function)` you must pass in the ITL of the service topic and the function that is called when a service response is received. This function will be called with a `CommandMessage` as an argument.

To make service requests you must use the

`DiffusionClient.command(string, string, TopicMessage)` method to send command messages. The first string is the command to send. The second string is a correlation ID for the response. The `TopicMessage` is the message sent to the client with the correct topic and any additional headers or payload you want to send in the request.

Use an ordinary topic listener to get the ITL to create the service topic handler. This listener is not required for any subsequent message processing and you are encouraged to remove it after you have the ITL.

You must generate a unique value for the correlation ID.

Paged topic data in JavaScript Classic API

The JavaScript API provides an interface for using paged topics.

The API contains the following classes:

PagedTopicHandler

Provides methods that enable you to change and navigate the page view.

PagedTopicListener

Provides callbacks for when an action is performed on a page or the topic.

PageStatus

Contains values that describe the status of a page or topic.

For more information, see [JavaScript Classic API documentation](#).

Handler methods

The following example code shows some of the handler methods wrapped in functions you can use to add buttons to a client's user interface.

```
// Get the next page in the topic
function next() {
  handler.next();
}
```

```

}

// Get the previous page in the topic
function prior() {
  handler.prior();
}

// Get the first page in the topic
function first() {
  handler.first();
}

// Get the last page in the topic
function last() {
  handler.last();
}

// Close the paged view of the topic
function pagedclose() {
  handler.close();
}

```

Listener methods

The following example code creates a `PagedTopicHandler` and implements the listener methods `add`, `page`, `ready`, `statusChanged`, and `update`.

```

var connectionDetails = {
  onDataFunction : function() {
  },
  onCallbackFunction : function() {
    // Creates the handler for a topic
    DiffusionClient.createPagedTopicHandler(topic, {
      ready : function(handler) {
        // Add here the code you want to run when the handler
        // is created.

        // For example, open a view that is 20 lines long and
        // contains the first page of data:
        handler.open(20, 1);
      },
      page : function(handler, status, lines) {
        // Add here the code you want to run when the page is
        // loaded.
      },
      update : function(handler, status, index, line) {
        // Add here the code you want to run when a line on
        // the current page is updated.

        // For example, refresh the page.
        handler.refresh();
      },
      statusChanged : function(handler, status) {
        // Add here the code you want to run when the status
        // of the current page changes.

        // For example, check whether the page is dirty and
        // refresh the page if this is true.
        if (status.isDirty){
          handler.refresh();
        }
      }
    });
  }
};

```

```

    }
  },
  add : function(handler, status, lines) {
    // Add here the code you want to run when a line is
    // added to the current page.
  }
  });
},
debug : true
};

```

DEPRECATED: ActionScript Classic API

The ActionScript Classic API is bundled in a library called `clients/flex/diffusion-flex.swc`.

This can be embedded into a Flex/Flash or Air application. Full asdoc is issued with the product so the sections below provide a brief outline of the uses for the classes and examples of their use. The ActionScript library is based on the event model. There are many different types of events that a `DiffusionClient` dispatches. These must be registered before notification happens.

For full API documentation, see [Flex Classic API documentation](#)

Diffusion also provides a debug-friendly version of the library: `diffusion-flex-debug-version.swc`, where *version* is the Diffusion version number, for example 5.9.24. This version is larger, but you can embed it into you application to receive additional information from any stack traces.

Using the ActionScript Classic API

`DiffusionClient` (`com.pushtechnology.diffusion.DiffusionClient`) is the main class that is used. This class enables the user to set all of the connection and topic information.

DiffusionClient

Connection example

```

import com.pushtechnology.diffusion.ServerDetails;
import com.pushtechnology.diffusion.ConnectionDetails;
import com.pushtechnology.diffusion.DiffusionClient;
import
  com.pushtechnology.diffusion.events.DiffusionConnectionEvent;
import com.pushtechnology.diffusion.events.DiffusionTraceEvent;
import com.pushtechnology.diffusion.events.DiffusionMessageEvent;
import
  com.pushtechnology.diffusion.events.DiffusionExceptionEvent;
import com.pushtechnology.diffusion.events.DiffusionPingEvent;

// Get a new DiffusionClient
var theClient:DiffusionClient = new DiffusionClient();

// Set everything to enable the cascading
var serverDetails:ServerDetails = new ServerDetails("https://
diffusion.example.com:443");
var connectionDetails:ConnectionDetails = new
  ConnectionDetails(serverDetails, "Trade");
connectionDetails.setCascade(true);

```

```

// Add the listeners
theClient.addListener(DiffusionConnectionEvent.CONNECTION,
    onConnection);
theClient.addListener(DiffusionMessageEvent.MESSAGE,
    onMessages);
theClient.addListener(DiffusionTraceEvent.TRACE, onTrace);
theClient.addListener(DiffusionExceptionEvent.EXCEPTION,
    onException);
theClient.addListener(DiffusionPingEvent.PING, onPing);

// Connect
theClient.connect(connectionDetails);

```

Setting credentials

If credentials are required by the Diffusion server then use the `setCredentials` method on the `DiffusionClient` class. The `DiffusionClientCredentials` class takes a constructor argument of username and password. Please bear in mind, that these are only tokens and can contain any information that the `AuthorisationHandler` requires. However, if you set the username as an empty string (that is, an anonymous user) the password is not stored and you cannot retrieve it with `getCredentials`.

```

var credentials:DiffusionClientCredentials = new
    DiffusionClientCredentials(username, password);
theClient.setCredentials(credentials);

```

Connection event

The connection event contains information about the success of the connection attempt. Below is a coding example of the possibilities for the connect event.

```

public function onConnection(event:DiffusionConnectionEvent) : void {
    if (event.wasConnectionRejected()) {
        theClientIDBox.text = "Connection Rejected by Diffusion Server";
    } else if (event.wasClientAborted()) {
        theClientIDBox.text = "Connection aborted";
    } else if (event.isConnected()) {
        theClientIDBox.text = event.getClientID();
        theConnectedTransportLabel.text = theClient.getTransportMode();
    } else {
        theClientIDBox.text = "Connection failed " +
            event.getErrorMessage();
    }
}

```

You can receive a connection event after you have successfully connected, which might be because of a lost connection, or in the case of client aborted the Diffusion server has deliberately closed the client connection. This normally means that a publisher has aborted the connection and the client must not try and connect again.

onMessage event

When messages arrive from the Diffusion server on a subscribed topic, the `DiffusionMessageEvent` is dispatched. Contained in the event is a `TopicMessage` object (`com.pushtechnology.diffusion.TopicMessage`). This class contains helper methods that surround the message itself, like `getTopic()` and `isInitialTopicLoad`. For more information, see the API documentation.

```
public function onMessages(event:DiffusionMessageEvent) : void {
    var message:TopicMessage = event.getTopicMessage();
    ...
}
```

Subscriptions

Once the client has connected, you can issue subscribe and unsubscribe commands. The subscribe and unsubscribe methods take a string format, that can be a topic selection pattern, a list of topics that are comma delimited or a single topic.

Send

Once connected a client can send messages to the Diffusion server on a particular topic. To do this, use the send method.

```
theClient.send("Fred", "Hello publisher that looks after Fred");
```

In the example above, the publisher that looks after topic Fred receives a `messageFromClient` notification. If a message with user headers or encoding is required, you must use the `sendTopicMessage` method. A `TopicMessage` (`com.pushtechnology.diffusion.TopicMessage`) allows for the setting of user headers and message encoding

Ping

The client can ping the Diffusion server. To receive the Ping response, the listener is added to the client.

```
theClient.addEventListener(DiffusionPingEvent.PING, onPing);
```

The resulting ping event has two attributes in it, firstly the time taken to do a round trip from the client to the Diffusion server and back again. The second attribute is how many items are currently in the client queue at the server. This information enables the client to get some vital connection information. It is down to the implementation of the client to specify the ping frequency, if at all required.

Topic listeners

During the life time of the connection, it might be required to have modular components notified about topic messages – these are topic listeners. A topic listener calls a supplied function with a `TopicMessage` object when the topic of the message matches the pattern supplied. The topic listeners are called in the order that they are added, and before the default `DiffusionMessageEvent.MESSAGE`, that is called as well as the topic listener event. You can have many topic listeners on the same topic pattern if required. The function supplied in charge of processing the message can signal that

a message is consumed, returning TRUE. In this case, this message is not relayed to subsequent TopicListeners and the default listener. For example, if you want to be notified about a particular topic, use the following code:

```
var listenerRef:String = theClient.addTopicListener( "^Logs$",  
theLogsDataGrid.onMessage );
```

Note the syntax here, the ^ \$ are regex pattern strings, the above means that the listener is only interested in receive the message event if the topic is Logs. If the following was issued.

```
var listenerRef:String = theClient.addTopicListener( "Logs",  
theLogsDataGrid.onMessage );
```

Any topic name that has “Logs” in it matches. You must store the reference to remove the topic listener at a later date.

Timed topic listeners

A timed topic listener calls a supplied function with an array of topicMessage objects when the topic of the message matches the pattern, and only if the time supplied by the arguments has expired. Otherwise, the TopicMessage is stored until the time expired.

Note:

The function in charge of processing the message cannot determine if a message is consumed as you can do in a topic listener. For example, if you want to be notified about a particular topic, use the following code:

```
var timedListenerRef:String =  
theClient.addTimedTopicListener( "^Logs$",  
theLogsDataGrid.onMessage, 2000, false );
```

The third parameter is the frequency at which the function supplied is called. The optional fourth parameter can be set if this function must be called, even if no messages are stored.

Failover

The ActionScript client supports autofailover. For more information, see [ActionScript failover](#) documentation.

Special features

Paged topic data handling

Where paged topic data is in use at the server there are features within the client API which simplify the handling of messages to and from such a topic.

Reconnecting with the ActionScript Classic API

The ActionScript Classic API supports reconnection. If you have reconnection enabled and you lose your connection to a server, you can reestablish it, using the same client ID and with the client subscriptions preserved.

Liveness monitor

The ActionScript Classic API implements a liveness monitor that listens for pings from the server and raises an event if the connection to the server is lost.

Before you make a connection to the Diffusion server, enable the liveness monitor by using the `enableLivenessMonitor()` method. For example:

```
client.enableLivenessMonitor(true);
client.connect(connectionDetails);
```

The Diffusion server sends out pings at regular intervals. The length of this interval is configured at the server by using the `system-ping-frequency` element in the `Connectors.xml` configuration file.

The liveness monitor in the ActionScript client library listens for the pings from the server and uses them to estimate the ping interval. The liveness monitor takes an average of the time between the pings it receives to estimate the ping interval. It revises this estimation each time it receives a ping, until it has received ten pings. After ten pings the liveness monitor has obtained the estimated ping interval that it uses for the rest of the client session.

If the liveness monitor does not receive a ping within a time interval equal to twice the length of the estimated ping interval, it considers the connection lost and raises a `DiffusionConnectionEvent` whose `hasLostConnection()` method returns true.

You can implement an event listener in your client that listens for this event and reacts to it by using the `reconnect()` method to reestablish the connection.

Warning:

The liveness monitor relies on server pings being received at regular intervals. If the server pings the client in addition to the regular pings, these additional pings can cause the liveness monitor to make an incorrect estimate of the ping interval. Because this incorrect estimate is shorter than the correct ping interval, this can cause the liveness monitor to incorrectly consider a connection lost.

To avoid this problem, if you are using the liveness monitor, ensure that you do not ping the client from a publisher or from the Introspector.

Reconnection example

To reconnect, you must use the `reconnect` method when you lose a connection. You cannot reconnect an aborted client.

The following code shows how to setup an event listener for connection events, if the connection has been lost how to reconnect and how to tell if you have successfully reconnected the client.

```
var client:DiffusionClient = createClient();

function onConnectionEvent(event:DiffusionConnectionEvent) {
    if (event.hasLostConnection()) {
        client.reconnect();
    }
    else if (event.isConnected()) {
        if (event.isReconnected()) {
            // Successful reconnection
        }
    }
}

function createClient():DiffusionClient {
    var client:DiffusionClient = new DiffusionClient();
    client.addEventListener(DiffusionConnectionEvent.CONNECTION,
        onConnectionEvent);
    return client;
}
```

```
}
```

Logging Flash

You can get additional information out of your Flash client by using a debug version of the library or of Flash Player.

Debug version of the Diffusion Flash client

Diffusion also provides a debug-friendly version of the Flash client libraries: `diffusion-flex-debug-version.swc`, where *version* is the Diffusion version number, for example 5.9.24. You can use this to output additional debug information, such as line numbers, in any stack traces that you experience.

Debug version of Flash Player

1. Download the debug version for the Flash Player. Flash player debug versions are available at the following location: <http://www.adobe.com/support/flashplayer/downloads.html>
2. Create a `mm.cfg` file

The table below shows where to create a `mm.cfg` file

The following text is an example of entries in a `mm.cfg` file.

```
# Enables policy file logging
PolicyFileLog=1
# Optional; do not clear log at startup
PolicyFileLogAppend=0
ErrorReportingEnable=1
TraceOutputFileEnable=1
```

Viewing the log files

In the following directory there can be the following files: `flashlog.txt` and `policyfiles.txt`

Table 39: Location of the `flashlog.txt` file

Windows 95/98/ME	%HOMEDRIVE%\%HOMEPATH%
Windows 2000/XP	C:\Documents and Settings\username
Windows Vista	C:\Users\username
OS X/macOS	/Library/Application Support/Macromedia
Linux	/home/username

Table 40: Location of the `policyfiles.txt` file

Windows 95/98/ME/2000/XP	C:\Documents and Settings\username \Application Data\Macromedia\Flash Player\Logs
Windows Vista	C:\Users\username\AppData\Roaming \Macromedia\Flash Player\Logs

OS X/macOS	/Users/username/Library/Preferences/Macromedia/Flash Player/Logs
Linux	home/username/Macromedia/Flash_Player/Logs

DEPRECATED: Silverlight Classic API

The Silverlight Classic API is bundled in an assembly called `clients/silverlight/ PushTechnology.Transports.dll`. This can be embedded into a Silverlight application.

Full API documentation is issued with the product, so the sections below provide a brief outline of the uses for the classes and examples of their use. The Silverlight library is based on an asynchronous event model. There are a few events that the client object invokes. The client must subscribe to these events before notification can happen.

For full API documentation, see [Silverlight Classic API documentation](#)

Using the Silverlight Classic API

The `DiffusionClient` class is the main class that is used. This class enables the user to set all of the connection and topic information.

Instantiation and connection example

```

theClient = new DiffusionClient( Dispatcher );

// Instantiate the server details object and the initial topic
to subscribe to
ServerDetails details = new ServerDetails( "http://
localhost:8080", "SpotOnly" );

// Add the server details to the client
theClient.AddServerDetails( details );

// Add the event listeners
theClient.ConnectionStatus +=
DiffusionConnectionStatus;
theClient.MessageReceived +=
theClient_MessageReceived;

// Now connect
theClient.Connect();

```

Setting credentials

If credentials are required by the Diffusion server then use the `Credentials` property on the `DiffusionClient` class. The `DiffusionClientCredentials` class takes a constructor argument of `userName` and `password`. Please bear in mind that these are only tokens and can contain any information that the `AuthorisationHandler` requires. However, if you set the username as an empty string (that is, an anonymous user) the password is not stored and you cannot retrieve it with `getCredentials`.

```

theClient.Credentials = new DiffusionClientCredentials(
"username", "password" );

```

Rejection of credentials event

If the credentials are rejected by the Diffusion server, a `ServerRejectedCredentials` event is fired. This can be subscribed to by using the following code:

```
theClient.ServerRejectedCredentials +=  
    ServerRejectedCredentials;
```

Message not acknowledged event

When a message is created with the "acknowledge" flag, this event is fired when a message is not acknowledged by the Diffusion server within the specified time period. This can be subscribed to by using the following code:

```
theClient.MessageNotAcknowledged += MessageNotAcknowledged;
```

The `ConnectionStatus` event

The `ConnectionStatus` event contains information about whether the connection was successful. Here follows an example of the usage of this event.

```
/// <summary>  
    /// Called when the connection state to Diffusion changes.  
    /// </summary>  
    /// <param name="sender"></param>  
    /// <param name="e"></param>  
    void DiffusionConnectionStatus(  
        object sender,  
        DiffusionConnectionStatusEventArgs e )  
    {  
        switch( e.StatusType )  
        {  
            case DiffusionConnectionStatusType.ConnectionFailed:  
            {  
                Dispatcher.BeginInvoke( () =>  
                    MessageBox.Show( "Unable to connect to  
Diffusion. Diffusion reports: " + e.ExtraData,  
                                    "Connection failed",  
                                    MessageBoxButton.OK ) );  
            }  
            break;  
  
            case DiffusionConnectionStatusType.ConnectionReset:  
            {  
                Dispatcher.BeginInvoke( () =>  
                    MessageBox.Show( "The connection to Diffusion  
has been reset. Diffusion reports: " + e.ExtraData,  
                                    "Connection failed",  
                                    MessageBoxButton.OK ) );  
            }  
            break;  
        }  
    }  
}
```

Note: You can receive a `ConnectionStatus` event after you have successfully connected; it might be because of a lost connection, or in the case of `ConnectionAborted`, the Diffusion server has closed the client connection. This normally means that a publisher has aborted the connection and the client must not try to connect again.

The MessageReceived event

When messages arrive from the Diffusion server on a subscribed topic, the `MessageReceived` event is fired. This event contains a sender and a `TopicMessageEventArgs` object which itself contains a `TopicMessage` object which can be interrogated to discover the contents of the received message.

The TopicStatusMessageReceived event

When the status of a topic changes on the Diffusion server, the `TopicStatusMessageReceived` event is fired. This event contains a sender and a `TopicStatusMessageEventArgs` object which contains the alias of the topic on which the status has changed. Currently, only the notification of the removal of a topic is implemented.

Subscriptions

After the client has connected, you can issue `Subscribe` and `Unsubscribe` commands. These commands take string arguments which can be a topic selection pattern, a list of topics that are comma-delimited, or a single topic.

Sending non-encoded topic messages

Once the client has connected, it can send messages to the Diffusion server on a particular topic. To do this, use either the `Send` or `SendTopicMessage` methods on the `DiffusionClient` object, as shown in the following code:

```
theClient.Send( "Fred", "Hello, publisher that looks after Fred" );

TopicMessage message =
    new TopicMessage( "Fred", "Hello, publisher that looks after Fred" );

theClient.SendTopicMessage( message );
```

Note: The `TopicMessage` itself contains methods to set (for instance) user headers and encoding, or the convenience methods described below can handle the alternate encoding scenarios.

In the above examples, the publisher that looks after the topic `Fred` receives a `messageFromClient` notification internally.

Sending an encrypted topic message

Sending an encrypted topic message is achieved by calling the `SendTopicMessageEncrypted` method on the `DiffusionClient` object by using the following code:

```
theClient.SendTopicMessageEncrypted( new TopicMessage( "Fred", "Hello, publisher that looks after Fred" ) );
```

This sets the relevant encoding flags on the message itself, and the message will be encrypted immediately prior to sending to the Diffusion server.

Note: Because of the limitations of HTTP-based transports, attempting to send a message of this type results in a non-encoded message being sent.

Sending a compressed topic message

Sending a compressed topic message is achieved by calling the `SendTopicMessageCompressed` method on the `DiffusionClient` object by using the following code:

```
theClient.SendTopicMessageCompressed( new TopicMessage( "Fred",  
"Hello, publisher that looks after Fred" ) );
```

This sets the relevant encoding flags on the message itself, and the message will be compressed immediately prior to sending to the Diffusion server.

Note: Because of the limitations of HTTP-based transports, attempting to send a message of this type results in a non-encoded message being sent.

Sending a Base64-encoded topic message

Sending a Base64-encoded topic message is achieved by calling the `SendTopicMessageBase64` method on the `DiffusionClient` object to using the following code:

```
theClient.SendTopicMessageBase64( new TopicMessage( "Fred", "Hello,  
publisher that looks after Fred" ) );
```

This sets the relevant encoding flags on the message itself, and the message will be Base64-encoded immediately prior to sending to the Diffusion server.

Ping

A client can ping the Diffusion server. To process the ping response, the user monitors the `MessageReceived` event and checks for a message type of `PingServer`, as shown in the following code:

```
private void HandleServerPingMessage( TopicMessageEventArgs e )  
{  
    var message = e.Message as PingMessage;  
  
    if( message != null )  
    {  
        tbElapsedTime.Text = message.ElapsedTime.ToString();  
        tbQueueSize.Text = message.QueueSize.ToString();  
    }  
}
```

Fetch

Using the `fetch` method, a client can send a request to the Diffusion server for the current state of a topic, which returns a state message to the client. A client can do this even if not subscribed to the topic.

Topic listeners

During the lifetime of the connection, it might be required to have modular components that get notified about topic messages – these are known as topic listeners. A topic listener calls a supplied function with a `TopicMessage` object when the topic of the message matches the pattern supplied. It is also worth noting that the `OnMessageReceived` event is called as well as the topic listener event itself.

You can have many topic listeners on the same topic pattern if required. For example, if you want to be notified about a particular topic, use the following code:

```
instrumentListener = theClient.AddTopicListener(
    "^SpotOnly$", ProcessInstruments, this );
```

Note: The “^” and “\$” characters are regular expression pattern strings; the above means that the listener is only interested in receiving the message if the topic is SpotOnly.

Enabling JavaScript method invoking

To call JavaScript functions (and they are permitted to do so by the Silverlight runtime), use the following method call:

```
theClient.InitialiseJavaScriptMethodInvoking( HtmlPage.Window );
```

Listening to internal transport debug messages

To subscribe to the internal log tracings of the Silverlight API, the user can subscribe to the `DiffusionTraceEvent` on the `DiffusionClient` object, as shown in the following code:

```
theClient.DiffusionTraceEvent += theClient.DiffusionTraceEvent;
```

This enables the user to monitor all internal debug messages within the Silverlight API.

DEPRECATED: iOS Classic API

The static libraries and header files that comprise the iOS API are provided in the file `diffusion-iphoneos-classic-version.zip`, where *version* is the version number, for example 5.9.24.

The iOS library is provided with the Diffusion server installation in the `clients/apple` folder.

The iOS library uses the delegate model. There are a number of Diffusion events dispatched by a `DFClient` instance as Objective-C messages. To receive these events, provide an implementation conforming to the `DFClientDelegate` protocol.

The API documentation is also available as an Xcode docset. Once installed into Xcode the iOS client can be browsed within the **Xcode** Documentation viewer.

For full API documentation, see [iOS Classic API documentation](#)

Capabilities

To see the full list of capabilities supported by the iOS Classic API, see .

Support

Table 41:

--	--

Getting started with iOS Classic API

Create a client application within minutes that connects to the Diffusion server.

Before you begin

Ensure that the iOS client libraries are available on your development system. The libraries are included in the Diffusion installation, which is available from the following location: [Get the iOS libraries from the Diffusion installation](#). Install Diffusion and get the `diffusion-iphones-classic-version.zip` file from the `clients/apple` folder of the installation. For more information, see [Installing the Diffusion server](#) on page 535.

About this task

These instructions have been created using Xcode 6.0.1.

Procedure

1. Extract the contents of the `diffusion-iphones-classic-version.zip` file to your preferred location for third-party SDKs for use within Xcode.

For example, you might have a directory within your `Documents` folder for `code` within which you have a sub-directory for software development kits (SDKs). In this case, locate the iOS SDK for Diffusion in the following directory: `~/Documents/code/SDKs/diffusion-iphones-classic-version/`, where `version` is the version number, for example 5.9.24

2. Create an Xcode project for your Diffusion client.

- a) From the **File** menu, select **New > Project...**

Xcode prompts you to **Choose a template for your new project**.

- b) Select **iOS > Application** on the left.
- c) Select **Single View Application** on the right and click **Next**.

Xcode prompts you to **Choose options for your new project**.

- d) Configure your project appropriately for your requirements.

The Diffusion iOS Classic API does not work with Swift without additional code, so select Objective-C as the **Language**.

For example, use the following values:

- **Product Name:** `TestClient`
- **Language:** `Objective-C`
- **Devices:** `Universal`

- e) Click the **Next** button.

Xcode prompts you to select a destination folder for your new project.

- f) Select a target folder. For example, `~/Documents/code/`, and click **Create**.

3. Import the Diffusion iOS SDK.

Use the Xcode **Build Settings** to define the location of your Diffusion iOS SDK.

- a) Go to the **Build Settings** tab for the Project or Target.
- b) Click the plus sign (+) and select **Add User-Defined Setting**.
- c) Set the name of the user-defined setting to `DIFFUSION_ROOT` and the value to the top-level directory of your extracted Diffusion iOS SDK.

We recommend that you use the Xcode `SRCROOT` property in order to provide a relative location. For example, `$(SRCROOT)/../SDKs/diffusion-iphones-classic-version` defines the location of the Diffusion iOS SDK as the directory given in step 1 on page 467

- d) Go to the **User Header Search Paths** (`USER_HEADER_SEARCH_PATHS`) setting and add the following value: `$(DIFFUSION_ROOT)/headers`

Use the default, non-recursive option.

- e) Go to the **Library Search Paths** (LIBRARY_SEARCH_PATHS) setting and add the following values:

- Debug configuration: \$(DIFFUSION_ROOT)/Debug-universal
- Release configuration: \$(DIFFUSION_ROOT)/Release-universal

Use the default, non-recursive option.

- f) Go to the **Other Linker Flags** (OTHER_LDFLAGS) setting and add the following value:

```
-lDiffusionTransport
```

For more information, see https://developer.apple.com/library/mac/documentation/DeveloperTools/Reference/XcodeBuildSettingRef/1-Build_Setting_Reference/build_setting_ref.html.

4. Add the required system libraries

The Diffusion iOS SDK depends on Zlib and on ICU: International Components for Unicode. These libraries are not included with the linker requirements by default for new Xcode projects so you need to add them.

- a) Go to **Target > Build Phases > Link Binary With Libraries**
- b) Add `libz.dylib`
- c) Add `libicucore.dylib`

The following libraries are included by default for new Xcode projects and are required by the Diffusion iOS SDK:

- `CFNetwork.framework`
- `Foundation.framework`
- `Security.framework`

5. Create a client that connects to the Diffusion server when the view controller loads. (ViewController.m)

```
#import "ViewController.h"
#import "diffusion.h"

@interface ViewController (DFClientDelegate) <DFClientDelegate>
@end

@implementation ViewController
{
    DFClient *_diffusionClient;
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    NSURL *const serverURL = [NSURL URLWithString:@"ws://
diffusion.example.com:80"];
    DFServerDetails *const serverDetails = [[DFServerDetails alloc]
initWithURL:serverURL error:nil];
    DFConnectionDetails *const connectionDetails =
[[DFConnectionDetails alloc] initWithServer:serverDetails

        topics:@"Assets/"

        andCredentials:nil];
    _diffusionClient = [[DFClient alloc] init];
    _diffusionClient.delegate = self;
}
```

```

        [_diffusionClient setConnectionDetails:connectionDetails];
        [_diffusionClient connect];
    }

@end

@implementation ViewController (DFClientDelegate)

-(void)onConnection:(const BOOL)isConnected
{
    NSLog(@"Diffusion %@connected.", (isConnected ? @"" : @"NOT
"));
}

-(void)onMessage:(DFTopicMessage *const)message
{
    NSLog(@"Diffusion message: \"%@\\" = \"%@\\"", message.topic,
message.records[0]);
}

// Implement other required methods
-(void)onAbort { }
-(void)onConnectionSequenceExhausted:(DFClient *const)client { }
-(void)onLostConnection { }
-(void)onMessageNotAcknowledged:(DFTopicMessage *const)message { }
-(void)onPing:(DFPingMessage *const)message { }
-(void)onServerRejectedConnection { }

@end

```

- a) Import the `diffusion.h` header file.
 - This file pulls in the other required header files.
- b) Conform to the `DFClientDelegate` protocol, using a category with the same name to enhance readability.
- c) In the `viewDidLoad` method, assign `serverURL` to point to the Diffusion server using the DPT protocol.
- d) Create a `DFServerDetails` object, `serverDetails`. Use the `initWithURL` method to wrap `serverURL`.
 - Change the URL from that provided in the example to the URL of the Diffusion server.
- e) Create a `DFConnectionDetails` object, `connectionDetails`. Use the `initWithServer` method to include `serverDetails`. Request a default, recursive subscription to the Assets topic.
- f) Define a `_diffusionClient` instance variable.
- g) Assign `self` to the delegate property of `_diffusionClient`.
- h) Use the `setConnectionDetails` method to include `connectionDetails`.
- i) Use the `connect` method to connect `_diffusionClient` to the Diffusion server.
- j) Implement the `DFClientDelegate` category.
- k) In the `DFClientDelegate` implementation, implement `onConnection:` to perform the required actions when the client connects. For example, log that the connection was successful.
- l) In the `DFClientDelegate` implementation, implement `onMessage:` to perform the required actions when a message is received. For example, log the message content.
- m) In the `DFClientDelegate` implementation, implement the other required methods.
 - `onAbort:`
 - `onConnectionSequenceExhausted:`
 - `onLostConnection:`
 - `onMessageNotAcknowledged:`

- onPing:
- onServerRejectedConnection:

These implementations can be empty.

Results

The client connects to the Diffusion server. It receives a callback from the Diffusion iOS SDK through the `onConnection:` implementation. When connected the client receives topic messages (both for initial topic load and deltas) from the Diffusion iOS SDK through the `onMessage:` implementation.

Using the iOS Classic API

There are features, issues, and considerations that are specific to clients that are implemented using the iOS Classic API.

Diffusion Delegate

The Diffusion Delegate class is a custom class that must adhere to the `DFClientDelegate` protocol. The protocol consists of the following methods

```
/**
 * Protocol implemented by classes wishing to receive notifications.
 * Notification primarily of new messages and the state of the
 * connection to the server.
 */
@protocol DFClientDelegate

/**
 * This method is called when the DFClient tries to connect, if the
 * connection is made, isConnected is true
 * @param isConnected
 */
- (void) onConnection:(BOOL) isConnected;

/**
 * This method is called when the DFClient has lost connection
 */
- (void) onLostConnection;

/**
 * This method is called when the Diffusion server has terminated the
 * connection (barred)
 */
- (void) onAbort;

/**
 * This method is called when a message has been received from the
 * Diffusion server.
 * This method is called as well as any topicListeners that might
 * match the topic.
 */
- (void) onMessage:(DFTopicMessage *) message;

/**
 * This method is called on receipt of the ping request
 * @see DFClient
 * @param message PingMessage
 */
- (void) onPing:(DFPingMessage *) message;

/**
```

```

* This method is called after a send credentials message, and the
server rejected the credentials
* @see DFClient
*/
- (void) onServerRejectedConnection;

/**
* This method is called if the server did not respond to an Ack
message in time
* @see TopicMessage
*/
- (void) onMessageNotAcknowledged:(DFTopicMessage *) message;

/**
The list of DFServerDetails object has been exhausted, and no
connection can be placed.
Once this method is called the set of DFServerDetails is reset and
further connections can be placed. In most simple scenarios where
there is only one DFServerDetails object in the DFConnectionDetails
object call method [client connect] here.
@param client DFClient that has exhausted its set of DFServerDetails
object from the DFClientDetails object.
*/
-(void)onConnectionSequenceExhausted:(DFClient*)client;

@optional

/**
Conveys news from the Diffusion server that the named topic no
longer exists
*/
-(void)onTopicRemoved:(NSString*) topicName;

/**
The given DFServerDetails object has been selected for connection.
@param details Details object that has been chosen.
@param client DFClient that has chosen this DFServerDetails
*/
-(void)onConnectionDetailsAcquired:(DFServerDetails*)details
forClient:(DFClient*)client;

```

You can receive an `onConnection` event after you have successfully connected, this might be because of a lost connection.

Credentials

When credentials are required, use the `credentials` property on the `DFClient` class. Create a `DFCredentials` class and set it on the client before you call `connect`.

onMessage event

When messages arrive from the Diffusion server on a subscribed topic, the `onMessage` method is called on the delegate provided. The message is wrapped in a class called `TopicMessage`. This class contains helper methods that surround the message itself, such as the `topic` and `isInitialLoad` properties. For more information, see [iOS Classic API documentation](#).

Subscriptions

Once the client has connected, you can issue `subscribe` and `unsubscribe` commands. The `subscribe` and `unsubscribe` methods take a string format, that can be a topic selection pattern, a list of topics that are comma delimited, or a single topic.

Send

Once connected, you can send messages to the Diffusion server on a particular topic path. To do this, use the `send` method.

```
[mClient send:"Fred" : "Hello Fred"];
```

In the example above, the registered message handler for topic Fred receives a `messageFromClient` notification. If you want to send a message with user headers, use the `sendTopicMessage` method. A `TopicMessage` enables you to set user headers.

Ping

You can ping the Diffusion server. The delegate is notified by the `onPing` method. The resulting ping event has two attributes in it, firstly the time stamp of the request. The second attribute is how many items are currently in the client queue at the server. This information enables the client to get some vital connection information. It is down to the implementation of the client to specify the ping frequency, if at all required.

Topic listeners

During the lifetime of the connection, it might be required to have modular components notified about topic messages – these are topic listeners. A topic listener calls a supplied method with a `TopicMessage` class when the topic path of the message matches the topic name.

Note: For performance the iOS topic listeners do not have regular expression patterns but topic name matching.

You can have many topic listeners on the same topic pattern if required. For example, if you want to be notified about a particular topic, issue the following listener:

```
[mClient addTopicListener:aTopicListener];
```

Where a topic listener implements the protocol `DFTopicListenerDelegate` which is shown below.

```
/**
 * Protocol for receiving messages from a particular topic.
 */
@protocol DFTopicListenerDelegate

/**
 * This method is called if the TopicMessage matches the message
 * received from Diffusion
 *
 * @param message
 * @return YES if the message is consumed and must not be relayed to
 * subsequent DFTopicListenerDelegate, nor the default listener.
 */
- (BOOL) onMessage:(DFTopicMessage *) message;

/**
 * This is the topic used to see if the message from Diffusion
 * matches (equals) this String
 */
- (NSString *) getTopic;
```

Topic listeners can be removed by calling the `removeTopicListener` method on the `DFClient` class.

iOS Classic API examples

Examples that use the iOS Classic API.

F1 Steering Wheel demo

The F1 Steering Wheel demo is available on Github: <https://github.com/pushtechology/blog-steering-wheel/>. This demo uses the iOS Classic API to create a display client that shows the realtime input from a driving game controller.

DEPRECATED: Android Classic API

The Android API is bundled in a library called `diffusion-android-version.jar`, where *version* is the version number, for example 5.9.24.

The Android library is provided with the Diffusion server installation in the `clients/android` folder.

`DiffusionClient` class is the main class that is used. `ConnectionDetails` and the `ServerDetails` classes enables the user to set all of the connection and topic information.

The `DiffusionClient` uses a listener model where the `DiffusionConnectionListener` interface is responsible for all of the event notifications from `DiffusionClient`

Capabilities

To see the full list of capabilities supported by the Android Classic API, see .

Support

Table 42:

--	--

Getting started with Android Classic API

Create a client application within minutes that connects to the Diffusion server.

About this task

The example demonstrates an empty Android client that you can base your clients on.

Procedure

1. Download the Android SDK and the Diffusion Android library.
2. Create an Android project that references the `diffusion-android-version.jar` library, where *version* is the version number, for example 5.9.24.
3. In your project, create a Java file that extends the `android.app.Activity` class.

```
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
```

```

import android.widget.TextView;

import com.pushtechonology.android.diffusion.DiffusionClient;
import com.pushtechonology.mobile.APIException;
import com.pushtechonology.mobile.ConnectionDetails;
import com.pushtechonology.mobile.DiffusionConnectionListener;
import com.pushtechonology.mobile.DiffusionTopicStatusListener;
import com.pushtechonology.mobile.MalformedURLException;
import com.pushtechonology.mobile.Message;
import com.pushtechonology.mobile.PingMessage;
import com.pushtechonology.mobile.ServerDetails;
import com.pushtechonology.mobile.ServiceTopicError;
import com.pushtechonology.mobile.ServiceTopicHandler;
import com.pushtechonology.mobile.ServiceTopicListener;
import com.pushtechonology.mobile.ServiceTopicResponse;
import com.pushtechonology.mobile.TopicListener;
import com.pushtechonology.mobile.TopicMessage;
import com.pushtechonology.mobile.URL;
import com.pushtechonology.mobile.enums.EncodingValue;

public class DemoClient extends Activity implements
    DiffusionConnectionListener, DiffusionTopicStatusListener {

    private DiffusionClient theClient;
    static final String TAG = "Diffusion Client";
    private static final String SERVICE_TOPIC = "SERVICE";

    ConnectionDetails cnxDetails;
    {
        try {
            ServerDetails svrDetailsArr[] = new ServerDetails[] {
                new ServerDetails( new URL( "URL:port" ) ),
            };
            cnxDetails = new ConnectionDetails( svrDetailsArr );
            cnxDetails.setTopics( SERVICE_TOPIC );
            cnxDetails.setCascade( true );
            cnxDetails.setAutoFailover( true );
        } catch (MalformedURLException ex) {
            writeln( ex.toString() );
        }
    }
    ServerDetails currentServerDetails;

    // Set Diffusion connection details, and place the Diffusion
    connection
    private void connectToServer()
    {
        theClient = new DiffusionClient();
        theClient.setConnectionDetails( cnxDetails );
        theClient.addTopicListener(new TopicListener());
        theClient.setConnectionListener(this);
        theClient.setTopicStatusListener( this );
        theClient.connect();
        statusText.setText( String.format( "Connecting to %s",
            currentServerDetails.getUrl() ) );
    }

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {

```

```
super.onCreate(savedInstanceState);  
  
// Set up the UI of your Android app  
connectToServer();  
}  
}
```

- a) Import `com.pushtechonology.android.diffusion.DiffusionClient` and the classes in the `com.pushtechonology.mobile` package.
- b) Implement `DiffusionConnectionListener` and `DiffusionTopicStatusListener`.
- c) Create a `ConnectionDetails` object that includes the connections details and settings that you require.
- d) Create a `connectToServer()` method that uses the `ConnectionDetails` object to set the connections details and places a connection to Diffusion.
- e) Override the `onCreate()` method inherited from `Activity`. In this method call `connectToServer()` to place the Diffusion connection when the application starts.
- f) Use the `onCreate()` method to set up the views and content of your Android application.

Related information

<http://developer.android.com/training/index.html>

Using the Android Classic API

There are features, issues, and considerations that are specific to clients that are implemented using the Android Classic API.

Credentials

When credentials are required, there are three ways to set the credentials. The `ServerDetails`, the `ConnectionDetails` and the `DiffusionClient` all have a `setCredentials` method. It is required that the user create a `DiffusionCredentials` object and pass it to one of these methods before calling `connect`. Use only one of these ways. If more than one way is used, the selection of the credentials to use is undefined. The Android Classic API only supports sending credentials on connection.

onMessage event

When messages arrive from the Diffusion server on a subscribed topic, the `onMessage` is called on the delegate provided. The message is wrapped in a interface called `Message`. This interface contains helper methods that surround the message itself, like `getTopic()` and `isInitialTopicLoad`. For more information, see [Android Unified API documentation](#).

Subscriptions

Once the client has connected, you can issue subscribe and unsubscribe commands. The `subscribe` and `unsubscribe` methods take a string format, that can be a topic selector pattern, a list of topics that are comma delimited, or a single topic.

Send

Once connected a client can send messages to the Diffusion server on a particular topic. To do this, use the `send` method.

```
theClient.send("Fred", "Hello Fred");
```

In the example above, the registered message handler of the topic Fred receives a `messageFromClient` notification. If the message requires a user header, use the `sendTopicMessage` method. A `TopicMessage` allows for the setting of user headers.

```
TopicMessage message = new TopicMessage("Fred");
message.addUserHeader("myHeaders");
message.setMessage("Hello Fred");
theClient.sendTopicMessage(message);
```

Ping

The client can ping the Diffusion server. The delegate is notified of the response by the `onPing` function. The resulting ping event has two attributes in it:

- the timestamp of the request
- the number of items in the client queue

This information enables the client to get some vital connection information. It is down to the implementation of the client to specify the ping frequency, if at all required.

Topic listeners

During the life time of the connection, it might be required to have modular components that get notified about topic messages, these are topic listeners. A topic listener is called using its `onMessage` method with a message object when the topic of the message matches the topic name.

Note: For performance the Android topic listeners do not have regular expression patterns but topic name matching.

You can have many topic listeners on the same topic path if required. For example, if you want to be notified about a particular topic, use the following code:

```
theClient.addTopicListener(topicListener);
```

Where `topicListener` implements the interface `TopicListener`. See the following example.

```
/**
 * getTopic
 *
 * @return the topic that this listener is interested in. This
does
 * not take regular expressions this must be an exact match
 */
String getTopic();

/**
 * onMessage
 *
 * @param message message which topic matches the getTopic method
 */
void onMessage(Message message);
```

Remove topic listeners by calling the `removeTopicListener` method on the `DiffusionClient` class.

Threading concerns

The `DiffusionClient` creates and dedicates a thread to listening to traffic from the Diffusion server and reacting to messages from it. Consequently methods on the `DiffusionConnectionListener` and `DiffusionTopicStatusListener` are executed in the same thread. Android does not allow background threads to interact with GUI controls, only the main thread is allowed to do so.

To overcome this, any non-main thread can pass a `java.lang.Runnable` to the main thread for execution via `android.view.View.post(Runnable action)`. For example:

```
/*
 * Called when the Diffusion connection is established
 */
public void connected() {
    // Post this Runnable to the GUI thread to change the display
    String statusTextStr = String.format( "Connected to %s:%d\n",
    HOST, PORT );
    setStatus( statusTextStr );
    Log.i( TAG, statusTextStr );
}

/**
 * Set the content of the status view
 * @param statusStr
 */
private void setStatus(final String statusStr)
{
    // Pass a Runnable to the GUI thread to execute using one of its
    // widgets
    outputView.post( new Runnable() {
        public void run() {
            statusText.setText( statusStr );
        }
    } );
}
```

User permissions in Manifest.xml

To establish a connection with the Diffusion server, Android devices must add the user permission `INTERNET` within the `Manifest.xml`. This permission allows applications to open network sockets.

Android Classic API examples

Examples that use the Android Classic API.

DiffusionClient

The following code shows an example of connection:

```
// Get a new DiffusionClient
theClient = new DiffusionClient();

//Set the connection details
ServerDetails serverDetails = new ServerDetails(new URL("dpt://
diffusion.example.com:80"));
```

```

    ConnectionDetails connectionDetails = new
    ConnectionDetails(serverDetails);
    theClient.setConnectionDetails(connectionDetails);

    // Make this listen to the DiffusionClient events
    theClient.setConnectionListener(this);

    // Connect
    theClient.connect();

```

DiffusionConnectionListener

The DiffusionConnectionListener interface consists of the following methods (for further information refer to the API documentation)

```

/**
 * connected, called upon connection
 */
void connected();

/**
 * errorConnecting, called if there is an error connecting
 *
 * @param e
 */
void errorConnecting(Exception e);

/**
 * disconnected, called when the connection list lost
 */
void disconnected();

/**
 * connectionAborted, called when DiffusionServer has rejected
the connection
 */
void connectionAborted();

/**
 * onMessage, called when a message has been received from
Diffusion
 *
 * @param message
 */
void onMessage(Message message);

/**
 * onPingMessage, called when a ping response is received
 *
 * @param message
 */
void onPingMessage(PingMessage message);

/**
 * onMessageNotAcknowledged, called when an ack message has not
been acknowledged by Diffusion
 *
 * @param message
 */
public void onMessageNotAcknowledged(TopicMessage message);

```

```

/**
 * onConnectionSequenceExhausted, called when the complete list
 of ServerDetails have been exhausted.
 */
public void onConnectionSequenceExhausted();

/**
 * onConnectionDetailsAcquired, called each time a ServerDetails
 object is selected for connection.
 *
 * @param serverDetails
 */
public void onConnectionDetailsAcquired(ServerDetails
serverDetails);

/**
 * onServerRejectedCredentials, called when Diffusion reject the
 credentials.
 */
public void onServerRejectedCredentials();

```

Change the URL from that provided in the example to the URL of the Diffusion server.

DEPRECATED: C Classic API

The C Classic API is provided as a source distribution in the file `diffusion-c-classic-version.zip`, where `version` is the version number, for example 5.9.24. This file is located in the `clients/c` directory of your Diffusion installation.

Note: This API is deprecated and will be removed in the next release.

The source builds to either a dynamic or shared library on UNIX systems and is supported on Red Hat 7.2 and CentOS 7.2. You can link it into your own C/C++ applications, or used as the foundation for creating Diffusion clients for other languages which can be extended through binary APIs.

For full API documentation, see [C Classic API documentation](#)

Using the C Classic API

Build the library using the make command and ensure that it is on your `LD_LIBRARY_PATH`.

Building

To build the library, type `make` in the source directory. This builds shared and static versions of the Diffusion library, `libdiffusion.so` and `libdiffusion.a` respectively. Additionally, a number of sample applications are also built.

Installation

Copy the libraries to a location on the user's `LD_LIBRARY_PATH` (or equivalent). Copy the header files, `diffusion.h` and `l1ist.h` to your C compiler's include path.

For example:

```

$ mkdir /usr/local/include/diffusion
$ mkdir /usr/local/lib/diffusion
$ cp include/*.h /usr/local/include/diffusion/
$ cp lib/libdiffusion.* /usr/local/lib/diffusion/
$ echo "/usr/local/lib/diffusion" > /etc/ld.so.conf.d/
libdiffusion.conf

```

```
$ ldconfig
```

Example usage

The following example shows how to connect to a Diffusion instance (no credentials):

```
DIFFUSION_CONNECTION *c = diff_connect("localhost", 8080, NULL);
if(c == NULL) {
    fprintf(stderr, "Failed to connect to Diffusion\n");
    return(-1);
}
```

The following example shows how to connect to a Diffusion instance (with credentials)

```
SECURITY_CREDENTIALS creds;
creds.username = strdup("smith");
creds.password = strdup("secret");
DIFFUSION_CONNECTION *c = diff_connect("localhost", 8080, &creds);
if(c == NULL) {
    fprintf(stderr, "Failed to connect to Diffusion\n");
    return(-1);
}
```

The following example shows how to request a subscription to a topic:

```
DIFFUSION_CONNECTION *c = diff_connect(...);

if(diff_subscribe(c, "Assets") == -1) {
    fprintf(stderr, "Failed to subscribe to topic\n");
    return(-1);
}
```

The following example shows how to use the event loop and callbacks:

```
void on_initial_load(DIFFUSION_MESSAGE *msg) {...}
void on_delta(DIFFUSION_MESSAGE *msg) {...}

...

DIFFUSION_CONNECTION *c = diff_connect(...);
diff_subscribe(...);

DIFFUSION_CALLBACKS callbacks;
DIFF_CB_ZERO(callbacks); // Reset callback structure
callbacks.on_initial_load = &on_initial_load;
callbacks.on_delta = &on_delta;

diff_loop(c, &callbacks);
```

diffusion-wrapper.js

The Diffusion wrapper is a script which addresses a weakness in the Flash and Silverlight VMs – when running inside a web browser there is no provision for the execution of callback code when the user closes either the containing tab or the entire browser window.

Consequently there is no opportunity for the Diffusion client to inform the server that the client is willingly closing the connection, instead the connection is severed. This can result in various server warnings (dependent on the transport mechanism, that is DPT or HTTP) as well as maintaining the server-side reference to the client if any reconnection timeout is specified.

The JavaScript environment in the hosting browser provides browser closing callbacks, and these are employed by DiffusionWrapper.js. By supplying the `setClientDetails` function with client and server info after a connection has been established within the Flash or Silverlight API, DiffusionWrapper attaches a JavaScript function to the `onbeforeunload` event that is triggered when a browser window or tab closes. This notifies Diffusion that the client is deliberately closing.

`diffusion-wrapper.js` can be found in `clients/flex` and `clients/silverlight` directories from a default installation.

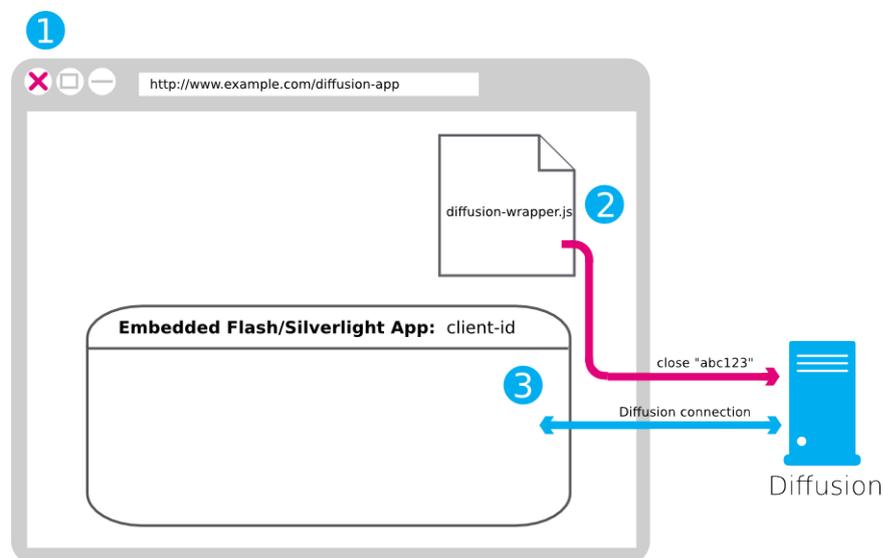


Figure 25: Diffusion wrapper

The preceding diagram shows `diffusion-wrapper.js` in use. When the user closes the containing browser tab or window, the following events occur:

1. The user closes the browser. This engages a JavaScript callback that calls `diffusion-wrapper.js`.
2. `diffusion-wrapper.js` runs and sends a closure request to the Diffusion server.
3. The Flash or Silverlight client dies. It has no chance to run cleanup code.

How to use Diffusion wrapper

The HTML page that loads the Flash/Silverlight app, must to load the Diffusion wrapper script.

The following example shows the `diffusion-wrapper.js` file being included in a web page:

```
<html>
  <head>
    <script src="diffusion-wrapper.js" language="javascript"></script>
  </head>
```

```

<body>
. . . .
</body>
</html>

```

The Flash/Silverlight app makes a call to the external DiffusionWrapper method `setClientDetails`, typically in the `onConnection` callback, supplying the client ID and server URL. In the example below, the method `ExternalInterface.call("setClientDetails")` provides DiffusionWrapper with the client ID and server URL. DiffusionWrapper adds a method to the `onbeforeunload` event of the window, which informs the Diffusion server that the client is intentionally closing the connection when the browser window or tab is closed.

```

<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                width="371" height="334" minWidth="955"
                minHeight="600" layout="absolute">
<mx:Script>
import com.pushtechology.diffusion.ConnectionDetails;
import com.pushtechology.diffusion.DiffusionClient;
import com.pushtechology.diffusion.ServerDetails;
import com.pushtechology.diffusion.events.DiffusionConnectionEvent;
import com.pushtechology.diffusion.events.DiffusionExceptionEvent;
import com.pushtechology.diffusion.events.DiffusionMessageEvent;

private var theServerUrl:String = "http://127.0.0.1:8080"
private var theInitialTopic:String = "Echo"
private var theClient:DiffusionClient;

private function onConnect():void{
    theClient = new DiffusionClient();
    var theConnectionDetails:ConnectionDetails = new
ConnectionDetails(new ServerDetails(theServerUrl),null);

    // Add the listeners
    theClient.addEventListener(DiffusionConnectionEvent.CONNECTION,
onConnection);
    theClient.addEventListener(DiffusionMessageEvent.MESSAGE,
onMessages);
    theClient.addEventListener(DiffusionExceptionEvent.EXCEPTION,
onException);

    //Lets Go...
    theClient.connect(theConnectionDetails);
}

private function onConnection(event:DiffusionConnectionEvent):void{
    // Is the client connected?
    if(event.isConnected()){
        //Check the externalInterface availability
        if(ExternalInterface.available){
            // Send the ClientId and serverURL to the DiffusionWrapper
            ExternalInterface.call("setClientDetails",
theClient.getClientID(), event.getServerDetails().getURL());
        }else{
            //The externalInterface is not available, so the
DiffusionWrapper is not called.
        }
    }
}

private function onMessages(event:DiffusionMessageEvent):void{

```

```

//Message received
    theMessages.text += event.toString();
}

private function onException(event:DiffusionExceptionEvent):void{

    //Exception received
    theMessages.text += event.toString();
}

</mx:Script>
<mx:TextArea id="theMessages" x="18" y="40" width="335" height="283"
text="Messages"/>
<mx:Button id="bConnect" x="105.5" y="10" width="160"
label="Connect" click="onConnect()"/>
</mx:Application>

```

Developing a publisher

You can develop a publisher in Java by using the Publisher API.

Note: We recommend using a client to create and publish to topics, instead of a publisher. The ability to create and publish to topics is available in the following client APIs:

- [Java](#) on page 160
- [.NET](#) on page 162
- [C](#) on page 164
- [Apple](#) on page 156
- [Android](#) on page 158
- [JavaScript](#) on page 153

Publisher basics

A publisher is a user-defined object deployed within a Diffusion server which provides one or more topics on which it publishes messages to clients.

There can be one or more publishers deployed with a Diffusion server.

Clients connect to the server and subscribe to topics. Messages relate to topics and when a publisher publishes a message it is broadcast to all clients that are currently subscribed to the message topic. A publisher can also send messages to individual clients and receive messages sent from clients. Clients can request (fetch) topic state, even when not subscribed.

A publisher must be written by the user in Java (utilizing the publisher API) and deployed within the server. This is done by extending a supplied publisher class and implementing methods as required. Implement the methods relating to the functionality that you require. For more information, see [Writing a publisher](#) on page 492.

Defining publishers

How to define publishers that start with Diffusion.

The Diffusion server is able to start the publishers defined in the `etc/Publishers.xml` file when the server starts. The XML file can contain any number of publishers. Each publisher must have at least a name and a class. The class must implement the publisher by extending the `Publisher` class. For more information, see [Creating a Publisher class](#) on page 492.

```
<publishers>
  <publisher name="Publisher">
    <class>com.example.Publisher</class>
  </publisher>
  ...
</publishers>
```

The name must be unique on the server, and the class must exist on the classpath of the Diffusion server (For more information, see [Classic deployment](#) on page 640). This is sufficient for the publisher to start when Diffusion does. There are other options, including those that can prevent the publisher from starting.

When the `enabled` element is false, the publisher class is not loaded. If the `start` element is false, the publisher is not started when the server starts. If the `topic-aliasing` element is false, topic aliases is not used by topic messages. The following example shows the default values for these optional settings.

```
<publishers>
  <publisher name="Publisher">
    <class>com.example.Publisher</class>
    <enabled>true</enabled>
    <start>true</start>
    <topic-aliasing>true</topic-aliasing>
  </publisher>
</publishers>
```

For more information, see .

You can define properties in the `etc/Publishers.xml` that can be accessed from the publisher. For more information, see .

DEPRECATED: Server connections can be configured in the `etc/Publishers.xml` for connecting to other publishers.

The full configuration file options can be found in the XSD document for the `etc/Publishers.xml` or in [Publishers.xml](#) on page 606.

Loading publisher code

This describes how to load publisher classes or code it is dependent upon.

When you write a publisher class (or any other classes it uses), you can deploy them in any folder as long as it is specified in the configuration (`usr-lib` in `etc/Server.xml`). JAR files can also be deployed in user libraries and any other software libraries that the publisher requires can be specified in this way.

Also, when Diffusion starts, the `data` directory is on the class path. The `ext` folder, and its sub-directories are scanned for jar files and class loaded. This means that you can easily add new jars to the Diffusion runtime, without having to edit the startup scripts.

Take care when creating backup jars in the `ext` folder as anything that ends in `.jar` is class loaded.

Related tasks

[Building a publisher with mvndar](#) on page 516

Use the Maven plugin *mvndar* to build and deploy your publisher DAR file. This plugin is available from the Push Public Maven Repository.

Load publishers by using the API

You can configure and load custom publishers using the Diffusion API at any point in the Diffusion server's lifecycle.

Similarly to loading publishers using configuration files, each publisher must have at least a name and a class. The class must implement the publisher by extending the `Publisher` class. For more information, see [Creating a Publisher class](#) on page 492.

```
PublisherConfig config =
    ConfigManager.getServerConfig().addPublisher("MyPublisher",
        "com.acme.foo.MyPublisher");
Publisher publisher = Publishers.loadPublisher(config);
```

The name must be unique on the server, and the class must exist on the classpath of the Diffusion server. For more information, see [Classic deployment](#) on page 640. By default the `autostart` property is enabled on the `PublisherConfig`, so the publisher starts once it is loaded. If this option is disabled, you can load a publisher and retain a reference to it, to start at a later point in time.

If the default configuration options are suitable for your requirements (as detailed within the API docs for `com.pushtechology.diffusion.api.config.PublisherConfig`) there are several convenience methods that can be used to load a given publisher and get a reference to it without the need for construction a specific `PublisherConfig` instance.

```
// Create Publisher with classname
Publisher publisher = Publishers.createPublisher("MyPublisher",
    "com.acme.foo.MyPublisher");

// Create Publisher with Class
Publisher publisher = Publishers.createPublisher("MyPublisher",
    MyPublisher.class);
```

You can load a default publisher instance. This facilitates programmatic access any features exposed through the publisher abstract class that do not require method overriding.

```
Publisher publisher =
    Publishers.createPublisher("MyDefaultPublisher");
```

Starting and stopping publishers

Typically publishers are started when the server starts but you can prevent such automatic start up and allow publishers to be started using System Management.

Publishers can also be stopped and restarted using System Management functions and are automatically stopped and removed when the server closes.

In order for a publisher to function properly on being stopped and restarted from System Management it must be able to cater for the integrity of its data and client connections. For this reason a publisher cannot be stopped by default and must override the `isStoppable` method to enable this functionality.

Publisher startup steps

When a publisher is started it goes through its initial processing in the order shown below:

Table 43: Start publisher

Add initial topics	Initial topics configured for the publisher are added.
Load server connections	Server connections configured for the publisher are loaded and validated.
<code>initialLoad</code>	The <code>initialLoad</code> notification method is called. This can be used to perform any initial processing required for the publisher. Topics can be added here. Other aspects of the publisher, such as topic loaders and client listeners can also be set up here. If an exception is thrown by this method, the publisher fails to start.
Connect to servers	A connection is made to each server connection.
STARTED	At this point the publisher is considered to have started.
<code>publisherStarted</code>	The <code>publisherStarted</code> notification method is called.

Publisher closedown steps

When a publisher is stopped, either during server closedown or by System Management it goes through the following steps:

Table 44: Stop publisher

<code>publisherStopping</code>	The <code>publisherStopping</code> notification method is called to allow the publisher to perform any preliminary close processing.
Remove topics	All topics owned by the publisher are removed.
Close server connections	Any server connections made by the publisher are closed.
STOPPED	At this point the publisher is considered to be stopped.
<code>0publisherStopped</code>	The <code>publisherStopped</code> notification method is called.
Client events Stopped	Client event notifications are stopped.

Publisher removal

A publisher is removed after it is stopped during server closedown but you can also remove a stopped publisher at any time using System Management. Once removed a publisher cannot be restarted again until the server is restarted.

In either case, after removal the `publisherRemoved` notification method is called.

Publisher topics

Topics are the mechanism by which publishers provide data to clients.

Each publisher can provide one or more topics but each topic must be unique by name within the server. Topics are hierarchical in nature and so topics can be parents of topics and a tree of topics can be set up. Using hierarchies allows clients to subscribe to branches of the hierarchy rather than having to subscribe to individual topics. Only the owner of a topic can create new topics below it in the hierarchy.

Adding topics

In the simplest case a publisher can name the topics it provides within its configuration. In this case such topics are automatically added as the publisher is started. These topics can be obtained from within the publisher using the `getInitialTopicSet` method.

More typically a publisher adds the topics it requires itself as it starts up. A Publisher can choose to add some topics at start up and others later. Topics can be added at any time using the publisher's `addTopic` or `addTopics` method. They can be added only if they are added by the owner of the parent topic.

A topic can be a simple topic where all of the handling of the topic state is provided by the publisher. Alternatively a topic can be created with topic data which handles the state of the topic automatically.

As soon as a topic has been added clients can subscribe to it.

Loading topics

Simple topic processing involves sending all of the data that defines a topic (the topic load) to a client when they first subscribe and then subsequently sending deltas (or changes to the data). There are two mechanisms for performing the topic load:

Send on subscribe

When the publisher is notified of subscription it creates, populates and sends a topic load message to the client.

Topic loaders

Define a topic loader for the topic which is automatically called to perform the topic loading when a client subscribes.

If a topic has topic data, the current state is automatically provided to a client when they subscribe.

Subscribing clients to topics

Clients normally request subscription to a topic and if the topic exists the clients become subscribed to it at that point.

A client can subscribe to a topic that does not exist at that time – this is called pre-emptive subscription. When the publisher creates a topic, any clients that have pre-emptively subscribed to a topic are subscribed to that topic automatically.

A publisher can also force all currently connected clients to become subscribed to a topic by calling `subscribeClients` with `force=true`.

Subscribing clients to topics that they were already subscribed to causes the topic load to be performed again.

A publisher can also cause individual clients to be subscribed to a topic using the client's `subscribe` method or unsubscribed using the `unsubscribe` method.

Providing topic state

The publisher method `fetchForClient` must be implemented if clients are to obtain state using the topic fetch feature.

Handling topics that do not exist

A topic is an entity that notionally has state but in some circumstances a client might request access to a topic that does not exist. Client notifications provide a mechanism whereby this situation can be handled.

Where a client attempts to subscribe to a topic that does not exist, a `clientSubscriptionInvalid` notification occurs which gives the publisher the opportunity to dynamically create the topic (and subscribe the client to it) if that is what is required.

Where a client attempts to fetch the state of a topic that does not exist, a `clientFetchInvalid` notification occurs which gives the publisher the opportunity to return a response to the fetch request (using `sendFetchReply`) even if the topic does not exist. This can provide an efficient request/response mechanism without the overhead of actually creating topics.

If a Classic API client attempts to send a message to a topic that does not exist, a `clientSendInvalid` notification occurs, allowing the publisher to either create the topic, process the message as stand-alone data, or discard the message as appropriate.

Removing topics

A publisher can also remove topics at any time using its `removeTopic` or `removeTopics` methods.

Removing a topic causes all clients that are subscribed to it to be unsubscribed.

Receiving and maintaining data

A publisher can obtain the data it is to publish and transform that data in any way that is appropriate.

The publisher maintains the state of its own data by updating it whenever any changes are received so that as a new client subscribes it can be sent the latest state of the data as a whole. As such changes are received they are also published as deltas to all currently subscribed clients.

Receiving messages from a remote service

Remote service can also provided a data feed into a publisher. The remote service can publish to topics and the updates are applied to the topics and passed onto subscribed clients.

Publishing and sending messages

Publishing messages to clients and sending messages to clients

Creating messages

Messages can be created using the factory methods on the publisher or on a topic for creating messages (called `createLoadMessage` and `createDeltaMessage`).

If within a class that does not have a direct reference to the publisher or topic objects, the equivalent static methods in the `Publishers` class can be used. Messages can be populated with data using the many variants of the `put` method.

Publishing messages

Messages (whether load or delta) can be sent to all clients that are subscribed to the message topic. For stateless topics, use `Topic.publishMessage()`. For stateful topics (those with topic data), use `PublishingTopicData.publishMessage()`.

Messages can be sent to an identified group of clients using client groups.

Exclusive publishing

You might want to publish a message to all but a particular client. For example, a message can be sent to the publisher from a client and the publisher can, publish the message to all of the other subscribed clients.

This is done using the publisher's `publishExclusiveMessage` method. This facility also exists on client groups.

Sending messages to individual clients

To send a message to an individual client the `Client.send` method can be used.

To send a message to a group of clients the `ClientGroup.send` method can be used.

Publisher notifications

A publisher is notified of certain events by certain methods on it being called. These methods can be overridden by the user to perform processing at these points as required.

By default these methods (other than those indicated) perform no processing. You do not have to override any of these methods unless you choose to. The notification methods are:

Table 45: Notification methods

<code>initialLoad</code>	Called when the publisher is first loaded. Is typically overridden to perform any initial processing required to prepare the publisher.
<code>publisherStarted</code>	Called after <code>initialLoad</code> (see startup steps).
<code>subscription</code>	Called when a client subscribes to a topic that the publisher owns. References to the topic and the client are passed and also a flag to indicate if the topic has already been loaded by a <code>TopicLoader</code> . If the topic has not been loaded already, typically a publisher sends an initial load message to the client at this point. It might not be necessary to override this method if topic loaders are in use.
<code>unsubscription</code>	Called when a client unsubscribes from a topic that the publisher owns.
<code>messageFromClient</code>	Called when a message is received from a client. References to the message and the client are passed.
<code>messageFromServer</code>	Called when a message is received from a server connection. References to the message and the server connection are passed.
<code>fetchForClient</code>	Called when a client requests a fetch of the topic state for stateless topics.
<code>messageNotAcknowledged</code>	Called when a message which required acknowledgment was sent by the publisher and was not acknowledged by one or more clients within the given timeout period.
<code>serverConnected</code>	This is called when a server connection is made. A reference to the server connection is passed.
<code>serverTopicStatusChanged</code>	This is called when a topic subscribed to at a <code>ServerConnection</code> has its status changed (for example, is removed). The topic name and reference to the server connection is passed.
<code>serverDisconnected</code>	This is called when a <code>ServerConnection</code> is lost. A reference to the server connection is passed.

<code>publisherStopping</code>	This is called when the publisher has been requested to stop. It gives the publisher the opportunity to tidily perform any close processing.
<code>publisherStopped</code>	This is called after a publisher has stopped. The publisher can still be restarted (but only if <code>isStoppable</code> is true).
<code>publisherRemoved</code>	This is called when a publisher is removed and provides the opportunity for final tidy up. The publisher cannot be restarted after this is called.
<code>systemStarted</code>	This is called when the Diffusion system has completed loading and is ready to accept connections. Publishers are started before connectors, so this notification is used all Diffusion sub systems are loaded.

Publisher notification threads

To understand issues of concurrency when writing a publisher it is necessary to understand in which threads the various publisher notifications occur.

When a message or request is received from a client (of any sort) or server connection, the inbound thread pool is used to process it. Depending upon the number of threads in the pool this can mean that the publisher can receive such notifications concurrently.

Message acknowledgment notifications use the background thread pool.

Other notifications come from various control threads.

All of the above considerations mean that concurrency must always be taken into account in publisher code and it must be made thread safe as appropriate.

Client handling

A publisher can receive notifications about and perform actions on individual clients.

Closing/Aborting clients

A publisher can close a client at any time using the `close` method. This disconnects the client which might choose to subsequently reconnect.

Alternatively a publisher can use the `abort` method, which sends an abort notification to the client before disconnecting it. A client receiving an abort notification must not attempt to reconnect.

Client notifications

A publisher can choose to receive additional client notifications so that it can be informed when clients connect, disconnect etc.

Client pings

A client ping message is one that can be sent to a client which reflects it back to the server to measure latency. A publisher can send a ping message to a client using the `Client.ping` method and receives a response on the `ClientListener.clientPingResponse` method within which the message passed can be queried to establish the round trip time.

Client message filtering

You can filter the messages that get queued for any particular client. For more information, see .

Publisher properties

Properties for a publisher are defined in the `etc/Publishers.xml` configuration file.

As well as the standard properties a publisher can have user-defined properties. These properties can be read using convenience methods available on the publisher (for example, `getProperty`, `getIntegerProperty` etc).

Using concurrent threads

Often within a publisher you might have to initiate some processing in a separate thread so that the publisher itself is not blocked.

For example, a thread can be used to poll data from some data source.

Diffusion provides a mechanism for easily managing concurrent processing using the threads API.

Publisher logging

Every publisher is assigned its own Logger which can be used within the publisher itself for logging diagnostic messages.

This Logger is obtained using the `getLogger` method.

The log level of the publisher can be changed dynamically at any time using the `setLogLevel` method.

DEPRECATED: Server connections

Connecting to other Diffusion servers from within a publisher

Publishers can act as clients of other publishers for the purpose of distributed processing. In this case there is a client/server relationship between two publishers. One publisher can be a client of many other publishers and a publisher can have many publisher clients.

A publisher acting as the server in such a relationship sees the client as a normal client. The only thing distinguishing it is its client type (obtained using `Client.getClientType`).

However, for a publisher to act as the client of another publisher it must make an outbound connection to the Diffusion server that hosts the server publisher. In fact, the client publisher knows nothing of the server publisher, only the server and the topics it subscribes to, as if it were a normal client.

Server connections can be made automatically for a publisher by declaring them in `etc/Publishers.xml`. In this case the connections are made automatically during publisher startup. You can configure the behavior when such connections fail. It might cause the publisher to fail (if it is dependent upon the server as a data source) or it might allow the publisher to start but retry the connection periodically until it succeeds. Alternatively, it might do nothing.

Alternatively, the publisher can dynamically make server connections as, and when required. To do this it uses the `addServerConnection` method to create a `PublisherServerConnection` object, configure the connection as required and use the `connect` method on the object to make the connection

Whether server connections are made automatically or manually the publisher is notified when a connection is made using the `serverConnected` method and when the connection is closed or lost using the `serverDisconnected` method.

The publisher receives messages from server connections on the `messageFromServer` notification method.

The publisher is notified of the change of status (for example, removal) of any topics it is subscribed to at a server connection on the `serverTopicStatusChanged` notification method.

General utilities

General purpose utilities that can be used from within a publisher

There are a number of general purpose utilities available which can aid in the process of writing a publisher, for example:

Table 46: General publisher utilities

Utils	A set of general purpose utilities which include file handling, property handling, date and time formatting and more.
XMLUtils	A set of utilities to aid in the processing of XML.
HTTPUtils	A set of utilities to aid in HTTP processing.

Writing a publisher

How to approach writing a publisher

Note: This section covers only the main aspects of the publisher API. See the API documentation for full details.

There are demo publishers issued with Diffusion which have the source provided and these act as examples of working publishers.

In its simplest sense a publisher is responsible for providing topics, and publishing messages relating to those topics.

Before a publisher is written you need to carefully consider what it needs to do and what methods need to be implemented. The areas that need to be considered and the methods relating to them are discussed in the following sections.

There are many ways to approach the design of a publisher. For more information, see .

Related concepts

[Classic deployment](#) on page 640

Installing publishers into a stopped Diffusion instance.

Creating a Publisher class

A publisher is written by extending the abstract `Publisher` class (see Publisher API) and overriding any methods that must be implemented to achieve the functionality required by the publisher.

In all but the simplest of publishers it is likely that other classes must be written to perform the functionality required of the publisher.

The considerations of which methods must be overridden are discussed further within this section.

After the class is written and compiled, you can deploy it in the Diffusion server by specifying its details in `etc/Publishers.xml`

Publishers can also be deployed as a DAR file, sidestepping `etc/Publishers.xml`

See the section on testing for information about how to test the publisher.

Related tasks

[Building a publisher with mvndar](#) on page 516

Use the Maven plugin *mvndar* to build and deploy your publisher DAR file. This plugin is available from the Push Public Maven Repository.

Publisher startup

When a publisher is first loaded by the Diffusion server it can also be automatically started.

If not automatically started (or if it has been manually stopped), a publisher can be manually started by using the System Management interface. In either case the publisher processing goes through a number of startup steps. During these steps the `initialLoad` and `publisherStarted` methods are called and these methods can be implemented by the publisher to perform any initial processing like setting up the initial data state or adding initial topics.

Data state

A publisher typically holds some data on topics which it updates according to any data update events it might receive.

The data held by the publisher on the topics it provides is referred to as its state. It is up to the publisher whether the data state is managed as a whole or on topic by topic basis.

It is the responsibility of the publisher to initialize its state and keep it updated as appropriate. Clients that subscribe to topics usually want to know the current state of the data relating to that topic and the publisher provides this as an initial topic load message. Clients are notified of all changes to that state by the publisher sending out delta messages.

A publisher typically has its own data model represented by classes written to support the data for the publisher. Ways in which such a data model can be managed are discussed in [Designing your data model](#) on page 59.

Initial state

A publisher's data typically has some initial state which can be updated during the life of the publisher. The state clearly must be set up before a client requires it but exactly when this is done is up to the publisher.

The state of the data as a whole can be set up when the publisher starts. This can be done in the `initialLoad` method where all topics required can be set up and the data loaded as appropriate.

Alternatively, the state of the data relating to a topic can be initialized when the topic is added, which is not necessarily when the publisher is started.

Another option is that the initial state is provided by a data feed as it connects (or is connected to). If data is provided by a server connection, the initial state can be set up when the server connection is notified to the publisher or more typically the server provides an initial topic load message.

Data integrity

The integrity of the data is also the responsibility of the publisher and care must be taken to ensure that all updating of data state is thread-safe. For example, it must be borne in mind that a client can request a load of current state (for example, by subscription) at the same time as the state is being updated.

Note: The topic data feature automatically handles such data locking and in other cases topics might be locked as and when required.

Providing data state

If clients are to use the fetch facility to obtain the current state of topics, it will be necessary to consider the implementation of the `fetchForClient` method of the publisher.

Stateful and stateless topics

The topics that the publisher provides can store data state, but not all topics store data state. Topics that store data state are called *stateful topics*. Topics that do not store data state are called *stateless topics*.

The publisher has different mechanisms for publishing data through stateful or stateless topics. For more information, see [Publishing messages](#) on page 496.

Data inputs

For a publisher to be able to publish data to clients it must have a source for that data.

The data can be obtained from some type of feed, perhaps provided by some external API or it can be from some other application communicating using Diffusion protocols. This is entirely up to the publisher but Diffusion does offer some mechanisms.

DEPRECATED: Server connections

A publisher can receive its data from another publisher in a distributed environment as if it were a client of that publisher. To do this it is necessary for the publisher to connect to the server of the remote publisher and subscribe to its topics.

Connection to remote servers can be set up automatically for a publisher by configuring the connection in `etc/Publishers.xml`. A reference to such a server connection can be obtained by the publisher using the `getServerConnection` method or all connections can be obtained using `getServerConnections`.

Alternatively, a publisher can explicitly connect to remote servers using `addServerConnection`.

However the connection is made, the publisher is notified of the connection via the `serverConnected` method.

If a server connection is closed by the remote server or lost the publisher is notified through the `serverDisconnected` method.

Once a `serverConnection` is established the publisher must subscribe to topics on it and receive topic messages through the `messageFromServer` method. Typically, the first such method is a load message providing the initial state for the topic. Processing of the messages can be done within this method or alternatively the publisher can specify topic listeners to handle the messages on a per topic basis.

Control clients

A publisher can receive input from a control client.

Control clients can use the `TopicUpdateControl` feature to publish messages to topics. Where such topics have topic data the topic state is automatically updated and deltas are published to subscribed clients. Where topics do not have topic data, published messages are forwarded to subscribed clients (that is, it is assumed that the control client maintains the data state).

Control clients can also send messages to specific clients and these are forwarded to the clients automatically.

Handling client subscriptions

Clients subscribe to topics provided by publishers and whenever this occurs the publisher is notified through its subscription method. The publisher can perform any processing it requires on subscription.

Performance considerations

Any queries about subscriptions are expensive on resources and time, because these queries are synchronous and blocking. For example, querying whether a client is subscribed to a topic, what clients are subscribed to a specific topic, or what topic a specific client subscribes to.

If your publishers respond to add topic notifications or subscription notifications, ensure that these responses are efficient. These publisher actions are now serialized in a single thread and as a result the publisher can become a bottleneck and hold up processing.

Using topic data

Where a topic is inherently stateful and has associated data, the use of topic data is recommended. Topic data automatically handles topic loading.

Topic loading

Typically, on subscription, the publisher provides the client with the current state of the data for the topic. It can do this by creating a new topic load message and populating it with a representation of the state. Rather than doing this every time a client subscribes it is generally more efficient for the publisher to create a topic load message only when the state changes and send this same message out to every client that subscribes.

This provision of the current state is known as the topic load. This can be done in one of the following ways:

Topic load in subscription method

If the topic has not already been loaded by a topic loader (see below), the loaded parameter of the subscription method is false. In this case, the normal action is for the publisher to send a topic load message to the client (passed as a parameter to subscription) through its send method.

Topic loaders

A topic loader is an object that implements the `TopicLoader` interface and can be used to perform any topic load processing that is required for one or more topics. Topic loaders can be declared for a `Publisher` using the `Publisher.addTopicLoader` method. This is typically done in the `initialLoad` processing and must be done before any topics that are loaded by the topic loader are added.

Hierarchic subscription

When a client subscribes to a topic the publisher can choose to subscribe the client to other topics or to subordinate topics. This can be done using the `Client.subscribe` methods.

A client itself can request subscription to a hierarchy of topics using topic selectors but this is an alternative method of handling hierarchies.

Publishing messages

Publishing a message means sending it to all clients subscribed to a topic. The message itself nominates the topic to which it relates.

A message for publishing can be created and populated by the publisher and then published using publishing methods on the topic or the publisher itself.

Exclusive messages

To send a message from a publisher to all clients subscribed to a topic except one single client, it can use the `publishExclusiveMessage` method. This might be appropriate if the message being published is a result of receiving a message from a client which you do not want to send back to that client.

Message priority

The priority at which a message is to be sent can be altered from the normal priority. For example, an urgent message can be sent with high priority causing it to go to the front of the client's queue.

Message acknowledgment

If acknowledgment of a message is required then it can be set as an acknowledged message. If any clients do not respond to a acknowledged message within a specified timeout, the publisher is notified on its `messageNotAcknowledged` method.

Publishing using stateful topics

Stateful topics are topics that store a current value in the Diffusion server. You can publish using stateful topics in a simple way or as part of a more complex transactional update.

This section covers working with topics that have associated topic data that extends the `PublishingTopicData` interface. There are other types of topic data that can be associated with topics, for example `PagedTopicData`.

Simple updates to a stateful topic

Use the `updateAndPublish` or `updateAndPublishFromDelta` method on the topic data of a stateful topic to update the topic state. Updating the topic data of a stateful topic publishes a delta to all subscribed client that includes the changes to the topic data.

```
topic.getData().updateAndPublish(update);  
// OR  
topic.getData().updateAndPublishFromDelta(deltaUpdate);
```

Transactional updates to a stateful topic

Stateful topics can be updated transactionally by bracketing the updates with the `startUpdate` and `endUpdate` methods of the associated topic data.

Combining updates to the topic data as part of a single transaction can be useful when the stateful topic is a record topic that has multiple records and fields that can be updated from separate sources. These fields can be updated separately within the transaction, but all updates in the topic state are published to the subscribing clients at the same time.

```
// Start the transaction  
data.startUpdate();  
try {  
    // Make multiple updates as part of a single transaction  
    data.update(firstUpdate);  
    data.update(secondUpdate);  
    data.update(thirdUpdate);  
    data.update(fourthUpdate);  
}
```

```

        data.update(fifthUpdate);

        // Publish the updates that have been made in this transaction
        if (data.hasChanges()){
            data.publishMessage(data.generateDeltaMessage());
        }
    }
    finally {
        // Complete the transaction
        data.endUpdate()
    }
}

```

Publishing using stateless topics

Stateless topics are topics that have no associated current value in the Diffusion server. You can publish using a stateless topic in a simple way or as part of a complex action triggered by a client subscription to that topic.

Stateless topics have no associated topic data. These topics simply pass through any updates that are made to them to the subscribing clients.

Simple updates using a stateless topic

Use the `publishMessage` method on the topic to publish data using the stateless topic at any time. This data is not stored on the Diffusion server and is sent as-is to all current subscribers to the stateless topic.

```
topic.publishMessage(data);
```

On-subscription updates using a stateless topic

Stateless topics pass through data from the publisher to the subscribing clients. This published data can be a full update or a delta on previous updates. If a client subscribes to a stateless topic after a full update and before a delta, the client receives the delta, but does not have the base data to apply it to.

To ensure that a newly subscribing client receives a full update for that topic, the publisher `subscription` method — which is called every time a client subscribes to a topic managed by the publisher — can publish an update that contains all the data required by the subscriber to that topic. This data is not published until the client subscription to the topic is complete.

Using the `subscription` method can cause performance issues. For more information, see [Handling client subscriptions](#) on page 495.

To publish a message to the topic whenever a client subscribes to the topic, override the `Publisher.subscription()` method in your own publisher class and include in the method a call to `topic.publishMessage()` that passes in all the data to publish.

```

@Override
protected void subscription(final Client client, final Topic
topic, final boolean loaded)
    throws APIException {

    // Do the required processing to create the full update
    message to
    // publish for the newly subscribed client.

    // Publish that message to the stateless topic
    topic.publishMessage(data);
}

```

DEPRECATED: Publishing using paged topics

Paged topics are topics that store current values as lines in a page. Paged topic data can be updated at any time using a set of methods that enable additions, updates, and deletions.

While paged topics have associated topic data, that data does not extend the `PublishingTopicData` interface. Because of this paged topics do not act in the same way as other stateful topics.

All of these methods lock the data, perform the update, notify any affected clients of changes as appropriate and unlock the data. If it is required to lock the data over more than one update so that it cannot be changed whilst manipulating it, use the `lock()` mechanism on the data.

The method signatures vary according to the type so the line referred to in the table below refers to a String or a record as appropriate.

Some methods are usable only with unordered topic data, some only with ordered, and some behave differently according to the type.

Methods for use with ordered topic data typically act on only one record at a time and do not require an index reference to the record.

Table 47: Usable methods with ordered topic data

<code>add(line)</code>	Add the specified line to the end of the data (if unordered) or at the appropriate position (if ordered).
<code>add(List<line>)</code>	Add the specified list of lines to the data. For unordered data the lines are all added at the end. For ordered data this is the same as calling <code>add(line)</code> repeatedly.
<code>add(int, List<line>)</code>	Add the specified list of lines into the data at the specified index.
<code>update(line)</code>	This is functionally equivalent to calling <code>remove(line)</code> followed by <code>add(line)</code> .
<code>remove(int, int)</code>	Removes one or more lines. The first number specified is the index to start from and the second is the number of lines to remove. The code <code>remove(0, 1)</code> removes the first line.
<code>remove(line)</code>	The current line that is equal to the specified line according to the comparator is removed. If there is more than one matching line, the duplicates policy specifies which is removed. If no matching line is found, this has no effect.

Methods for use with unordered topic data typically requires an index reference to the record that they act on.

Table 48: Usable methods with unordered topic data

<code>add(line)</code>	Add the specified line to the end of the data (if unordered) or at the appropriate position (if ordered).
<code>add(List<line>)</code>	Add the specified list of lines to the data. For unordered data the lines are all added at the

	end. For ordered data this is the same as calling <code>add(line)</code> repeatedly.
<code>add(int, line)</code>	Add the specified line into the data at the specified index. The line is inserted at the specified index. Indexing starts at 0. Using <code>add(0, line)</code> is the same as inserting the line at the start of the data.
<code>add(int, List<line>)</code>	Add the specified list of lines into the data at the specified index.
<code>update(int, line)</code>	Update the line at the specified index with the specified line. This effectively replaces the line with the one supplied.
<code>remove(int, int)</code>	Removes one or more lines. The first number specified is the index to start from and the second is the number of lines to remove. The code <code>remove(0, 1)</code> removes the first line.

Methods are also available to get specified lines or a range of lines which might help in updating.

DEPRECATED: Topic locking

All locking of the topic state is handled automatically. However, when the state of the topic is not maintained by the topic (the topic is stateless), it is the responsibility of the publisher application to handle locking.

The publisher must consider the issue of locking the topic whilst its state is changed and delta messages published.

By default, all topics have locking enabled which allows the publisher to lock and unlock the topic as required. When a client subscribes to a topic the subscription method of the publisher, normally sends the current state of the topic to the client.

Threads that update the topic state and publish messages must also lock the topic for the duration of the update and publish. If this technique is not employed, a delta message might be sent to a client before the subscription method has the opportunity to send a topic load message. This can cause a failure at the client if topic aliasing is in use as aliasing relies upon the topic load message reaching the client first. Even if topic aliasing were not in use, the client application must be prepared for a delta arriving before the topic load.

Handling clients

Interacting with clients from within a publisher

A publisher is notified when a client subscribes to one of its topics through the `subscription` method and when the client unsubscribes the `unsubscription` method is called.

A publisher can receive message from clients and send messages to clients (see below).

A client can request the state of any topic or topics at any time even if not subscribed to it. This is referred to as 'fetch' request. Such a request can be routed to the publisher's `fetchForClient` method if a topic has no topic data.

Other than the above, a publisher is not normally notified of any other client activity. However a publisher can choose to receive client notifications using the `Publishers.addListener` method. Using client notifications, a publisher can even handle a fetch request for a topic that does not exist and return a reply (using `Client.sendFetchReply`) without the overhead of actually creating a topic.

A publisher can also choose to close or abort clients.

Sending and receiving client messages

In addition to publishing messages to all clients subscribed to a topic, you can send a message to only a single client using the `Client.send` method.

A client can also send messages to the publisher and these are received on the `messageFromClient` method which handles them accordingly. Only implement this method if messages are expected from clients. Alternatively the publisher specifies topic listeners to handle the messages on a per topic basis.

If it is a Unified API client that sends a message to a publisher on a topic, the message is mapped to a `delta TopicMessage`.

The 'client groups' facility allows messages to be sent to all clients in a group. There is also an 'exclusive message' facility which caters for sending to all but one client in a group.

Publisher closedown

A publisher is stopped and removed when the Diffusion server closes but can also be stopped and restarted, or stopped and removed by using the System Management interface.

However a publisher is stopped it always goes through a set of closedown steps, during which the `publisherStopping` and `publisherStopped` methods are called. A publisher can implement these methods if required to perform any special processing such as tidying up resources used.

Publisher removal

When a publisher is finally removed (either during server closedown or by using System Management), it cannot be restarted again within the same server instance. After removal the `publisherRemoved` method is called and this gives the publisher the opportunity to perform any final tidy up processing.

Stopping and restarting using System Management

By default, you cannot stop and restart a publisher using the System Management functions because in order for this to work the publisher must cater for the integrity of its state when this happens. As topics are also removed during stopping, the publisher must also be able to restore these topics if it were restarted.

If a publisher does want to cater for stop and restart using System Management, it must override the `isStoppable` method to return true. The publisher code must be able to recover topics and data state on restart.

Testing a publisher

There are various ways you can test your publishers after you have written them and deployed them on a Diffusion server instance.

The easiest way to perform some initial tests is to start it and try it out using some of the supplied testing tools. For example, you can start one or more instances of the client test tool, connect each to the test server and subscribe to the publisher's topic or topics. The initial topic load data is displayed and any messages sent as deltas are also displayed in each client. This tool can also be used to send messages to the publisher from the client.

Ultimately such tests are limited and you might want to develop Java tests which simulate clients using the Java external client API (or the Windows external client API).

Test as soon as possible with the actual clients that are going to be used. So, for example, you might want to develop browser clients using JavaScript, Flash or Silverlight.

It can help to diagnose problems with the publisher if it has diagnostic logging encoded within it. Such logging can be provided only at fine level and this logging level used only during testing.

Client queues

How messages sent to clients are queued and how such queues can be manipulated by publishers

The Diffusion server maintains an outbound queue of messages for each client. Whenever a message is published to a topic, it is placed in the queue of each client subscribed to that topic as will any message sent explicitly to the client. Messages are sent to the client strictly in the order that they are enqueued.

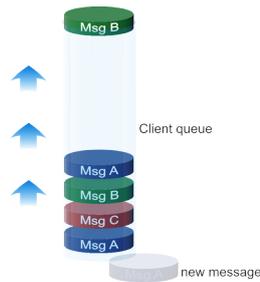


Figure 26: The message queue

A publisher is able to enquire upon the details of a particular client's queue and even to change some aspects of the queue's behavior.

Queue enquiries

How the publisher can access details of client queues

A publisher can enquire upon the following information about a particular client's queue using the client interface:

- The current queue size
- The maximum queue size (The limit the queue can reach before the client is automatically disconnected.)
- The largest queue size (The largest size the client queue has been since the client connected.)

Maximum queue depth

To limit the backlog of messages queued for a client that is not consuming them quickly enough you can indicate a maximum queue depth for clients.

Choose this size carefully as a large queue size can lead to excessive memory usage and vulnerability to Denial of Service attacks, whilst a small queue size can lead to slow clients being disconnected too frequently.

The maximum queue depth for clients can be configured for a client connector in `etc/Connectors.xml`. A default value can also be configured in `etc/Server.xml` for connectors that do not explicitly specify a value.

These values can be changed dynamically at run time using System Management but they only take effect for new clients.

Queue notification thresholds

A publisher can receive notifications when a client queue has reached a certain size and use this information to decide whether or not to act on the situation.

For example, the publisher might want to notify the client so that it can take some action (like suspending processing). As there is little point in queuing a message to tell the client that their queue is becoming full, this is probably done using a high priority message which goes to the front of the queue.

To this end, an upper notification threshold can be set for a client's queue. This is expressed as a percentage of the maximum queue depth at which a notification is sent to any client listeners that are declared. A client listener is any object that implements the `ClientListener` interface and such a listener can be added from within a publisher using the `Publishers.addEventListener` method. Listeners are notified of the limit breach using the `clientQueueThresholdReached` method.

In addition a lower notification threshold can be specified. The lower threshold is a percentage of the maximum queue depth at which a notification occurs when a message is removed from the queue causing the queue size to reach the threshold if (and only if) the upper threshold has been breached.

When the `clientQueueThresholdReached` method is called on the client listener it indicates whether it was the upper or lower threshold that was reached.

The thresholds to use for clients can be configured for a connector in `connectors.properties`. If not specified, the default thresholds specified in `etc/Server.xml` are used.

The thresholds on a client connector can be changed dynamically at run time using System Management, but the new values only take effect for new clients.

Thresholds can also be set or changed from within the publisher for a connected client using the `Client.setQueueNotificationThresholds` method.

Tidy on unsubscribe

When a client unsubscribes from a topic, the topic updates that are already queued for delivery to the client are delivered. These messages can be cleared from the queue if the client does not want to receive them.

After a message is queued for a client, it will be delivered. This means that a client can unsubscribe from a topic but still receive messages queued for it on it on that topic. This is generally what is required as the messages were sent whilst the client was subscribed.

However, it can be decided that once the client has unsubscribed from a topic then the client no longer has any interest in any messages for that topic and such messages are removed from the queue. To achieve this there is an option on a topic (using the `setTidyOnUnsubscribe` method) to indicate that messages for the topic must be removed from client queues when the client unsubscribes from that topic.

Client Geo and Whols information

When a client connects to Diffusion, information about that client's geographic location is looked up and the information is made available to publishers.

When a client first connects to a Diffusion server, its remote IP address is immediately available (using the `Client.getRemoteAddress` method) as well as other details obtained from the embedded Geolp database. Further host and geographic details about the client are obtained using the Diffusion "WhoIs service".

Geolp information

Diffusion ships with a GeolP database from [MaxMind](#). This provides information about Locale and geographic co-ordinates. The Java API includes utilities (`GeoIPUtils`) to make use of this database.

This is a public domain database and is free to use. You can purchase the more accurate database from MaxMind and change the configuration in the `etc/Server.xml` properties to use the new database.

The database can be disabled but its use is mandatory if you are going to use client connection or subscription validation policies. For more information, see .

Whols

The inbuilt Whols service can provide additional information about clients, however the lookup of the Whols information might take some time, especially if it is not already cached. This means that notification of the connection and further processing of the client cannot wait for this information to become available. For this reason the resolution of the client's Whols details is notified to client listeners separately from client connection on the `clientResolved` method.

When a client is first connected it is likely that the Whols details of the client are not available. This can be checked using the `Client.isResolved` method. When the details become available they can be obtained from the client using the `getWhoIsDetails` method which returns an object containing the following information:

Table 49: Whols

Address	The client's IP Address – this is the same as that obtained using <code>Client.getRemoteAddress</code> .
Host	The resolved host name of the client. If the host name cannot be resolved, the address is returned.
Resolved name	The fully resolved name of the client. Exactly what this means depends upon the Whols provider in use. If a fully resolved name cannot be obtained, the host name value is returned.
Locale	<p>Returns the result of a geographic lookup of the IP address indicating where the address was allocated. The country of the locale is set to the international two-letter code for the country where the internet address was allocated (for example, NZ for New Zealand). If the internet address cannot be found in the database, the country and language of the returned locale are set to empty Strings.</p> <p>Three country values can be returned that do not exist within the international standard (ISO 3166). These are EU (for a non-specific European address), AP (for a non-specific Asia-Pacific address) and ** (an internet address reserved for private use, for example on a corporate network not available from the public internet).</p> <p>The language of the returned locale is set to the international two-letter code for the official language of the country where the internet address was allocated. Where a country has more than one official language, the language is set to that which has the majority of native speakers. For example, the language for Canada is set to English (en) rather than French (fr). Non-specific addresses (EU and AP), private internet addresses (**), and addresses not found within the database, all return an empty string for language.</p>
WholsData	This is data extracted from an enquiry upon a 'Whols' data provider.

Local	Indicates whether the client address is a local address, in which case no locale or WhoIsData is available.
Loopback	Indicates whether the client address is a loopback address in which case no locale or WhoIsData is available.

The Diffusion WhoIs service

The Diffusion WhoIs service runs as a background task in the Diffusion server. It looks up client details and caches them in case the same client reconnects later.

The behavior of the WhoIs service is configured in `etc/Server.xml`. This allows the following to be specified:

Table 50: WhoIs service

The WhoIs provider	This specifies a class to use for WhoIs lookups. A default WhoIs provider is provided with Diffusion.
Number of threads	The number of background threads that processes WhoIs resolver requests. More threads will improve the WhoIs performance. Setting this to 0 disables WhoIs.
WhoIs Host/Port	These details provide the location of an internet based WhoIs lookup server that adheres to the RFC3912 WhoIs protocol. This is used by the default WhoIs provider. This defaults to using the RIPE database.
Cache details	Specifying the maximum size of the cache of details and how long cache entries are retained before being deleted. If you envisage large numbers of different clients connecting over time, it is important to consider the automatic cache tidying options on the service.

The WhoIs service can be disabled both by setting the number of threads to zero and removing the whois configuration element.

WhoIs providers

The Diffusion WhoIs provider is a class which implements the `WhoIsProvider` interface of the WhoIs API. This is used by the WhoIs service to lookup WhoIs details for connected clients.

Default provider

A default `WhoIsProvider` (`WhoIsDefaultProvider`) is provided with Diffusion.

A connection is made to the WhoIs server specified in `etc/Server.xml` and returned details are parsed and used to update the supplied details. Child details objects are added for any separate WhoIs records found and the type of such objects is the key of the first WhoIs record entry (for example, person). Where duplicate field names occur then all but the first are suffixed by “_n”, where *n* is a number distinguishing the entries.

The netname entry is used as the resolved name if present.

Custom provider

If the behavior of the issued default WhoIs provider is not exactly what is required then users can write their own WhoIs provider which must implement the `WhoIsProvider` interface. The name of the user-written class can be specified in `etc/Server.xml` and must be deployed on the Diffusion server's classpath.

Client groups

Clients that are connected can be managed in client groups allowing messages to be sent to groups of clients

Client groups are a convenient way of managing groups of clients with common attributes.

A publisher can create a client group and add clients to it. Messages can be sent to the group of clients rather than to individual clients.

When a client disconnects it is automatically removed from all client groups of which it is a member.

A client group belongs to the publisher that created it and can be used only from within that publisher. Group names are unique within the publisher only.

Client groups are a feature of the Java API using the `Publisher` interface.

Creating a client group

A publisher can create a new client group at any time using `createClientGroup`. When creating the group it is given a name which must be unique within the publisher. When a client group is created a reference to a `ClientGroup` object is returned.

Adding clients to a group

Clients can be added to a client group using the `addClient` method on a `ClientGroup` object.

Sending messages to clients in a group

A message can be sent to all clients in a client group using the `ClientGroup.send` method. To send to all but a specified client use `publishExclusiveMessage`.

Removing clients from a group

To remove clients from a client group, use the `ClientGroup.removeClient` method.

Removing a client group

To delete / remove a client group that is no longer required, use the `Publisher.removeClientGroup` method.

Other methods

All clients within a client group can be listed using `ClientGroup.getClients`.

You can check whether a particular client is already a member of a client group using `ClientGroup.containsClient`.

To enquire upon which client groups a particular client belongs to you can use `Publisher.getClientGroupMembership`.

To get a reference to a named client group from a publisher use `getClientGroup`.

Temporary client groups

Normally client groups have publisher scope, but you can create temporary groups using the `union` and `intersect` methods. These methods allow for the creation of client groups which are not managed by the client group manager. These temporary groups allow for the sending of messages which contain clients from different groups.

Client notifications

A publisher can opt to receive certain notifications regarding clients. It does this by adding a `ClientListener` which can be the publisher itself or any other object that implements the `ClientListener` interface.

A listener is added using the `Publishers.addEventListener` method.

All notifications are passed a reference to the client in question which can be interrogated for further information as required.

Notifications received on the `ClientListener` interface are as follows:

Table 51: Client listener notifications

<code>clientConnected</code>	This is called whenever a new client connects. It is not necessarily a client that is subscribing to one of the publisher's topics.
<code>clientResolved</code>	This is called when a newly connected client is resolved. A client's full geographical information is not necessarily available as soon as a client connects and so this method is called separately after the client has been resolved.
<code>clientSubscriptionInvalid</code>	This is called whenever a client attempts to subscribe to a topic that does not exist. This might be because the topic is not yet available and this gives a publisher the opportunity to create the topic and subscribe the client to it.
<code>clientFetchInvalid</code>	This is called whenever a client attempts to fetch a topic that does not exist. This gives the publisher the opportunity to respond to fetch request on a non-existent topic. A publisher can even reply to such a request without having to create a topic using the <code>sendFetchReply</code> method.
<code>clientSendInvalid</code>	This is called whenever a client attempts to send a message to a topic that does not exist, or to which the client is not subscribed. This enables a client to send a message to a topic and for that topic to be created and subscribed to on demand, or send data when a response is never expected.
<code>clientQueueThresholdReached</code>	This is called whenever a client's queue breaches an upper queue notification threshold or returns to a lower queue notification threshold. Parameters indicate which threshold has been reached and the threshold value.
<code>clientCredentials</code>	This is called whenever a client supplies new credentials after connection. It is called after the authentication handlers and authorization handlers (if any exist) have validated the credentials.

clientClosed	This is called whenever a client disconnects. The reason for disconnection can be obtained using theClient.getCloseReason method.
--------------	---

Adding a ClientListener

You can add a `ClientListener` to listen for client notifications.

About this task

Procedure

So a publisher can add itself as a listener for client notifications as follows:

Results

```
public class MyPublisher extends Publisher implements ClientListener
{
    protected void initialLoad() throws APIException {
        Publishers.addEventListener(this);
    }
}
```

Using DefaultClientListener

How to use the default client listener to avoid implementing all methods.

About this task

The publisher must implement all of the `ClientListener` methods.

Procedure

For convenience, an abstract `DefaultClientListener` class is provided which has empty implementations of all methods. This can be extended to produce a class which implements only the methods you are interested in. Alternatively an anonymous class can be used within the publisher as follows:

Results

```
protected void initialLoad() throws APIException {
    Publishers.addEventListener(
        new DefaultClientListener() {
            public void clientConnected(Client client) {
                LOG.info("Client {} connected",client);
            }

            public void clientClosed(Client client) {
                LOG.info("Client {} closed",client);
            }
        });
}
```

Developing other components

Diffusion provides Java APIs that enable you to customize the behavior of your Diffusion server and related components.

Local authentication handlers

You can implement authentication handlers that authenticate client connections to the Diffusion server.

A local authentication handler is an implementation of the `AuthenticationHandler` interface. Local authentication handlers can be implemented only in Java. The class file that contains a local authentication handler must be located on the classpath of the Diffusion server.

For more information, see [Authentication API](#).

Related concepts

[Configuring authentication handlers](#) on page 561

Authentication handlers and the order that the Diffusion server calls them in are configured in the `Server.xml` configuration file.

Developing a local authentication handler

Implement the `AuthenticationHandler` interface to create a local authentication handler.

About this task

Local authentication handlers can be implemented only in Java.

Note: Where `c.p.d` is used in package names, it indicates `com.pushtechnology.diffusion`.

Procedure

1. Create a Java class that implements `AuthenticationHandler`.

```
package com.example;

import com.pushtechnology.diffusion.client.details.SessionDetails;
import
    com.pushtechnology.diffusion.client.security.authentication.AuthenticationHan
import com.pushtechnology.diffusion.client.types.Credentials;

public class ExampleAuthenticationHandler implements
    AuthenticationHandler{

    public void authenticate(String principal, Credentials
        credentials,
            SessionDetails sessionDetails, Callback callback) {

        // Logic to make the authentication decision.

        // Authentication decision
        callback.abstain();

        // callback.deny();
```

```
        // callback.allow();  
    }  
}
```

- a) Ensure that you import `Credentials` from the `c.p.d.client.types` package, not the `c.p.d.api` package.
 - b) Implement the `authenticate` method.
 - c) Use the `allow`, `deny`, or `abstain` method on the `Callback` object to respond with the authentication decision.
2. Package your compiled Java class in a JAR file and put the JAR file in the `ext` directory of your Diffusion installation.

This includes the authentication handler on the server classpath.

3. Edit the `etc/Server.xml` configuration file to point to your authentication handler.
Include the `authentication-handler` element in the list of authentication handlers. The order of the list defines the order in which the authentication handlers are called. The value of the `class` attribute is the fully qualified name of your authentication handler class. For example:

```
<security>  
  <authentication-handlers>  
  
    <authentication-handler  
      class="com.example.ExampleAuthenticationHandler" />  
  
  </authentication-handlers>  
</security>
```

4. Start or restart the Diffusion server.
 - On UNIX-based systems, run the `diffusion.sh` command in the `diffusion_installation_dir/bin` directory.
 - On Windows systems, run the `diffusion.bat` command in the `diffusion_installation_dir\bin` directory.

Related concepts

[User-written authentication handlers](#) on page 143

You can implement authentication handlers that authenticate clients that connect to the Diffusion server or perform an action that requires authentication.

[Authentication](#) on page 140

You can implement and register handlers to authenticate clients when the clients try to perform operations that require authentication.

Related tasks

[Developing a composite authentication handler](#) on page 510

Extend the `CompositeAuthenticationHandler` class to combine the decisions from multiple authentication handlers.

[Developing a control authentication handler](#) on page 402

Implement the `ControlAuthenticationHandler` interface to create a control authentication handler.

[Developing a composite control authentication handler](#) on page 405

Extend the `CompositeControlAuthenticationHandler` class to combine the decisions from multiple control authentication handlers.

Developing a composite authentication handler

Extend the `CompositeAuthenticationHandler` class to combine the decisions from multiple authentication handlers.

About this task

If there are several, discrete authentication steps that must always be performed in the same order, packaging them as a composite authentication handler simplifies the server configuration.

This example describes how to use a composite authentication handler to call multiple local authentication handlers in sequence.

Procedure

1. Create the individual authentication handlers that your composite authentication handler calls. You can follow steps in the task [Developing a local authentication handler](#) on page 508. In this example, the individual authentication handlers are referred to as `HandlerA`, `HandlerB`, and `HandlerC`.
2. Extend the `CompositeAuthenticationHandler` class.

```
package com.example;

import com.example.HandlerA;
import com.example.HandlerB;
import com.example.HandlerC;

import
    com.pushtechology.diffusion.client.security.authentication.CompositeAuthenti

public class CompositeHandler extends
    CompositeAuthenticationHandler {

    public CompositeHandler() {
        super(new HandlerA(), new HandlerB(), new HandlerC());
    }

}
```

- a) Import your individual authentication handlers.
 - b) Create a no-argument constructor that calls the super class constructor with a list of your individual handlers.
3. Package your compiled Java class in a JAR file and put the JAR file in the `ext` directory of your Diffusion installation. This includes the composite authentication handler on the server classpath.
 4. Edit the `etc/Server.xml` configuration file to point to your composite authentication handler. Include the `authentication-handler` element in the list of authentication handlers. The order of the list defines the order in which the authentication handlers are called. The value of the `class` attribute is the fully qualified class name of your composite authentication handler. For example:

```
<security>
  <authentication-handlers>
```

```
<authentication-handler class="com.example.CompositeHandler" />
</authentication-handlers>
</security>
```

5. Start the Diffusion server.

- On UNIX-based systems, run the `diffusion.sh` command in the `diffusion_installation_dir/bin` directory.
- On Windows systems, run the `diffusion.bat` command in the `diffusion_installation_dir\bin` directory.

Results

When the composite authentication handler is called, it calls the individual authentication handlers that are passed to it as parameters in the order they are passed in.

- If an individual handler responds with ALLOW or DENY, the composite handler responds with that decision to the server.
- If an individual handler responds with ABSTAIN, the composite handler calls the next individual handler in the list.
- If all individual handlers respond with ABSTAIN, the composite handler responds to the server with an ABSTAIN decision.

Related concepts

[User-written authentication handlers](#) on page 143

You can implement authentication handlers that authenticate clients that connect to the Diffusion server or perform an action that requires authentication.

[Authentication](#) on page 140

You can implement and register handlers to authenticate clients when the clients try to perform operations that require authentication.

Related tasks

[Developing a local authentication handler](#) on page 508

Implement the `AuthenticationHandler` interface to create a local authentication handler.

[Developing a control authentication handler](#) on page 402

Implement the `ControlAuthenticationHandler` interface to create a control authentication handler.

[Developing a composite control authentication handler](#) on page 405

Extend the `CompositeControlAuthenticationHandler` class to combine the decisions from multiple control authentication handlers.

Push Notification Bridge persistence plugin

The Push Notification Bridge stores subscription information in memory. To persist this information past the end of the bridge process, implement a persistence plugin.

The persistence API

The Push Notification Bridge persistence API provides the following interfaces for you to use to develop your persistence plugin:

SaverFactory

Your implementation of this interface is referenced by the bridge configuration and is called by the bridge to build the `Saver` object.

Saver

Your implementation of this interface is called by the bridge when push notification subscriptions and unsubscriptions are made. It uses this information to update the persisted model of the subscriptions.

Loader

Your implementation of this interface is called by the bridge when it starts and is used to update the model of subscriptions held in memory by the bridge.

Context

This provides a context for events passed to the `Saver` interface. It is used for logging and audit trail purposes.

Full API documentation is available at the following location: [Java Unified API documentation](#).

An example implementation of the persistence API is available on GitHub: <https://github.com/pushtechology/push-notification-persistence-example>. This example is basic and uses Java serialization to persist the subscription model.

Note: The example persistence plugin is not suitable for production use.

Developing the persistence plugin

A JAR file that contains the persistence API is available on the Push Technology Maven repository.

To use Maven to declare the dependency, first add the Push Technology public repository to your `pom.xml` file:

```
<repositories>
  <repository>
    <id>push-repository</id>
    <url>https://download.pushtechology.com/maven/</url>
  </repository>
</repositories>
```

Next declare the following dependency in your `pom.xml` file:

```
<dependency>
  <groupId>com.pushtechology</groupId>
  <artifactId>push-notification-persistence-api</artifactId>
  <version>1.0</version>
</dependency>
```

Using the persistence plugin

1. Compile your persistence code.
2. Ensure that the compiled code is on the classpath of the JVM that runs the bridge.
3. Configure the bridge to use your persistence plugin.

Use the `saverFactory` attribute of the `persistence` element to specify the name of the `SaverFactory` class in your plugin. For example:

```
<persistence saverFactory="com.example.pnb.SaverFactory" />
```

The content of the `persistence` element can be text content. This content is passed into the `saver factory` as arguments.

For more information, see [Configuring your Push Notification Bridge](#) on page 663.

Related concepts

[Push notification networks](#) on page 123

Consider whether your solution will interact with push notification networks.

[Example: Send a request message to the Push Notification Bridge](#) on page 372

The following examples use the Unified API to send a request message on a topic path to communicate with the Push Notification Bridge. The request message is in JSON and can be used to subscribe or unsubscribe from receiving push notifications when specific topics are updated.

[Push Notification Bridge](#) on page 660

The Push Notification Bridge is a Diffusion client that subscribes to topics on behalf of other Diffusion clients and uses a push notification network to relay topic updates to the device where the client application is located.

Using Maven to build Java Diffusion applications

Apache™ Maven is a popular Java build tool and is well supported by Java IDEs. You can use Apache Maven to build your Diffusion applications.

The Push Technology public Maven repository

Push Technology publishes Diffusion components and related artifacts to a public Maven repository at the following location: <http://download.pushtechnology.com/maven>.

The published artifacts include the following:

Table 52: Artifacts

Artifact	Maven coordinates	Description
Diffusion Unified API	<code>com.pushtechnology.diffusion:api:jar:5.9.0</code>	The Diffusion Unified API interfaces only. Use this artifact for compilation only. The JAR includes the source and Javadoc attachments.
Diffusion Clients	<code>com.pushtechnology.diffusion:client:jar:5.9.24</code>	The Diffusion client library. The runtime library includes both the Unified API and the Classic API.
mvndar	<code>com.pushtechnology.tools:dar-maven-plugin:maven-plugin:1.2</code>	A Maven plugin for building DAR files.

To use the Push Technology public Maven repository, add the following repository description to your `pom.xml` file:

```
<repositories>
  <repository>
    <id>push-repository</id>
    <url>https://download.pushtechnology.com/maven/</url>
  </repository>
</repositories>
```

Related concepts

[Building the demos using mvndar](#) on page 1072

You can use the Maven plugin *mvndar* with the provided `pom.xml` file to build the demos.

Build client applications

You can build and run Diffusion Java client applications without installing the Diffusion product. The Diffusion client JAR is all you need.

The following `pom.xml` shows how to declare the appropriate dependencies:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>myclient</artifactId>
  <version>1.0-SNAPSHOT</version>

  <repositories>
    <repository>
      <id>push-repository</id>
      <url>https://download.pushtechnology.com/maven/</url>
    </repository>
  </repositories>

  <dependencies>
    <dependency>
      <groupId>com.pushtechnology.diffusion</groupId>
      <artifactId>diffusion-client</artifactId>
      <version>5.9.24</version>
    </dependency>
  </dependencies>
</project>
```

Build publishers with Maven

The Diffusion API for publishers is not available in the Push Technology public Maven repository. To build publishers, you must install the product locally and depend on `diffusion.jar` using a Maven system scope.

DAR files

The preferred way to deploy publishers is to build them into a DAR. DARs are JAR format files that contain compiled code, libraries, and configuration. They have a similar purpose to Java EE EAR or WAR files, and can be dynamically deployed to and undeployed from a running Diffusion server



Figure 27: Example folder structure inside a DAR file

The root folder name is the name of the publisher. For example, MyPublisher.

- The `META-INF` directory contains the `MANIFEST.MF` file.
This file contains an attribute, `Diffusion-Version`, which specifies the minimum version number of Diffusion on which this publisher runs. This prevents deployment of publishers to Diffusion instances which might not support features of the publisher or have different API signatures.

```
Manifest-Version: 1.0
Diffusion-Version: 5.9.24
```

- The `etc` directory can contain the following files.

etc/Publishers.xml

You must include this file.

The `Publishers.xml` file has the same structure and the one in a Diffusion installation's `etc` directory. For more information, see [Publishers.xml](#) on page 606.

For example:

```
<publishers>
  <publisher name="MyPublisher">

  <class>com.pushtechnology.diffusion.test.publisher.MyPublisher</
class>
      <start>true</start>
      <enabled>true</enabled>
  </publisher>
</publishers>
```

etc/Aliases.xml (optional)

Include this file if there are associated HTML files.

etc/SubscriptionValidationPolicy.xml

Include this file if it is referenced from the `etc/Publishers.xml` file.

These files are normally found in the Diffusion server installation's `etc` directory, but contain only information relating to the publisher being deployed. Files that affect the operation of the Diffusion server and have no relationship to the publisher are not loaded.

- The `ext` directory contains all Java code required by your publisher.
You can also include any required third-party JAR files or resources in this folder.

- The `html` is optional and can contain any HTML files or web assets required by the publisher.

mvndar

The preferred way to build a DAR *mvndar* is a Maven plugin for creating DAR files. More information about *mvndar* is available at the following locations:

- <http://pushtechology.github.io/mvndar/index.html>

Example: Using Maven to build the demo applications

If you selected the demo applications when you installed Diffusion, the source files and an example Maven `pom.xml` can be found in the directories beneath the `demos/src` directory. The example uses *mvndar*, and depends on `diffusion.jar` using system scope.

For more information, see [Building the demos using mvndar](#) on page 1072.

Related concepts

[Classic deployment](#) on page 640

Installing publishers into a stopped Diffusion instance.

[Hot deployment](#) on page 641

Installing publishers into a running Diffusion instance.

[Deployment methods](#) on page 641

There are two ways to deploy a DAR file: file copy or HTTP.

Building a publisher with mvndar

Use the Maven plugin *mvndar* to build and deploy your publisher DAR file. This plugin is available from the Push Public Maven Repository.

Before you begin

This task describes how to build existing publisher code into a DAR file for deployment on the Diffusion server. Develop your publisher code before beginning this task. For more information, see [Writing a publisher](#) on page 492.

You must have an installation of the Diffusion server on the system you use to build the DAR file.

Procedure

1. Create a Maven project.

```
mvn archetype:generate -DgroupId=group_id -DartifactId=publisher_id
-DarchetypeArtifactId=maven-archetype-quickstart -
-DinteractiveMode=false
```

Replace *group_id* with the group identifier for your publisher, for example, `com.example`. Replace *publisher_id* with the artifact name for your publisher, for example, `my-first-publisher`.

This command creates a *publisher_id* directory that contains a `pom.xml` and sample files.

2. Replace the sample code in the new *publisher_id* Maven project with your publisher code. Put your Java code in the `publisher_id/src/main/java/` directory.
3. Add a `Publishers.xml` file to your Maven project.

Create the file at `publisher_id/src/main/diffusion/etc/Publishers.xml`:

```
<publishers>
  <publisher name="publisher_name">
    <class>fq_publisher_name</class>
    <start>true</start>
  </publisher>
</publishers>
```

Replace `publisher_name` with the name of the class that extends the `Publisher` class, for example, `MyFirstPublisher`. Replace `fq_publisher_name` with the fully qualified name of the class that extends the `Publisher` class, for example, `com.example.publish.MyFirstPublisher`.

4. Edit the `pom.xml` file in the `publisher_id` directory:

- a) Remove the boilerplate code and ensure that the group and artifact IDs are set to the correct values.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <groupId>group_id</groupId>
  <artifactId>publisher_id</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>publisher_id</name>

</project>
```

- b) Create a system-scoped dependency on the Diffusion JAR.

```
<dependencies>
  <dependency>
    <groupId>com.pushtechonology</groupId>
    <artifactId>diffusion</artifactId>
    <version>5.9.24</version>
    <scope>system</scope>
    <systemPath>diffusion_installation_directory/lib/
diffusion.jar</systemPath>
    <optional>true</optional>
  </dependency>
</dependencies>
```

- c) Add the Push Public Maven Repository as a repository that plugins can be fetched from

```
<pluginRepositories>
  <pluginRepository>
    <id>push-repository</id>
    <url>http://download.pushtechonology.com/maven/</url>
  </pluginRepository>
</pluginRepositories>
```

- d) Add `mvndar` as configured plugin

```
<build>
  <plugins>
    <plugin>
      <groupId>com.pushtechonology.tools</groupId>
      <artifactId>dar-maven-plugin</artifactId>
```

```
<version>1.2</version>
<extensions>>true</extensions>
</plugin>
</plugins>
</build>
```

- e) Set the packaging method to dar instead of jar:

```
<packaging>dar</packaging>
```

5. Build your DAR file.

Run the `mvn clean package` command in the `publisher_id` directory.

Results

A DAR file is created in the `publisher_id/target` directory. This DAR file is ready to deploy to the Diffusion server. For more information, see [Deploying publishers on your Diffusion server](#) on page 640

Related concepts

[Classic deployment](#) on page 640

Installing publishers into a stopped Diffusion instance.

[Hot deployment](#) on page 641

Installing publishers into a running Diffusion instance.

[Deployment methods](#) on page 641

There are two ways to deploy a DAR file: file copy or HTTP.

[Creating a Publisher class](#) on page 492

A publisher is written by extending the abstract `Publisher` class (see [Publisher API](#)) and overriding any methods that must be implemented to achieve the functionality required by the publisher.

[Loading publisher code](#) on page 484

This describes how to load publisher classes or code it is dependent upon.

Build server application code with Maven

The Diffusion API for server application code is not available in the Push Technology public Maven repository. To build server components, you must install the product locally and depend on `diffusion.jar` using a Maven system scope.

The following `pom.xml` shows to declare the dependency on `diffusion.jar`. To use it, you must set the `DIFFUSION_HOME` environment variable to the absolute file path of your Diffusion installation.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.examplecorp</groupId>
  <artifactId>mypublisher</artifactId>
  <version>1.0-SNAPSHOT</version>

  <dependencyManagement>
    <dependency>
      <groupId>com.pushtechnology.diffusion</groupId>
      <artifactId>diffusion</artifactId>
```

```

        <version>local-installation</version>
    </dependency>
</dependencyManagement>

<profiles>
  <profile>
    <activation>
      <property>
        <name>env.DIFFUSION_HOME</name>
      </property>
    </activation>

    <dependencyManagement>
      <dependencies>
        <dependency>
          <groupId>com.pushtechology.diffusion</
groupId>
          <artifactId>diffusion</artifactId>
          <version>local-installation</version>
          <scope>system</scope>
          <systemPath>${DIFFUSION_HOME}/lib/
diffusion.jar</systemPath>
        </dependency>
      </dependencies>
    </dependencyManagement>
  </profile>
</profiles>
</project>

```

Related concepts

[Classic deployment](#) on page 640

Installing publishers into a stopped Diffusion instance.

[Hot deployment](#) on page 641

Installing publishers into a running Diffusion instance.

[Deployment methods](#) on page 641

There are two ways to deploy a DAR file: file copy or HTTP.

[Building the demos using mvndar](#) on page 1072

You can use the Maven plugin *mvndar* with the provided `pom.xml` file to build the demos.

Testing

This section covers some aspects of testing a Diffusion system.

DEPRECATED: Flex/Flash client

The Flex client test tool is a Flex application that uses the `diffusion-flex.swc` library. The tool can be found in the Tools section in the root menu of your Diffusion installation.

This tool enables the user to supply connection information as well as credential information. Select which transport is going to be used to connect to the Diffusion server. This client can issue a server ping. The ping response displays the elapsed time, its average and the current queue size for this client.

This tool can also be used to test reconnection, auto failover, and load balancing. Available servers can be listed by IP in the URL box. The **Reconnect** button, after a disconnect, returns the client to the server it was originally connected to, and receive messages queued during disconnection. Selecting **Auto Failover** causes the client to connect to the next available server, after the existing connection is forcibly killed. Selecting **Cascade** causes the client to cascade through the available servers until it finds one to which it can connect. Select **Load Balancing** in conjunction with **Auto Failover**, and the client connects to one of a shuffled list of available servers.

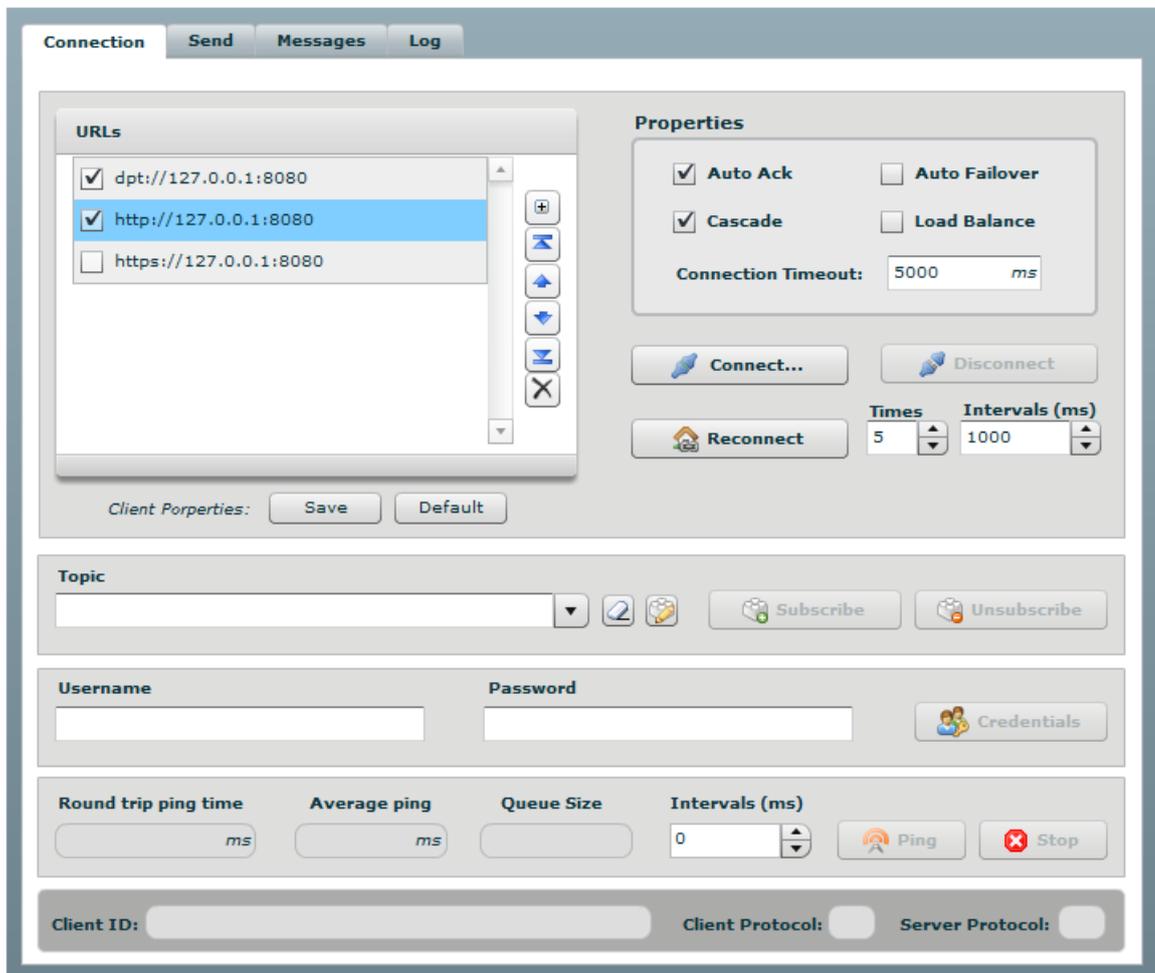


Figure 28: Flex client: Connection tab

You can pass user headers with a fetch request which are reflected back to the client in the response message as a Correlation ID. This allows the client to be able to associate replies with requests. The Flex client API has a new 'headers' parameter on the fetch methods. No changes are required to publishers to benefit from this.

Once connected to a Diffusion server, you can send messages to any topic. The publisher receives this message and notify the `messageFromClient` method in the publisher. The requested encoding can be set, as well as any user headers required. You can request an acknowledgment from the server.

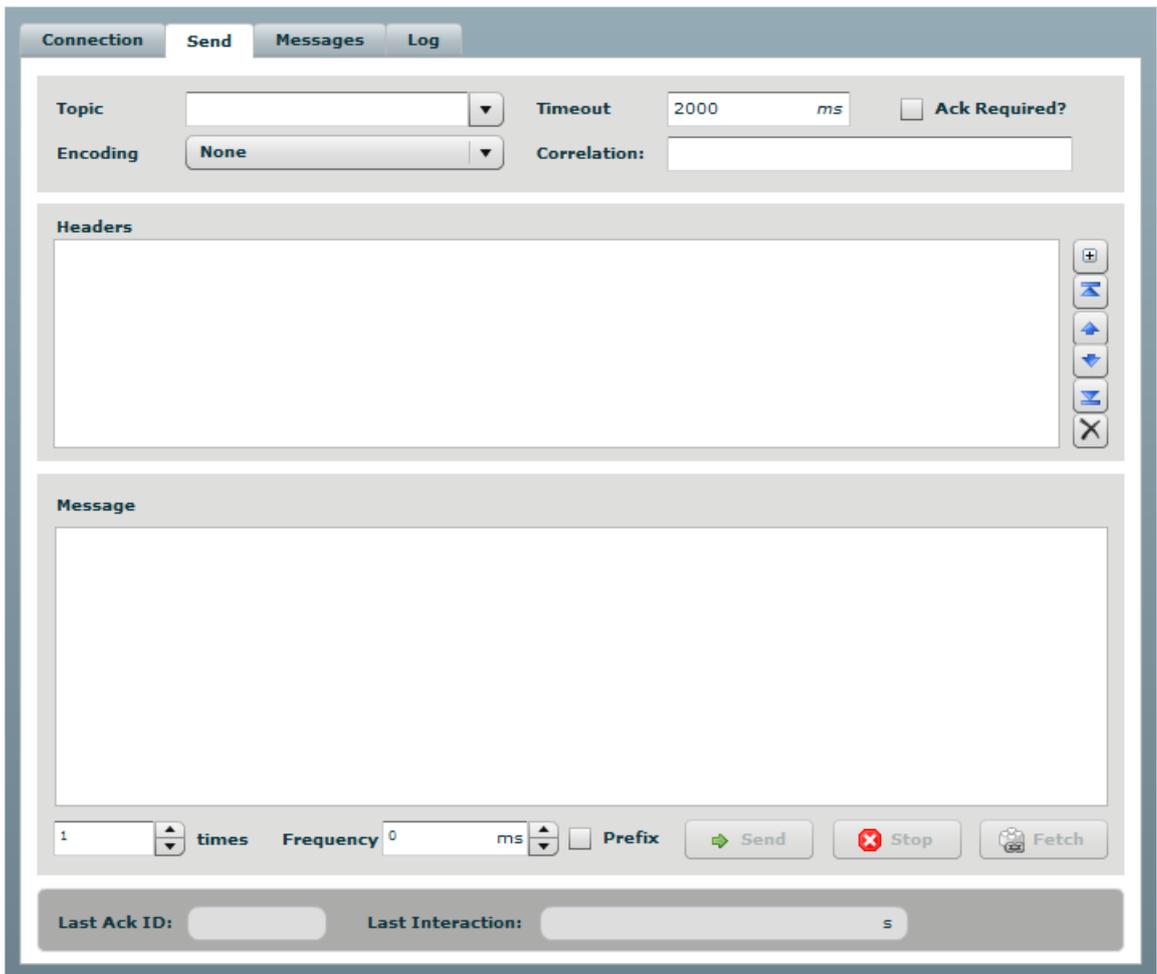


Figure 29: Flex client: Send tab

Messages are displayed in a tree format. It is not a true hierarchic tree, as each topic has its own node directly off the root node.

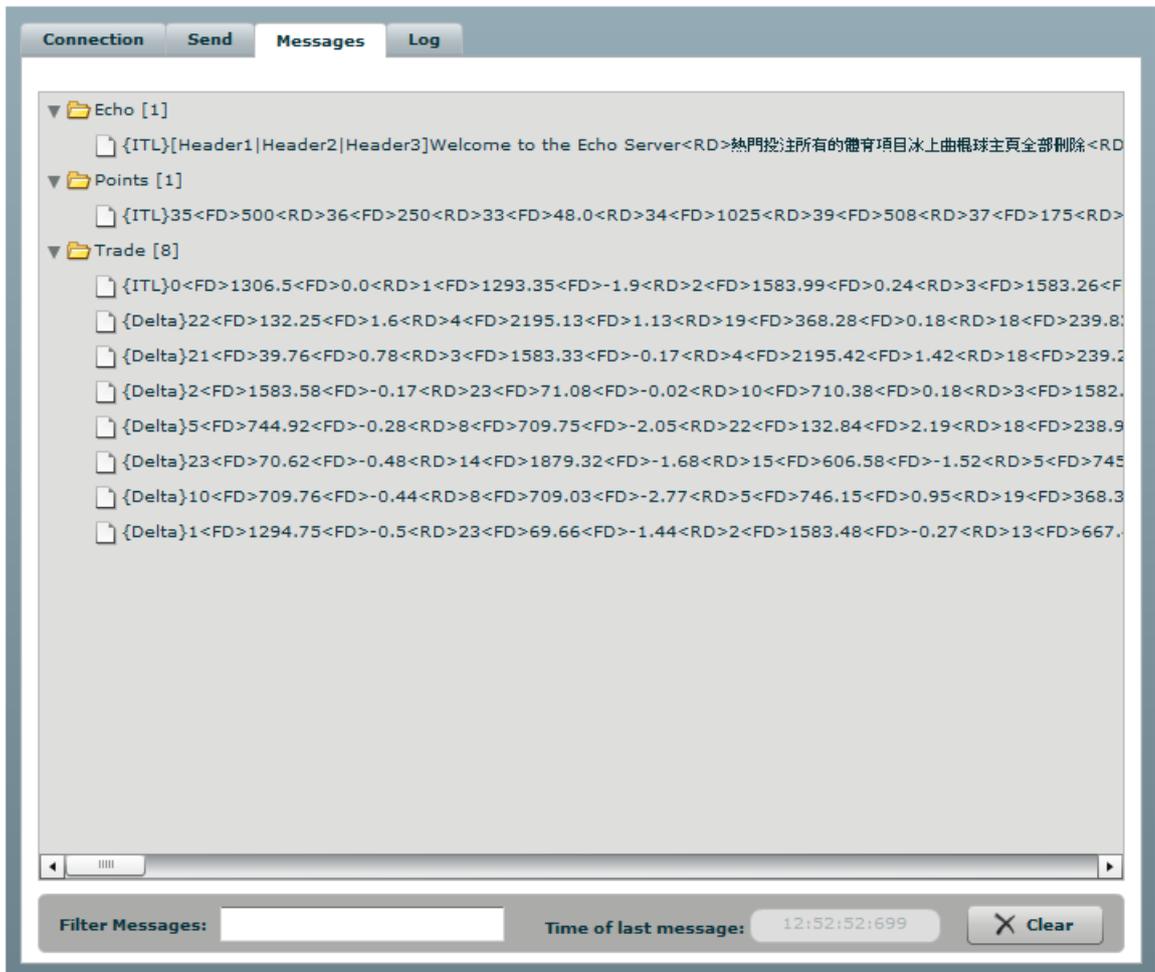


Figure 30: Flex client: Messages tab

The log tab shows messages that are sent from the Diffusion client when debugging is switched on.

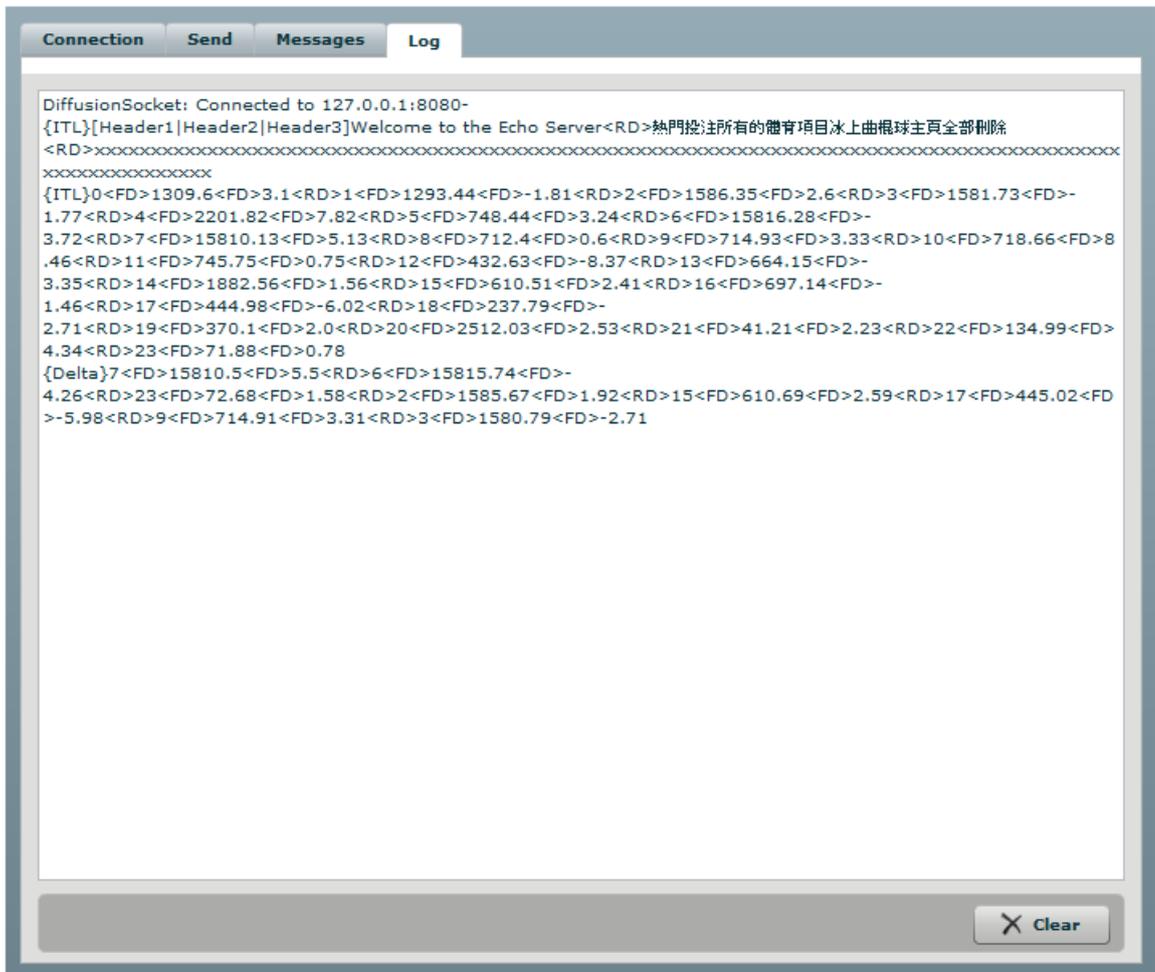


Figure 31: Flex client: Log tab

DEPRECATED: Java client test tool

The external client test tool is a swing application that enables the user to connect, using credentials if required to a Diffusion server.

The tool is found in the Tools folder in your Diffusion installation, and is run from the command line using either `extclient.bat` or `extclient.sh`.

It allows for many different connection types (for example, HTTP / DPTS, WebSocket). Once connected the user can subscribe and unsubscribe to topics or topic selectors. It is also possible to issue a server ping from this client. The ping response displays the elapsed time, the average and the current queue size for this client.

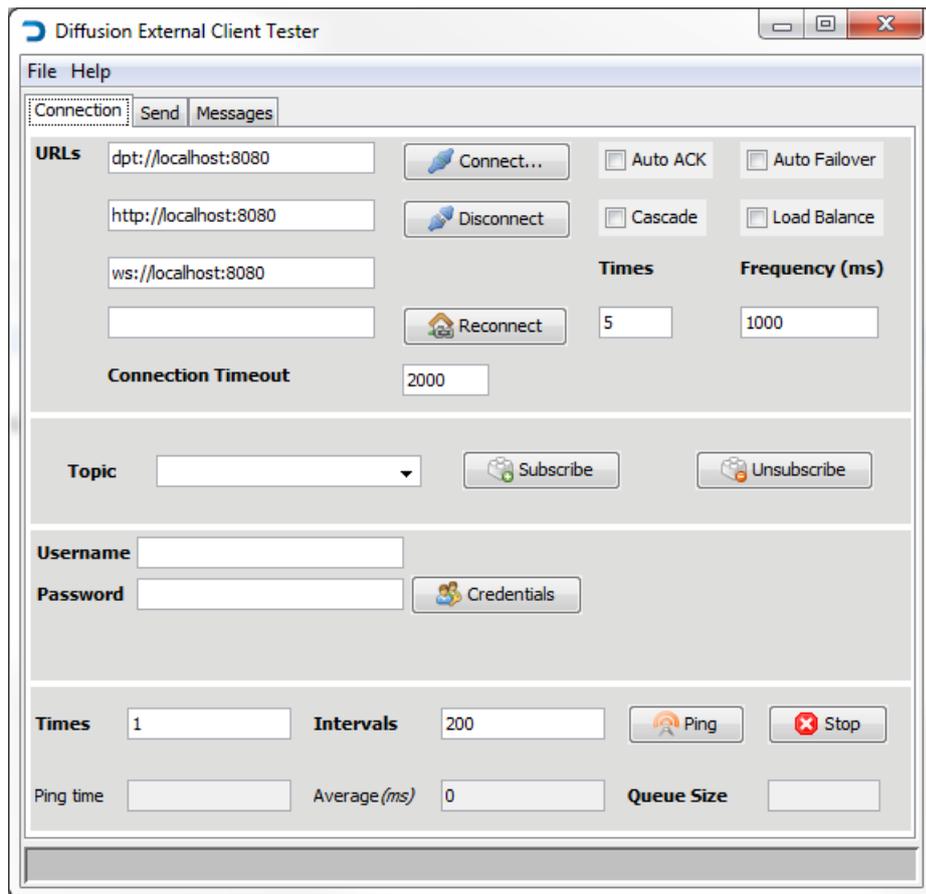


Figure 32: External client tester: Connection tab

You can connect to Diffusion securely. In order for the external client to work in secure mode, a certificate must be installed on the client machine. To aid in installing the certificate, there is an option under the file menu called Load Cert. This connects to Diffusion and detail instructions on how to install the newly created jssecerts file.

After subscribing to a topic, the client can send messages to that topic. The publisher receives this message and notify the `messageFromClient` method in the publisher. It is also possible to set the message encoding type and any user headers if required.

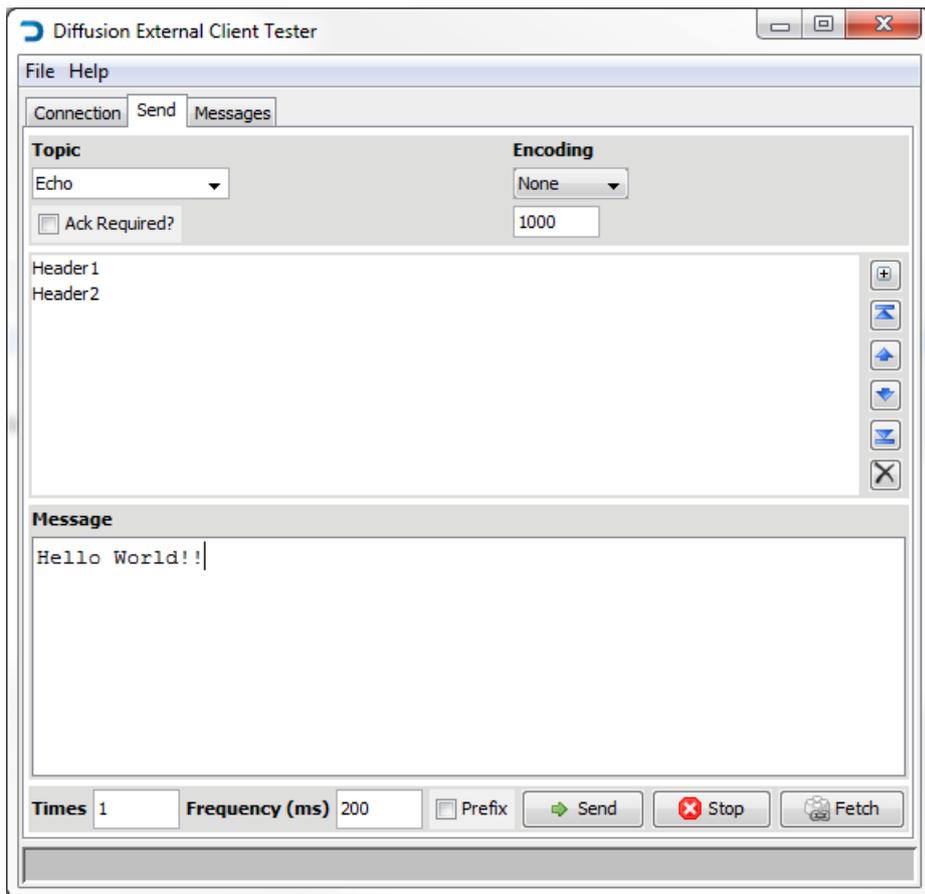


Figure 33: External client tester: Send tab

Messages are displayed in a tree format. It is not a true hierarchic tree, as each topic has its own node directly off the root node.

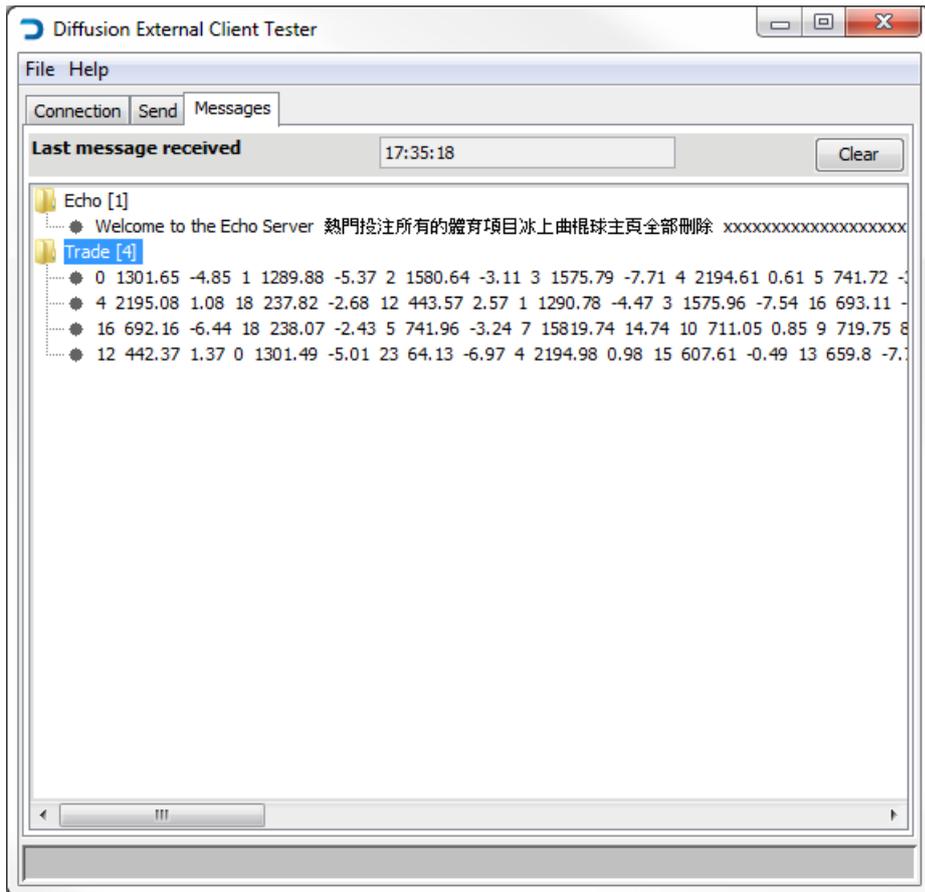


Figure 34: External client tester: Messages tab

It is also possible to examine a message by double clicking it

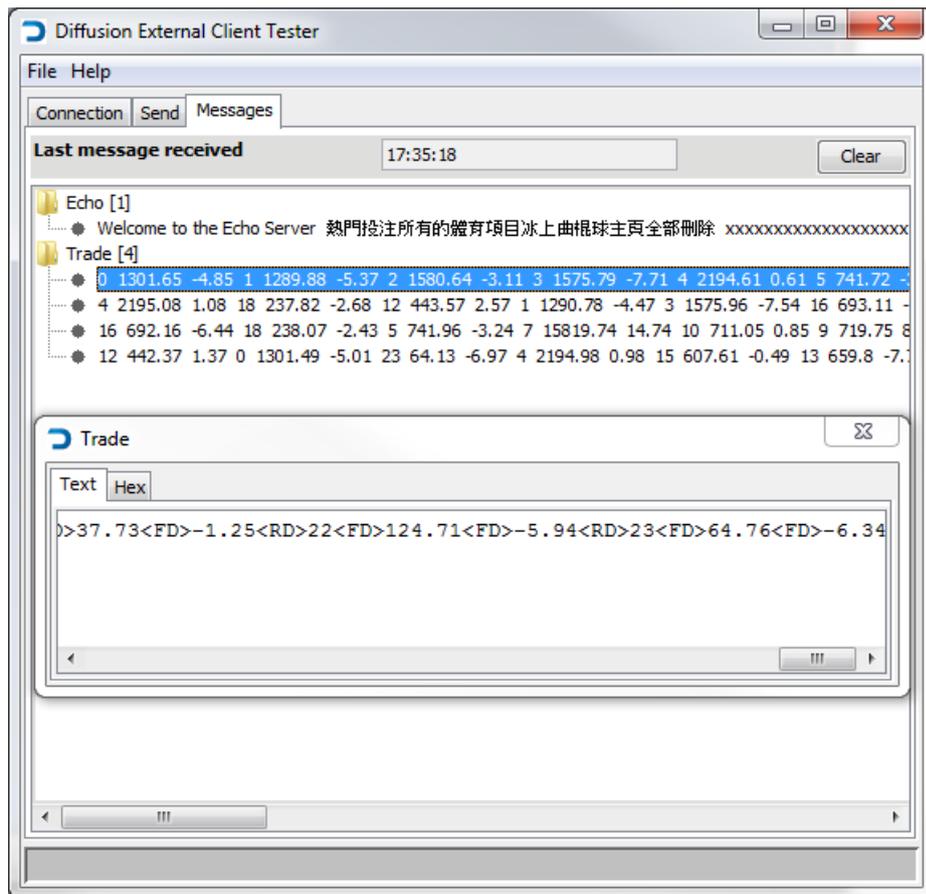


Figure 35: External client tester: Message details window

DEPRECATED: JavaScript client test tool

The JavaScript test tool allows the user to connect to a publisher and test the API.

The JavaScript test tool is browser based and can be found in the Root Menu of your Diffusion installation. It allows for different types of connections. The test tool also allows for Subscribing, Unsubscribing and Fetching against topic sets.

Fetch Correlation allows user headers to be passed with a fetch request which is reflected back to the client in the response message. This allows the client to be able to associate replies with requests. The JavaScript client API has a new headers parameter on the fetch methods. See API documentation for changes to the API specification. No changes are required to publishers to benefit from this.

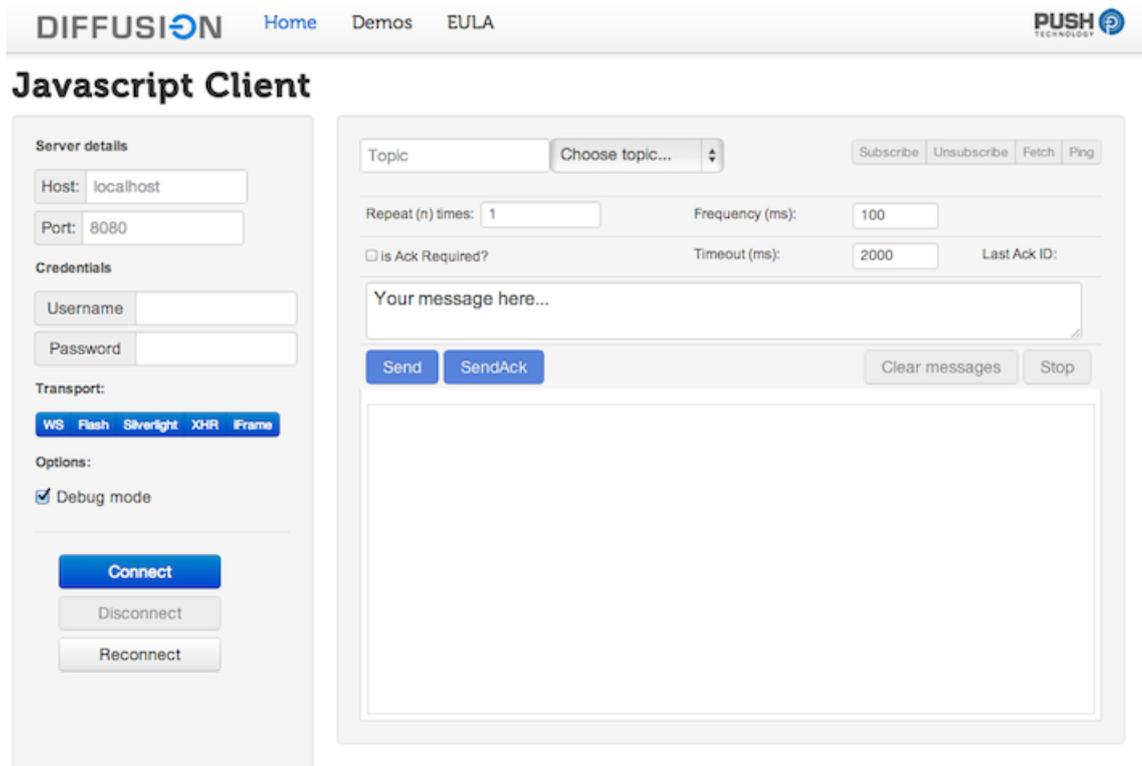


Figure 36: JavaScript test tool

Silverlight client test tool

The Silverlight client test tool is a Silverlight application that uses the `PushTechnology.Transports` assembly. This tool enables the user to perform various diagnostic functions without having to build a new Silverlight application.

The following screenshot shows the connection tab of the test tool:

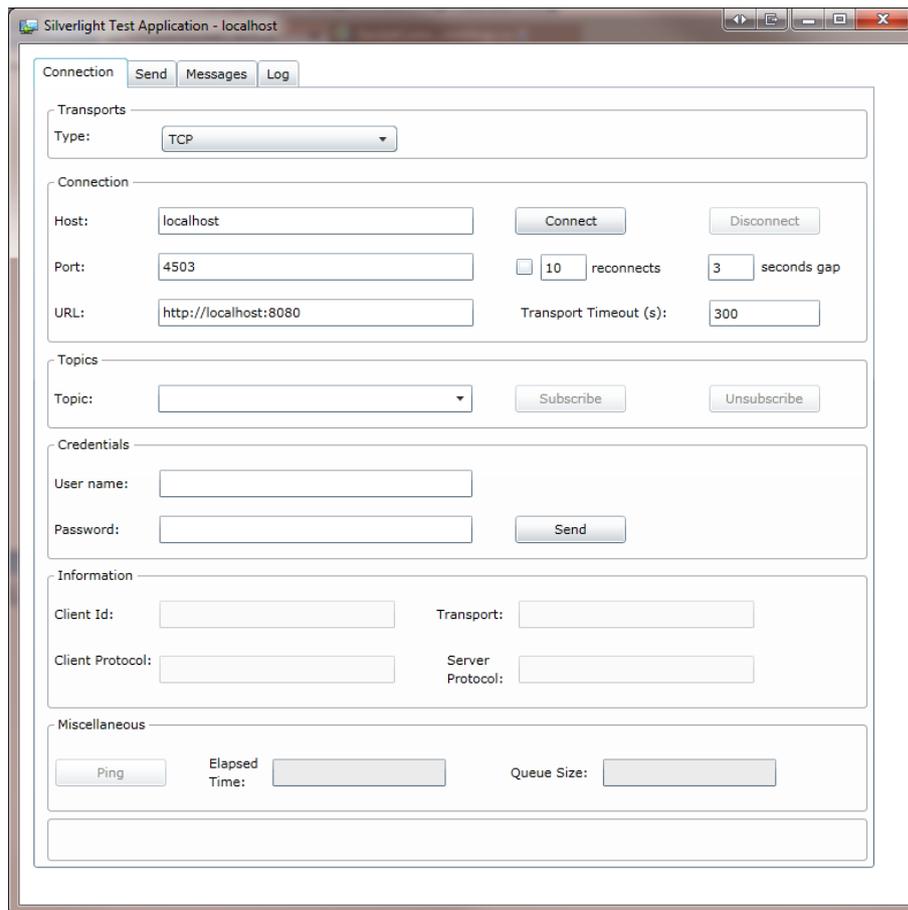


Figure 37: Silverlight test tool: Connection tab

Once connected to a Diffusion server, you can send messages to any subscribed topic. The Diffusion publisher then receives this message and notifies the `messageFromClient` method in the publisher. The requested encoding can be set, as well as any user headers.

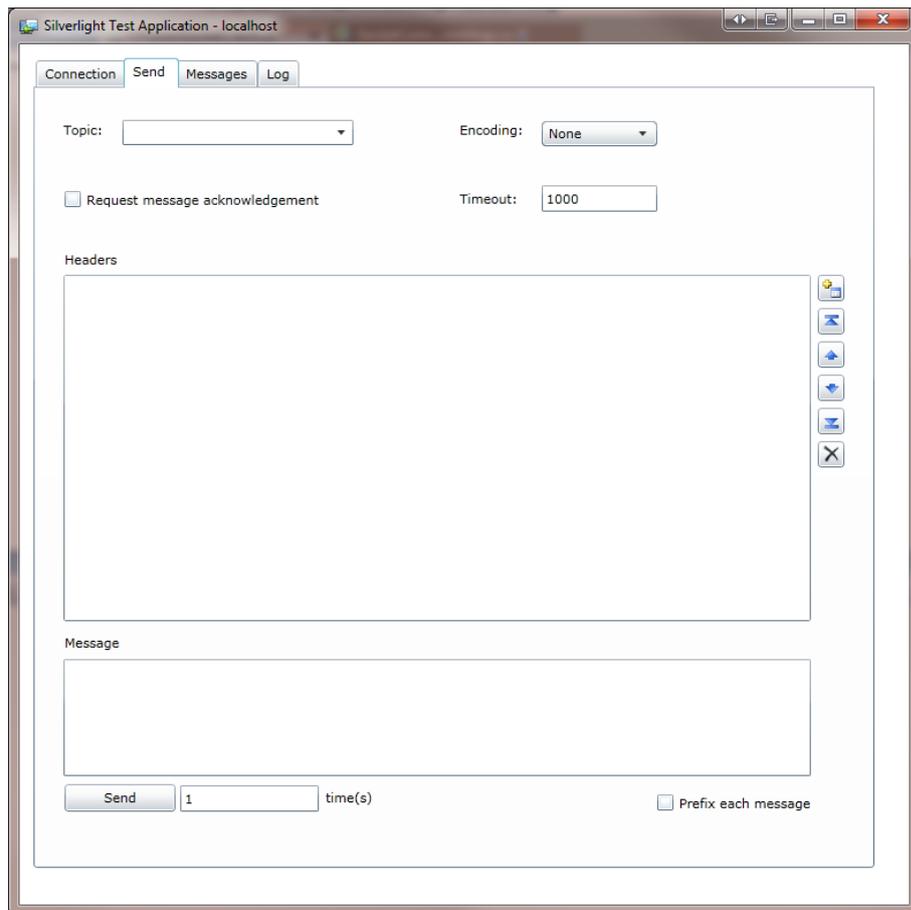


Figure 38: Silverlight test tool: Send tab

Received messages are displayed in a hierarchical format:

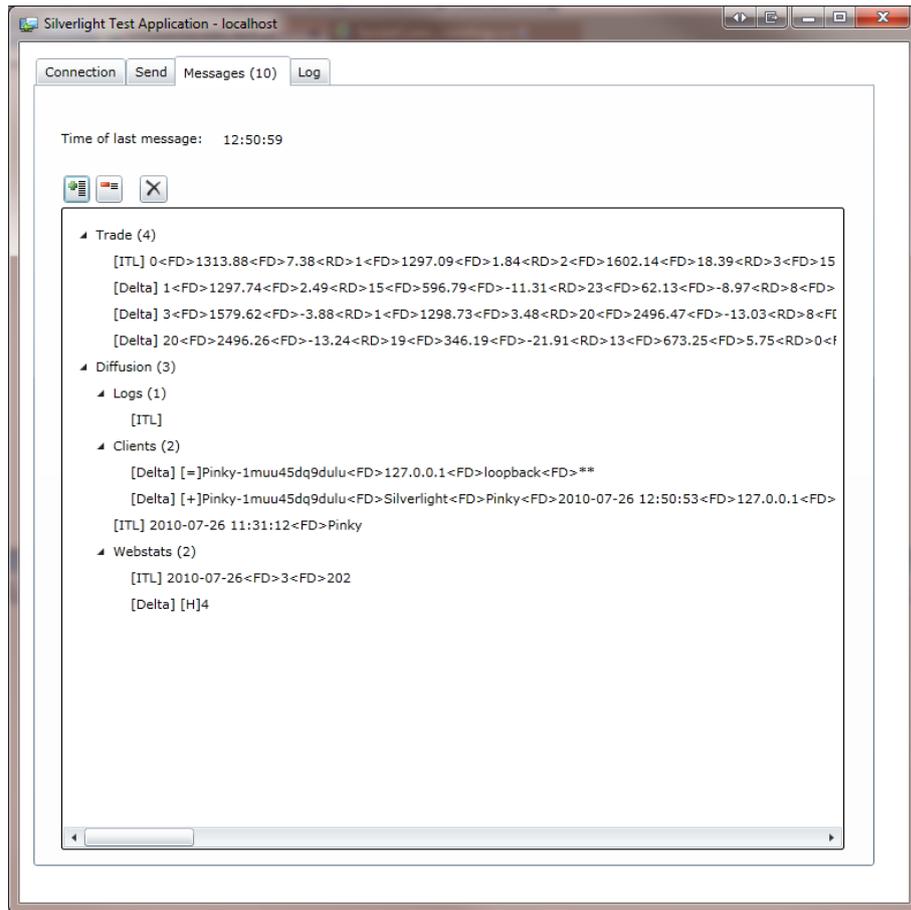


Figure 39: Silverlight test tool: Messages tab

Stress test tuning

Stress testing Diffusion requires various tuning changes to be made.

Table 53: Tuning changes for stress testing

Increase number of open files	If the Diffusion server is running in Linux, there is a limit to the number of open files/connections. See "Increasing Number of Open files".
Buffer sizes	Socket buffer sizes can be changed in <code>etc/Connectors.xml</code>
Inbound thread settings	The Inbound thread pool settings must be tuned otherwise the pool and queues get full. The settings are found in <code>etc/Server.xml</code> . Increase these values carefully and the server re-tested each time. Set the maximum size and queue size quite high for stress testing (for example, 250) so that the server can handle the hundreds or thousands of simultaneous connection attempts.

Client queues	Depending on the number and frequency of the messages, you might have to increase the client queue parameter.
Client multiplexers	By default, the number of client multiplexers configured is equal to the number of cores on the host system of the Diffusion server. You might have to increase the number of multiplexers as the client load on each Diffusion server increases.

Limitations

If more than 2000 clients are to be simulated, it is better to run the stress test from multiple machines. Most systems start to struggle when dealing with 2000 concurrent connections. By running the stress test on multiple machines any number of clients can be tested.

Socket exceptions can be encountered when a large number of clients are run on one machine. These can be thrown on either the Diffusion server or within the stress test tool. If this is occurring, the number of clients needs to be reduced

Stress test

Stress test is a good way of working out what the system requirements are going to be of a Diffusion server under duress. The stress test tool is issued to test the performance of Diffusion under stress and helps to understand how changes to the configuration can have an effect upon the performance.

Before you begin

Do not run the stress test tool on the same machine as the Diffusion server. A separate machine is required to run the stress test tool.

About this task

Installing the stress test tool;

Procedure

1. On the test machine, create a `stresstest` directory and copy the files `stresstest/stresstest.bat`, `stresstest/stresstest.sh`, `stresstest/stresstest.properties` and `stresstest/stresstest.jar` from the Diffusion installation
2. Configure stress test by editing the `stresstest.properties` file. The following need to be configured:
3. Number of clients
4. Transport type
5. Server details
6. Number of messages with which to run the test
7. The topic to which the clients subscribe
8. On a Windows system, launch `stresstest.bat`, on a UNIX machine launch `stresstest.sh`.

Benchmarking suite

A benchmarking suite for Diffusion is available on GitHub. You can use this suite to test the latency and throughput of publishers.

The benchmarking suite is available at the following location: <https://github.com/pushtechnology/diffusion-benchmark-suite>.

The benchmarking suite works on Linux only and requires the following software be installed on the system:

- Apache Ant™
- Java with JDK
- Diffusion server

For more information about using the benchmarking suite, see the readme file in the GitHub project.

Test tools

Test tools are provided which allow you to connect to a Diffusion server as an external client application. There are also tools that act as a publisher server application.

Generic Java versions are provided which work on any platform and Windows versions are provided specifically for use on Windows platforms.

Table 54: Testing tools

Java client	Provides a GUI interface simulating an external client connection using the Java API. Suitable for use on any platform.
Windows client	Provides a GUI interface simulating an external client connection using the Windows API. Suitable for use on Windows platforms only.
JavaScript client	Provides a GUI interface simulating a JavaScript client connection. Suitable for use on any platform.
Flex/Flash client	Provides a GUI interface simulating a Flash client connection. Suitable for use on any platform where Flash Player is installed.
Silverlight client	Provides a GUI interface simulating a Silverlight client connection. Suitable for use on any platform where Silverlight is installed.

Part V

Administrator Guide

This guide describes how to deploy, configure, and manage your Diffusion solution.

In this section:

- [Installing the Diffusion server](#)
- [Configuring your Diffusion server](#)
- [Starting the Diffusion server](#)
- [Deploying publishers on your Diffusion server](#)
- [Load balancers](#)
- [Web servers](#)
- [Push Notification Bridge](#)
- [JMS adapter](#)
- [Network security](#)
- [Going to production](#)
- [Managing and monitoring your running Diffusion server](#)
- [Tuning](#)
- [Demos](#)
- [Tools](#)

Installing the Diffusion server

You can install the Diffusion server from a JAR file, through Docker, or through Red Hat Package Manager.

Review the system requirements before installing Diffusion.

Download Diffusion from the following location: <http://download.pushtechnology.com/releases/5.9>

The Diffusion installation includes a developer license that allows up to five concurrent connections to the Diffusion server. To use Diffusion in production, you can obtain a production license from [Sales at Push Technology](#).

System requirements for the Diffusion server

Review this information before installing the Diffusion server.

The Diffusion server is certified on the system specifications listed here. In addition, the Diffusion server is supported on a further range of systems.

Certification

Push Technology classes a system as certified if the Diffusion server is fully functionally tested on that system.

We recommend that you use certified hardware, virtual machines, operating systems, and other software when setting up your Diffusion servers.

Support

In addition, Push Technology supports other systems that have not been certified.

Other hardware and virtualized systems are supported, but the performance of these systems can vary.

More recent versions of software and operating systems than those we certify are supported.

However, Push Technology can agree to support Diffusion on other systems. For more information, contact Push Technology.

Physical system

The Diffusion server is certified the following physical system specification:

- Intel Xeon E-Series Processors
- 8 Gb RAM
- 8 CPUs
- 10 Gigabit NIC

Network, CPU, and RAM (in decreasing order of importance) are the components that have the biggest impact on performance. High performance file system and disk are required. Intel hardware is used because of its ubiquity in the marketplace and proven reliability.

Virtualized system

The Diffusion server certified on the following virtualized system specification:

Host

- Intel Xeon E-Series Processors

- 32 Gb RAM
- VMware vSphere 5.5

Virtual machine

- 8 VCPUs
- 8 Gb RAM

Operating system

Diffusion is certified on the following operating systems:

- Red Hat 6.5, 6.6, and 7.2
- Windows Server 2012 R2

We recommend you install your Diffusion server on a Linux-based operating system with enterprise-level support available, such as Red Hat Enterprise Linux.

Operating system configuration

If you install your Diffusion server on a Linux-based operating system and do SSL offloading of secure client connections at the Diffusion server, you must disable transparent huge pages.

If you install your Diffusion server on a Linux-based operating system but do not do SSL offloading of secure client connections at the Diffusion server, disabling transparent huge pages is still recommended.

Having transparent huge pages enabled on the system your Diffusion server runs on can cause extremely long pauses for garbage collection. For more information, see <https://access.redhat.com/solutions/46111>.

Java

The Diffusion server is certified on Oracle Java 8 64-bit JDK

Only the Oracle JDK is certified.

Ensure that you use the Oracle JDK and not the JRE.

JVM configuration

If you do SSL offloading of secure client connections at the Diffusion server, you must ensure that you constrain the maximum heap size and the maximum direct memory size so that together these values do not use more than 80% of your system's RAM.

Networking

Push Technology recommends the following network configurations:

- 10 Gigabit network
- Load balancers with SSL offloading
- In virtualized environments, enable SR-IOV.

For more information about how to enable SR-IOV, see the documentation provided by your virtual server provider. SR-IOV might be packaged using a vendor-specific name.

Client requirements

For information about the supported client platforms, see [Platform support for the Diffusion Unified API libraries](#) on page 47.

Related concepts

[The Diffusion license](#) on page 543

Diffusion includes a development license that enables you to use make up to 5 concurrent connections to the Diffusion server.

[Installed files](#) on page 546

After installing Diffusion the following directory structure exists:

Related tasks

[Installing the Diffusion server using the graphical installer](#) on page 537

The Diffusion binary files are available from the Push Technology website. You can install Diffusion using the graphical installer.

[Installing the Diffusion server using the headless installer](#) on page 539

The Diffusion binary files are available from the Push Technology website. You can install Diffusion from the command line.

[Installing the Diffusion server using Red Hat Package Manager](#) on page 540

Diffusion is available as an RPM file from the Push Technology website.

[Installing the Diffusion server using Docker](#) on page 541

Diffusion is available as a Docker® image from Docker Hub.

[Verifying the Diffusion installation](#) on page 548

Start your Diffusion server, review the logs, and connect to the console to verify that your installation is correct.

Installing the Diffusion server using the graphical installer

The Diffusion binary files are available from the Push Technology website. You can install Diffusion using the graphical installer.

Before you begin

You must have Java 8 installed on your system to install and use Diffusion.

About this task

To install Diffusion using the graphical installer, complete the following steps:

Procedure

1. Go to the Diffusion download page:
<http://download.pushtechnology.com/releases/5.9>
2. Click on the following download links to download the required jar files into a temporary directory:
 - Diffusion (`Diffusion version_id.jar`)
 - Installer (`install.jar`)
3. In the temporary directory, double-click the `install.jar` file.
The graphical installer launches.
4. Optional: If you have a production license, you can load it into the Diffusion installation at this point.
You can skip this step if you are using the included development license.
 - a) Ensure that the license file is available on your system.
 - b) At the **Introduction** step, select **File > Load license file**
 - c) In the window that opens, navigate to the license file (`licence.lic`). Click **Open**.

5. At the **Introduction** step, click **Continue**.
6. At the **License agreement** step, select **Accept** to accept the End User License Agreement (EULA) and click **Continue**.
7. At the **Destination directory** step, select the install destination.
We recommend you create a `Diffusion` directory on your system. Click **Continue**.
8. At the **Select products** step, select the components you want to install.
We recommend you select **All**. Click **Continue**.
9. At the **Confirmation** step, review the install information. If the information is correct, click **Continue** to confirm.
The installer installs Diffusion into the directory specified.
10. At the **Summary** step, click **Done** to exit the graphical installer.

Results

You have successfully downloaded and installed Diffusion.

What to do next

Next:

- Edit the configuration of your Diffusion server to suit your requirements. For more information, see [Configuring your Diffusion server](#) on page 550.
- Edit the security setup of your Diffusion server.
- Start your Diffusion server using the `diffusion.bat` file, if on Windows, or the `diffusion.sh` file, if on Linux or OS X/macOS.

These start up scripts are located in the `bin` directory of your Diffusion installation.

Related concepts

[The Diffusion license](#) on page 543

Diffusion includes a development license that enables you to use make up to 5 concurrent connections to the Diffusion server.

[Installed files](#) on page 546

After installing Diffusion the following directory structure exists:

Related tasks

[Installing the Diffusion server using the headless installer](#) on page 539

The Diffusion binary files are available from the Push Technology website. You can install Diffusion from the command line.

[Installing the Diffusion server using Red Hat Package Manager](#) on page 540

Diffusion is available as an RPM file from the Push Technology website.

[Installing the Diffusion server using Docker](#) on page 541

Diffusion is available as a Docker® image from Docker Hub.

[Verifying the Diffusion installation](#) on page 548

Start your Diffusion server, review the logs, and connect to the console to verify that your installation is correct.

Related reference

[System requirements for the Diffusion server](#) on page 45

Review this information before installing the Diffusion server.

Installing the Diffusion server using the headless installer

The Diffusion binary files are available from the Push Technology website. You can install Diffusion from the command line.

Before you begin

You must have Java 8 installed on your system to install and use Diffusion.

About this task

You can install in headless mode in circumstances where the graphical installer cannot be used or is not appropriate.

Procedure

1. Go to the Diffusion download page:
<http://download.pushtechnology.com/releases/5.9>
2. Click on the following download links to download the required jar files into a temporary directory:
 - Diffusion (`Diffusion version_id.jar`)
 - Installer (`install.jar`)
3. Copy these files to a temporary directory on the system where Diffusion is to be installed.
4. In the terminal window, change to the directory where the Diffusion jar files are located.
5. Type the following command:

```
java -jar install.jar Diffusionn.n.n.jar
```

where *n.n.n* is the Diffusion release number.

6. If you agree to the terms of the license agreement, type `Y` and Enter.
7. Enter the full path to the directory in which to install Diffusion and type Enter.
8. Type `Y` to install all packages.
If you choose not to install all packages, the installer asks you about each package individually.

Results

You have successfully downloaded and installed Diffusion.

What to do next

Next:

- Edit the configuration of your Diffusion server to suit your requirements. For more information, see [Configuring your Diffusion server](#) on page 550.
- Edit the security setup of your Diffusion server.
- Start your Diffusion server using the `diffusion.bat` file, if on Windows, or the `diffusion.sh` file, if on Linux or OS X/macOS.

These start up scripts are located in the `bin` directory of your Diffusion installation.

Related concepts

[The Diffusion license](#) on page 543

Diffusion includes a development license that enables you to use make up to 5 concurrent connections to the Diffusion server.

[Installed files](#) on page 546

After installing Diffusion the following directory structure exists:

Related tasks

[Installing the Diffusion server using the graphical installer](#) on page 537

The Diffusion binary files are available from the Push Technology website. You can install Diffusion using the graphical installer.

[Installing the Diffusion server using Red Hat Package Manager](#) on page 540

Diffusion is available as an RPM file from the Push Technology website.

[Installing the Diffusion server using Docker](#) on page 541

Diffusion is available as a Docker® image from Docker Hub.

[Verifying the Diffusion installation](#) on page 548

Start your Diffusion server, review the logs, and connect to the console to verify that your installation is correct.

Related reference

[System requirements for the Diffusion server](#) on page 45

Review this information before installing the Diffusion server.

Installing the Diffusion server using Red Hat Package Manager

Diffusion is available as an RPM file from the Push Technology website.

About this task

On Linux systems that have Red Hat Package Manager installed, you can use it to install Diffusion.

Procedure

1. Go to the Diffusion download page:
<http://download.pushtechnology.com/releases/5.9>
2. Click on the following download link to download the required RPM file:
 - Diffusion RPM (`diffusion-n.n.n_build.noarch.rpm`)
3. Copy this file to a temporary directory on the system where Diffusion is to be installed.
4. In the terminal window, change to the directory where the Diffusion RPM file is located.
5. Type the following command:

```
rpm -ivh diffusion-n.n.n_build.noarch.rpm
```

where *n.n.n* is the Diffusion release number and *build* is an additional string containing numbers to represent the build level.

Results

Diffusion is installed in the following directory: `/opt/Diffusion`. A startup script is installed in the `/etc/init.d` directory that enables Diffusion to start when you start the system.

What to do next

Your Diffusion installation includes a development license that allows connections from up to five clients. To use Diffusion in production, you can obtain a production license from [Sales at Push Technology](#).

Copy the license file into the `/etc` directory of your Diffusion installation.

Related concepts

[The Diffusion license](#) on page 543

Diffusion includes a development license that enables you to use make up to 5 concurrent connections to the Diffusion server.

[Installed files](#) on page 546

After installing Diffusion the following directory structure exists:

Related tasks

[Installing the Diffusion server using the graphical installer](#) on page 537

The Diffusion binary files are available from the Push Technology website. You can install Diffusion using the graphical installer.

[Installing the Diffusion server using the headless installer](#) on page 539

The Diffusion binary files are available from the Push Technology website. You can install Diffusion from the command line.

[Installing the Diffusion server using Docker](#) on page 541

Diffusion is available as a Docker® image from Docker Hub.

[Verifying the Diffusion installation](#) on page 548

Start your Diffusion server, review the logs, and connect to the console to verify that your installation is correct.

Related reference

[System requirements for the Diffusion server](#) on page 45

Review this information before installing the Diffusion server.

Installing the Diffusion server using Docker

Diffusion is available as a Docker® image from Docker Hub.

Before you begin

You must have Docker installed on your system to run Diffusion from a Docker image. For more information, see <https://docs.docker.com/userguide/>.

About this task

You can use Docker to install the Diffusion server, and a minimal complete set of its dependencies, on a Linux system. This image contains a Diffusion server with a trial license and default configuration and security.

Using Docker enables you to install the Diffusion server in an isolated and reproducible way.

Procedure

1. Pull the latest version of the Diffusion image.

```
docker pull pushtechnology/docker-diffusion:latest
```

2. Run the image.

```
docker run -p 8080:8080 image_id
```

Where *image_id* is the ID of the image to run. Port 8080 is the port that is configured to allow client connections by default.

Results

Diffusion is now running in a container on your system. Clients can connect through port 8080.

Note: This Diffusion instance contains well known security principals and credentials. Do not use it in production without changing these values.

Related concepts

[The Diffusion license](#) on page 543

Diffusion includes a development license that enables you to use make up to 5 concurrent connections to the Diffusion server.

[Installed files](#) on page 546

After installing Diffusion the following directory structure exists:

Related tasks

[Installing the Diffusion server using the graphical installer](#) on page 537

The Diffusion binary files are available from the Push Technology website. You can install Diffusion using the graphical installer.

[Installing the Diffusion server using the headless installer](#) on page 539

The Diffusion binary files are available from the Push Technology website. You can install Diffusion from the command line.

[Installing the Diffusion server using Red Hat Package Manager](#) on page 540

Diffusion is available as an RPM file from the Push Technology website.

[Verifying the Diffusion installation](#) on page 548

Start your Diffusion server, review the logs, and connect to the console to verify that your installation is correct.

Related reference

[System requirements for the Diffusion server](#) on page 45

Review this information before installing the Diffusion server.

Next steps with Docker

The Diffusion image on Docker Hub includes the default configuration, default security, and trial license. Additional steps are required to secure and configure the Diffusion server.

Procedure

1. Create a Dockerfile that contains commands to configure a Diffusion image for your use.
 - a) Base your Docker image on the Diffusion image.

```
FROM pushtechnology/docker-diffusion:latest
```

- b) To use Diffusion in production, obtain a production license from [Sales at Push Technology](#).
The default Diffusion image includes a development license that allows connections from up to five clients.

- c) Copy the production license into the `/opt/Diffusion/etc` directory of your Diffusion image.

```
ADD license_file /opt/Diffusion/etc/licence.lic
```

Where `license_file` is the path to the production license relative to the location of the Dockerfile.

- d) Create versions of the Diffusion configuration files that define your required configuration.
For more information, see [Configuring your Diffusion server](#) on page 550.
- e) Copy these configuration files into the `/opt/Diffusion/etc` directory of your Diffusion image.

```
ADD configuration_file /opt/Diffusion/etc/file_name
```

Where `configuration_file` is the path to the configuration file relative to the location of the Dockerfile and `file_name` is the name of the configuration file.

- f) Create versions of the `Security.store` and `SystemAuthentication.store` that define roles, principals and authentication actions for your security configuration.

For more information, see [Pre-defined roles](#) on page 138, [DSL syntax: security store](#) on page 420, and [DSL syntax: system authentication store](#) on page 408.

You can instead choose to edit these files using a Diffusion client. However, your Diffusion server is not secure for production use until you do so.

- g) Copy these store files into the `/opt/Diffusion/etc` directory of your Diffusion image.

```
ADD store_file /opt/Diffusion/etc/file_name
```

Where `store_file` is the path to the store file relative to the location of the Dockerfile and `file_name` is the name of the store file.

- h) Include any additional configuration actions you want to perform on your image in the Dockerfile.

2. Build your image.

Run the following command in the directory where your Dockerfile is located:

```
docker build .
```

Results

The image you created contains a configured Diffusion server ready for you to use in your solution. You can run multiple identically configured Diffusion servers from this image.

The Diffusion license

Diffusion includes a development license that enables you to use make up to 5 concurrent connections to the Diffusion server.

To use Diffusion in production, contact [Sales at Push Technology](#) for production licenses.

Related concepts

[Installed files](#) on page 546

After installing Diffusion the following directory structure exists:

Related tasks

[Installing the Diffusion server using the graphical installer](#) on page 537

The Diffusion binary files are available from the Push Technology website. You can install Diffusion using the graphical installer.

[Installing the Diffusion server using the headless installer](#) on page 539

The Diffusion binary files are available from the Push Technology website. You can install Diffusion from the command line.

[Installing the Diffusion server using Red Hat Package Manager](#) on page 540

Diffusion is available as an RPM file from the Push Technology website.

[Installing the Diffusion server using Docker](#) on page 541

Diffusion is available as a Docker® image from Docker Hub.

[Verifying the Diffusion installation](#) on page 548

Start your Diffusion server, review the logs, and connect to the console to verify that your installation is correct.

Related reference

[System requirements for the Diffusion server](#) on page 45

Review this information before installing the Diffusion server.

License restrictions

The Diffusion license can include restrictions on how the Diffusion server is used.

Environments

A Production license must not be used on a Development server, and a Development license must not be used on a Production server. Order separate licenses defined as Production, QA/Testing, Disaster Recovery, and Development.

License expiry

All license files provided by Push Technology include an expiry date. To continue to use Diffusion after this date you must replace your license file with an updated license file.

The Diffusion server logs the number of days remaining on your license every day at midnight and when the server starts ([PUSH-000202](#) on page 856).

When the license has expired, the Diffusion server stops working within 24 hours. A message is logged when the license expires ([PUSH-000203](#) on page 857).

Concurrent client connections

An instance of the Diffusion server is licensed to only allow up to a certain number of client connections at the same time.

A license can include a soft limit and a hard limit on concurrent client connections. When the soft limit is reached, the Diffusion server logs a message ([PUSH-000201](#) on page 855) to say that the soft limit has been reached. When the hard limit is reached, the Diffusion server logs a message ([PUSH-000204](#) on page 857) to say that the hard limit has been reached. No further client connections can be made to the Diffusion server. Subsequent client connection attempts are refused and a message is logged ([PUSH-000204](#) on page 857).

MAC addresses or IP addresses

An instance of the Diffusion server can be licensed to run only on systems with a certain range of IP addresses or MAC addresses.

on startup, the Diffusion server checks the IP address or MAC address of the system the server runs on. If the Diffusion server cannot read the IP or MAC address of the host system, it logs a message ([PUSH-000207](#) on page 859 or [PUSH-000208](#) on page 859) and does not start. If the IP or MAC address of the host system is not in the licensed address range, the server logs a message ([PUSH-000200](#) on page 855 or [PUSH-000209](#) on page 860) and does not start.

Diffusion version

A Diffusion license can be valid for specific versions of Diffusion only.

If you use a license file with a version of Diffusion that it is not valid for, the Diffusion server logs a message ([PUSH-000199](#) on page 854) and does not start.

Related reference

[PUSH-000056](#) on page 814

User licence limit breached, connection rejected.

[PUSH-000199](#) on page 854

License is not valid for this version of Diffusion (license='{}' vs '{}').

[PUSH-000200](#) on page 855

Product '{}' not licensed for this address scheme."

[PUSH-000201](#) on page 855

Soft user limit ({} for {} exceeded. Hard limit at {}.

[PUSH-000202](#) on page 856

Product license expires in {} day(s).

[PUSH-000203](#) on page 857

License has expired.

[PUSH-000204](#) on page 857

License hard user limit ({} for {} exceeded.

[PUSH-000207](#) on page 859

License check failed : Unable to check server-side addressing for product [IP] '{}'.

[PUSH-000208](#) on page 859

License check failed : Unable to check server-side addressing for product [MAC] '{}'.

[PUSH-000209](#) on page 860

License check failed : Unable to match MAC address for [{}].

Updating your license file

You can update your Diffusion license file without having to restart the Diffusion server. Copy the new file over the old and ensure that the timestamp is updated.

Before you begin

Obtain a new or renewed license file from Push Technology.

About this task

When your license file expires, the Diffusion server continues to run for another day before it stops. We recommend you update your license file before your existing license file expires.

Procedure

1. Copy the new license file (`licence.lic`) over the existing file in the `diffusion_directory/etc` directory.

You do not need to stop or restart the server.

2. Check that the timestamp of `licence.lic` has updated.
 - On Windows, you might have to use the following command to copy the file over and force the timestamp to update: `COPY /B licence.lic +,,`
3. Diffusion checks the timestamp of the `licence.lic` every minute. If the license file has been updated, Diffusion reloads it and logs this to stdout.
4. You can verify that the license file has been updated in the server by accessing the mbean **`com.pushtechology.diffusion > Server > LicenseExpiryDate`**

Installed files

After installing Diffusion the following directory structure exists:

Table 55: Installed files

Folder name	Contents
<code>bin</code>	Executables for starting Diffusion
<code>clients</code>	Client Diffusion API libraries and related artifacts for all supported platforms.
<code>data</code>	Files used by publishers, the console, and third-party components. This directory is always on the server classpath. However, the <code>ext</code> directory is the preferred place to store resource files that are loaded by publishers.
<code>demos</code>	The compiled DAR files and source code for the demos issued with Diffusion. For more information, see Demos on page 1071.
<code>deploy</code>	Publisher DAR files that are deployed when the Diffusion server starts. If you selected during the install process to deploy the demos, the demo DAR files are in this directory.
<code>docs</code>	License information, release notes, and install notes.
<code>etc</code>	Diffusion configuration files and example policy files for Silverlight and Flash. For more information, see Configuring your Diffusion server on page 550.
<code>examples</code>	Example code that uses the Diffusion APIs.
<code>ext</code>	This directory, together with any jar files in this directory or subdirectories, are available through the classloader used to deploy application code to the Diffusion server. You can add library jar files to this directory that are required by application code such as publishers and local authentication handlers.
<code>html</code>	Files that are used by the default web server for issuales accessible through the browser.

Folder name	Contents
lib	The main Diffusion server JAR file, third-party libraries, and additional server-side components.
logs	The directory to which Diffusion server and web server logs are written.
pushnotifications	The directory that contains the Push Notification Bridge and its associated files.
stresstest	The stress test package. For more information, see Stress test tuning on page 531.
tools	Tools and utilities that help with testing and deploying Diffusion. For more information, see Tools and utilities on page 547
xsd	The schema files for the XML configuration files used by the server.

Tools and utilities

The following table describes the some of the contents of the `tools` directory.

Note: The files present and their suffixes vary according to the platform that the product is installed on.

Table 56: Tools and utilities

Tool	Description
<code>/ec2</code>	A sample configuration for setting up the Diffusion server in an Amazon™ EC2 instance.
<code>/init.d</code>	Sample <code>init.d</code> files to start the Diffusion server as daemon on macOS, Linux, or UNIX systems.
<code>/joyent</code>	A sample configuration for setting up the Diffusion server in a Joyent™ instance.
<code>extclient.bat/sh</code> and <code>tools.jar</code>	Generic external client test tool .
<code>externalclienttest.properties</code>	External client test tool properties.
<code>war.xml</code>	Example <code>war.xml</code> file
<code>web.xml</code> and <code>sun-web.xml</code>	Example <code>web.xml</code> files

Related concepts

[The Diffusion license](#) on page 543

Diffusion includes a development license that enables you to use make up to 5 concurrent connections to the Diffusion server.

Related tasks

[Installing the Diffusion server using the graphical installer](#) on page 537

The Diffusion binary files are available from the Push Technology website. You can install Diffusion using the graphical installer.

[Installing the Diffusion server using the headless installer](#) on page 539

The Diffusion binary files are available from the Push Technology website. You can install Diffusion from the command line.

[Installing the Diffusion server using Red Hat Package Manager](#) on page 540

Diffusion is available as an RPM file from the Push Technology website.

[Installing the Diffusion server using Docker](#) on page 541

Diffusion is available as a Docker® image from Docker Hub.

[Verifying the Diffusion installation](#) on page 548

Start your Diffusion server, review the logs, and connect to the console to verify that your installation is correct.

Related reference

[System requirements for the Diffusion server](#) on page 45

Review this information before installing the Diffusion server.

Verifying the Diffusion installation

Start your Diffusion server, review the logs, and connect to the console to verify that your installation is correct.

About this task

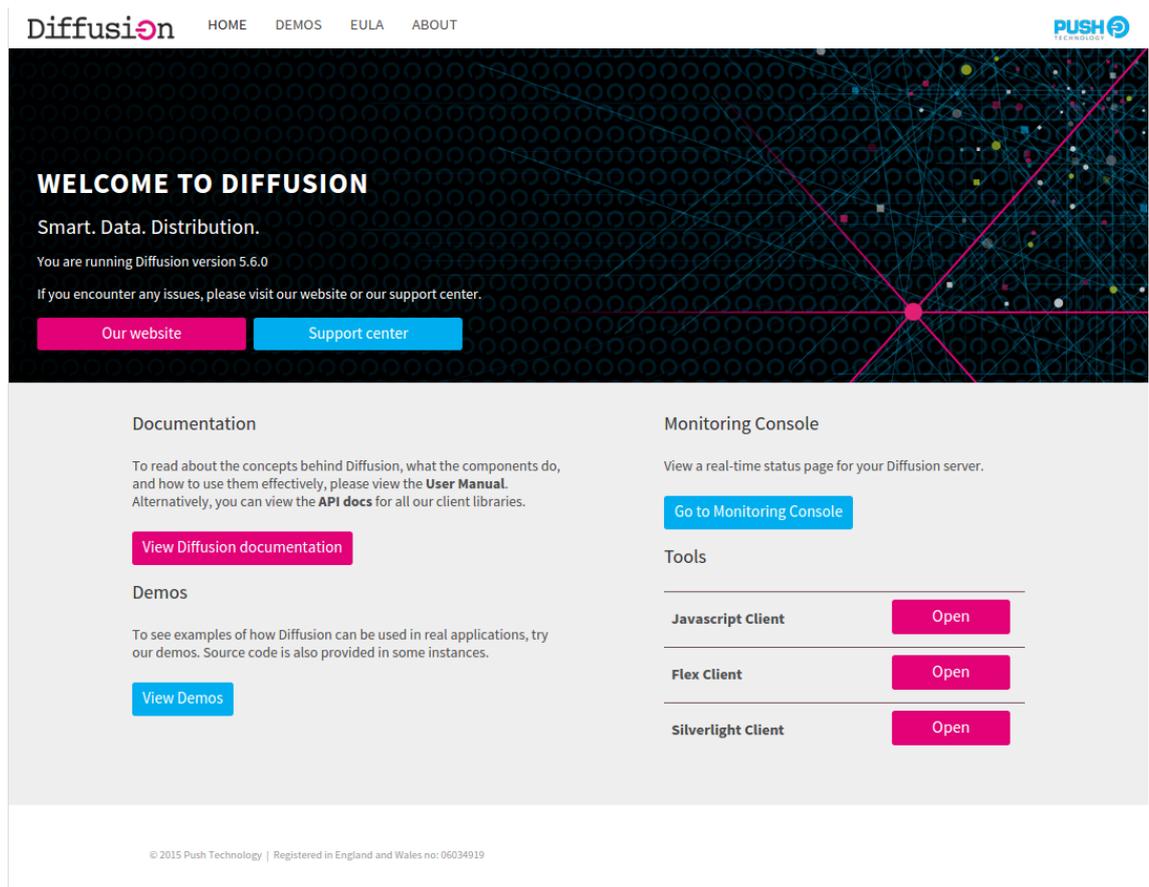
After installation, all of the Diffusion files are available in the directory specified during installation.

Procedure

1. Start the Diffusion server using one of the start script located in the `bin` directory of your Diffusion installation.
 - On Windows, use the `diffusion.bat` file.
 - On Linux, macOS, or UNIX, use the `diffusion.sh` file.
2. Inspect the log messages to ensure that the Diffusion server started successfully.

The terminal window displays logging information about the status of the Diffusion server. A log message containing the following text indicates that the server started successfully: `INFO|main|PUSH0165|Diffusion Server started.|com.pushtechnology.diffusion.DiffusionController` This line is typically the last one to be printed on terminal.
3. Inspect all log messages displayed in the terminal to search for `WARN` messages to ensure that all components have started correctly.
4. Open a browser and navigate to `http://serverAddress:8080` (or `http://localhost:8080`)

The browser shows the Diffusion landing page.



The landing page provides links to information regarding legal terms and conditions (for example, EULA), user guides, API documentation and demos.

The Diffusion server is ready to be used.

5. If you chose to install the demos, you can access them from the landing page. Use these demo publishers to verify your installation.

Related concepts

[The Diffusion license](#) on page 543

Diffusion includes a development license that enables you to use make up to 5 concurrent connections to the Diffusion server.

[Installed files](#) on page 546

After installing Diffusion the following directory structure exists:

Related tasks

[Installing the Diffusion server using the graphical installer](#) on page 537

The Diffusion binary files are available from the Push Technology website. You can install Diffusion using the graphical installer.

[Installing the Diffusion server using the headless installer](#) on page 539

The Diffusion binary files are available from the Push Technology website. You can install Diffusion from the command line.

[Installing the Diffusion server using Red Hat Package Manager](#) on page 540

Diffusion is available as an RPM file from the Push Technology website.

[Installing the Diffusion server using Docker](#) on page 541

Diffusion is available as a Docker® image from Docker Hub.

Related reference

[System requirements for the Diffusion server](#) on page 45

Review this information before installing the Diffusion server.

Configuring your Diffusion server

You can configure the Diffusion server using XML files which normally reside in the `etc` directory. You can also configure user security on the Diffusion server using the `.store` files in the `etc` directory.

Alternatively, a Diffusion server can be instantiated in a Java application and configured programmatically. Some properties can also be changed at runtime programmatically from within publishers.

In a Java client environment certain properties can also be configured programmatically.

All properties (whether configured from XML or programmatically) are available to read programmatically from within the Java API.

XML configuration

Configuring a Diffusion server using XML property files

XML Property files

A Diffusion server is configured using a set of XML property files typically loaded from the `etc` folder. In a new Diffusion installation example versions of these files are provided which can be edited as required.

XML is used rather than standard property files due to the hierarchic nature and the ability to support repeating groups.

The Introspector has a built in configuration editor, which is able to load and save the configuration files remotely if required.

XSD files are issued that define the content of the XML property files and this section summarizes the XSD content.

Configuration path loading

You can pass a parameter to Diffusion upon startup so that files are not automatically loaded from the `etc` folder but loaded from a different folder. This folder does not have to contain the complete set of XML files, but the file is loaded from the specified folder first, if it exists. If it does not, Diffusion loads the configuration file from the `etc` folder. When Diffusion starts, it logs where each configuration file has been loaded from.

XML Value types

When XML values are loaded, the schema is checked so that we know that it is valid, but to aid configuration, there are some extra data types. When values are loaded, they are trimmed of leading and trailing white space.

Table 57: XML Value types

Data type	Meaning
push:boolean	true or false
push:string	String value
push:int	A number between -2,147,483,648 and 2,147,483,647
push:long	A number between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807
push:double	A number between 2-1074 and (2-2-52)-†.21023
push:port	A positive number but less than 65535
push:millis	<p>A string that represents the number of milliseconds. Append the mnemonic for the time unit. The mnemonic can be either upper or lower case.</p> <p>s Seconds</p> <p>m Minutes</p> <p>h Hours</p> <p>d Days</p> <p>360000, 360s, 6m all represent 6 minutes</p>
push:bytes	<p>A string that represents the number of bytes. Append the mnemonic size unit. The mnemonic can be either upper or lower case.</p> <p>k Kilobytes</p> <p>m Megabytes</p> <p>g Gigabytes</p> <p>6291456, 6144k, 6m, all represent 6 Megabytes</p>
push:log-level	A log level can be ERROR, WARN, INFO, DEBUG, or TRACE.
push:percent	A value that represents a percentage, this can have the trailing percent sign (%)
push:positiveNonZeroInt	A number between 1 and 2,147,483,647
push:positiveInt	A number between 0 and 2,147,483,647
push:positiveNonZeroLong	A number between 1 and 9,223,372,036,854,775,807
push:positiveLong	A number between 0 and 9,223,372,036,854,775,807
<element>	This notation is used to indicate a complex element type. It can also be List<element> to indicate a repeating property group.

Environmental values

When defining custom configurations, you can define environmental variables that can be reused in all XML property files. These variables can be defined in the `etc/Env.xml` property file to be used in all other property files. Suppose, for example, the `etc/Env.xml` file defines a `server-name` variable, with value `d-unit` as follows:

```
<env>
  <property name="server-name">d-unit</property>
</env>
```

The `server-name` variable can be used in all other property files, where the value `d-unit` is appropriate, either as a value for an attribute, as in

```
<server name={server-name}>,Ä¶</server>
```

or as a name for an element as in:

```
<server>{server-name}</server>
```

As a side remark, it is worth noting that names can be combined to provide malleable environmental variables. Suppose for instance `Env.xml` contains the following entries:

```
<env>
  <property name="server-name">myServer</property>
  <property name="server-version">V2.0</property>
</env>
```

Then `server-name` and `server-version` can be combined, for instance within the same `etc/Env.xml`, as

```
<property name="server-and-version">{server-name}-{server-version}</property>
```

and used in all other configuration files.

Obfuscated values

Obfuscation is a technique through which sensitive entries can be hidden in clear text. Within the Diffusion context, obfuscation can be used to hide password, and any other data deemed to be sensitive, to be included in configuration files.

To create obfuscated values you can use the **Property Obfuscator** dialog in the Diffusion plugin for Eclipse™. Obfuscated entries are identified by a `OB:` prefix in clear text.

Related concepts

[Property obfuscator](#) on page 784

This dialog is part of the Diffusion perspective and can be used to hide sensitive Diffusion configuration file entries, such as passwords and JMS login credentials.

Programmatic configuration

An alternative to configuring a Diffusion server using XML property files is to instantiate a Diffusion server within a Java application and configure it programmatically before starting it.

If desired, some properties can be loaded from XML files and some supplied programmatically or default properties can be bootstrapped from XML files and overridden programmatically before the server is started.

Most server properties can be configured only before the server is started. Instantiate the server within an application and configure before starting the server. However, certain configuration items (examples being conflation and connection policies) can be configured at any time during the life of the server. The API documentation makes it clear if a property can be changed at runtime.

Because the properties that can be set programmatically reflect those that can be set in XML this section does not describe the properties in detail. The XSD property descriptions or the API documentation for the configuration API can be consulted for full details.

As well as allowing configuration properties to be set the configuration API also allows all properties that can be configured to be read at runtime. So publisher code has direct access to all property settings.

Related information

<https://docs.pushtechology.com/docs/5.9.8/java-classic/index.html?com/pushtechology/diffusion/api/config/package-summary.html>

Using the configuration API

General use

From within a Java application the root of the configuration tree can be obtained at any time using `ConfigManager.getConfig()`. This provides access to the general objects and can be used from within server-side or client-side code.

From within server-side code (for example, a publisher) the server configuration root can be obtained using `ConfigManager.getServerConfig()` which exposes all of the server side configuration also.

From the configuration root you can navigate to any subordinate configuration objects to view them or set their properties.

On the server side most properties cannot be changed after the server has started and they become locked so any attempt to change them results in an exception. Certain properties (such as conflation and connection policies) can be changed at runtime. The API documentation makes it clear which properties can be changed at runtime.

In client-side Java code the configuration does not become locked and can be changed at any time. However, some values are read at the start only. Ideally, set all properties before creating any client side objects.

For configuration objects which are optional but there can be many (multiplicity 0..n), there are appropriate add methods to add new objects. For example to add a publisher and set a property on it:

```
PublisherConfig publisher =
```

```
ConfigManager.getServerConfig().addPublisher(  
    "MyPublisher",  
    "com.pub.MyPublisher");  
publisher.setTopicAliasing(false);
```

In these cases there are also methods to obtain the full list (for example, `getPublishers()`) or to obtain a specific one by name (for example, `getPublisher("MyPublisher")`). In many cases there are also methods to remove an object.

Note: When there must be at least one object (multiplicity 1..n), you must configure at least one. However, if a server is started with missing configuration of this kind, suitable defaults are normally created and a warning logged.

Single instance configuration objects (multiplicity 1..1) subordinate to the root can be obtained so that their properties can be changed (or read). So, for example the `Queues` object (an instance of `QueuesConfig`) can be obtained using the `getQueues()` method.

When a single configuration object is optional (multiplicity 0..1), the `get` method can return null if it has not been defined. In this case to set it the `set` method (as opposed to `add`) returns the object created. An example of this is the file service (`FileServiceConfig`) on a web server (`WebServerConfig`) as shown in the following example code:

```
ServerConfig config = ConfigManager.getServerConfig();  
WebServerConfig webServer = config.addWebServer("MyWebServer");  
FileServiceConfig fileService = webServer.setFileService("File  
Service");
```

Configuring a server

After instantiating a Diffusion server in Java the root of the server configuration tree can be obtained from the server object itself and configuration objects can be navigated to and changed as required before starting the server.

For example, the following code shows how to add a connector that accepts client connections on port 9090:

```
DiffusionServer server = new DiffusionServer();  
ServerConfig config = server.getConfig();  
ConnectorConfig connector = config.addConnector("Client Connector");  
connector.setPort(9090);  
connector.setType(Type.CLIENT);  
server.start();
```

In reality, it is best to configure far more values. However, if any essential objects are omitted (such as queues), suitable defaults are created when the server starts and a warning is logged.

Configuration access from a publisher

Within a publisher the configuration object for the publisher itself can be obtained using the `getConfig` method which returns the publisher configuration (`PublisherConfig`) object.

Related information

<https://docs.pushtechology.com/docs/5.9.8/java-classic/index.html?com/pushtechology/diffusion/api/config/package-summary.html>

Configuring the Diffusion server

Use the `Server.xml` configuration file to configure the core behaviors of the Diffusion server.

Configuring fan-out

Configure the the Diffusion server to act as a client to one or more other Diffusion servers and replicate topics from those servers.

Use the `fanout` section of the `Server.xml` configuration files to define client connections for this secondary server to make to one or more primary servers and the topics on those primary servers to replicate locally.

Each `fanout-connection` element represents a client connection that your Diffusion server makes to another Diffusion server in your solution.

```
<fanout>
  <connection>
    <url>ws://primary_server_hostname:8080</url>
    <principal>client</principal>
    <password>password</password>
    <retry-delay>1000</retry-delay>
    <reconnect-timeout>60s</reconnect-timeout>
    <recovery-buffer-size>1024</recovery-buffer-size>
    <input-buffer-size>1024k</input-buffer-size>
    <output-buffer-size>1024k</output-buffer-size>
    <link><selector>?topic_path//</selector></link>
  </connection>
</fanout>
```

Connection

Use the `url` element to specify the URL of the primary server and the transport and port used for the connection.

Permissions

When connecting to another Diffusion server as a client, this secondary server can provide a principal and associated password. If a principal is not provided, the secondary server connects anonymously

To subscribe to topics on the primary server and replicate them locally, the secondary server's client session must have the `select_topic` and `read_topic` permissions for those topics. Ensure that the principal this secondary server uses is assigned a role with the appropriate permissions on the primary server. If the secondary server connects anonymously to the primary server, ensure anonymous sessions on the primary server are assigned the appropriate permissions.

Reconnection

Use the `retry-delay` element to specify the time in milliseconds between the connection or reconnection attempts that the secondary server makes to the primary server.

Use the `reconnect-timeout` element to specify the maximum time in milliseconds that the secondary server will attempt to reconnect to its existing session on the primary server after a disconnection. If this element is not specified, a value of 0 is assumed and reconnection is not attempted.

If the secondary server is configured to attempt to reconnect, it keeps a buffer of messages sent to the primary server. Use the `recovery-buffer-size` element to configure the size of this buffer.

Replicating topics

Each `fanout-connection` has one or more `link` elements. Each `link` element uses a topic selector to specify a set of topics on the primary server to replicate on this secondary server.

Note: The set of topics specified by a link cannot overlap the set of topics specified by any other link within either this `fanout-connection` or any of the others.

If you want missing topic handlers registered on the primary server to receive missing topic notifications when a subscription or fetch request is made on the secondary server to a part of the topic tree that matches a link selector, consider the following when configuring your secondary server links:

- Avoid using regular expressions in the selectors you use to configure when setting up fan-out links on the secondary server. Topic selectors containing regular expressions increase the likelihood of false negatives and false positives when propagating missing topic notifications.
- Ensure that the principal that the secondary server uses to make the fan-out connection to the primary server has the `SELECT_TOPIC` permission for the path prefix of the selector that triggered the missing topic notification.

For more information, see [Using missing topic notifications with fan-out](#) on page 102.

Configuring your primary server

The primary server in a fan-out configuration must be configured to handle serving the topics replicated by fan-out to the secondary server or servers.

Ensure that the primary server connector that the secondary server or servers connect to has a large enough queue to handle the number of primary server topics that will be replicated by fan-out. In the `Connectors.xml` file, inside the `<connector>` element that defines the connector used for fan-out connections, set the queue depth to greater than the number of fanned out topics:

```
<queue-definition>depth</queue-definition>
```

To allow the secondary server to reconnect, enable reconnection on the connector that the primary server uses to accept connections from the secondary server or servers. Ensure that the reconnection timeout (`keep-alive`) value for the connector is long enough to allow the secondary server time to reconnect. Set the maximum queue depth and recovery buffer sizes to values that are appropriate to the volume of messages you expect to occur between the primary and secondary servers.

For more information, see [Connectors.xml](#) on page 583.

Topic aliasing

Topic aliasing is not supported with fan-out. Ensure that it is disabled at the primary server.

In the primary server `Publishers.xml` file, set `topic-aliasing` as `false` for any publishers that create topics that are fanned out.

```
<topic-aliasing>false</topic-aliasing>
```

When starting the primary server ensure that `diffusion.publishers.v5.topic.aliasing.disabled` is set to `true`. Edit the `diffusion.sh` or `diffusion.bat` file to set it as a system property when starting the Diffusion server:

```
-Ddiffusion.publishers.v5.topic.aliasing.disabled=true
```

Related concepts

[Fan-out](#) on page 99

Consider whether to use fan-out to replicate topic information from primary servers on one or more secondary servers.

Configuring conflation

Use the `conflation` section of the `Server.xml` configuration file to define one or more conflation policies, which describe how the conflation is to be done, and map topics to those policies.

Note: Conflation policies can also be configured programmatically using the `addPolicy` methods on `ConflationConfig`. Such policies can also be added dynamically after the Diffusion server has started.

Conflation policies

One or more conflation policies can be configured, each defining different conflation mechanisms by using `conflation-policy` elements. Conflation policies comprise the following:

Table 58: Conflation policy elements

Property	Description
name	A unique name by which the policy is referred to.
mode	Indicating whether the new (or merged) message is to replace the current message in place or whether the current message is to be removed and the new one appended to the end of the queue.
matcher	A Java class which matches two messages and is used to locate an existing queued message as a candidate for conflation. If no matcher is specified then default matching finds a message that is of the same topic.
merger	A Java class which performs the merge of two messages of the same topic to produce a new message containing the data from both messages (or any resulting data required). If no merger is specified, no merging takes place and the current message is removed from the queue and the new message either replaces it or is appended to the queue depending upon the mode. The merger can also indicate that either the current or new message is to be used or even that no conflation takes place in this instance.

Having defined one or more conflation policies, you can map topics to them. This is done by specifying a topic name or a topic selection string (regex pattern) which maps to a particular conflation policy.

Conflation policies can be added or removed at runtime and the removal of a conflation policy automatically removes any mappings to it.

Conflation policy mode

The conflation policy mode determines whether the new (or merged) message is to replace the existing message in the client queue or be appended to the end of the client queue.

Available modes are:

Table 59: Conflation policy modes

Mode	Definition
REPLACE	The new (or merged) message will replace the existing message at its current position in the client queue.
APPEND	The current message is removed from the client queue and the new (or merged) message is appended to the end of the queue.

If no mode is specified, REPLACE is assumed.

The mode is specified in the `mode` property of a `conflation-policy` section in `etc/Server.xml`.

When defining conflation policies programmatically the mode is specified when creating the policy.

Message matchers

A message matcher is used by a conflation policy when queuing a new message for a client that has conflation enabled for a topic that has a conflation policy defined for it. The message matcher is used to locate the last message queued for a client that is a candidate for conflation.

If no message matcher is explicitly defined for a conflation policy, a default matcher is used which locates a message of the same topic.

A message matcher can be supplied if the matching is to be somehow dependent upon the content of the messages.

To implement a message matcher, write a Java class that implements `com.pushtechology.diffusion.api.conflation.MessageMatcher`. This has a single method called `matches` to which is passed the current message in the queue being tested and the new message to be queued.

Note: The existing message is always of the same topic as the new message so you do not have to check that is the case.

An example of a `MessageMatcher` implementation is shown below:

```
public class ExampleMessageMatcher implements messageMatcher {
    @Override
    public boolean matches(TopicMessage currentMessage, TopicMessage
newMessage) {
        return
currentMessage.nextField().equals(newMessage.nextField());
    }
}
```

`MessageMatcher` implementations must be thread safe and stateless. The same `MessageMatcher` instance can be supplied to more than one different conflation policy if so required.

Message mergers

A message merger can be specified on a conflation policy if the action of the policy is to merge the content of an existing queued message with the new message being queued. This technique can be used when message data comprises more than one data item and it is desirable to reduce the number of messages sent to the client whilst preserving the data from all messages.

If no message merger is specified for a conflation policy, the policy replaces the current message with the new.

To implement a message merger, write a Java class that implements `com.pushtechnology.diffusion.api.conflation.MessageMerger`. This has a single method called `merge` to which is passed the current message in the queue and the new message to be queued.

Note: The existing message is always of the same topic as the new message so you do not have to check that is the case.

The action of conflation will depend upon the message that is returned from the `merge` method, as follows:

Table 60: Action depending upon merge result

Returned message	Action
A new message	It is assumed that the new message represents a merging of the data of the two messages input and so the returned message either replaces the current message in the queue or the current message is removed and the returned message added to the end of the queue, depending upon the policy mode.
The current message	The current message is retained at its current queue position and the new message is not queued.
The new message	The new message either replaces the current message in the queue or the current message is removed and the new message appended to the end of the queue depending upon the policy mode. This is effectively the same as the result that occurs if there is no merger.
Null	No conflation will occur. The current message remains where it is in the queue and the new message is appended to the end of the queue,

An example of a message merger implementation is shown below:

```
package com.pushtechnology.diffusion.examples;

import com.pushtechnology.diffusion.api.message.MessageReader;
import com.pushtechnology.diffusion.api.message.Record;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.pushtechnology.diffusion.api.APIException;
import com.pushtechnology.diffusion.api.conflation.MessageMerger;
import com.pushtechnology.diffusion.api.message.TopicMessage;
import com.pushtechnology.diffusion.api.publisher.Publishers;

import java.util.ArrayList;
import java.util.List;

public final class MessageMergerExample implements MessageMerger {
    private static final Logger LOG =
        LoggerFactory.getLogger(MessageMergerExample.class);

    @Override
    public TopicMessage merge(TopicMessage currentMessage,
        TopicMessage newMessage) throws APIException {

        final MessageReader currentMessageReader =
            currentMessage.getReader();
```

```

        final MessageReader newMessageReader =
newMessage.getReader();

        final TopicMessage result =
Publishers.createDeltaMessage(currentMessage.getTopicName());

        Record cRecord = currentMessageReader.nextRecord();
        Record nRecord = newMessageReader.nextRecord();

        while (nRecord != null) {
            final List<String> mergedRecord = new
ArrayList<>(nRecord.size());
            for (int i = 0; i < nRecord.size(); i++) {
                final String nField = nRecord.getField(i);
                if (!nField.isEmpty() || cRecord == null || i >=
cRecord.size()) {
                    mergedRecord.add(nField);
                }
                else {
                    mergedRecord.add(cRecord.getField(i));
                }
            }

            result.putRecord(mergedRecord);

            nRecord = newMessageReader.nextRecord();
            cRecord = currentMessageReader.nextRecord();
        }

        if (LOG.isTraceEnabled()) {
            LOG.trace("MessageMerger merging - currentMessage: {},
newMessage: {}, merged: {}",
                    currentMessage.asRecords(),
                    newMessage.asRecords(), result.asRecords());
        }

        return result;
    }
}

```

The above example merges delta messages for record topics with variable records and fields.

`MessageMerger` implementations must be thread safe and stateless. The same `MessageMerger` instance can be supplied to more than one different conflation policy if so required.

Default conflation policy

You can specify a default conflation policy that is used for any topics that do not have explicit policy mappings.

Use a default conflation policy only if you want to apply conflation to all topics when conflation is enabled for a client.

This can be specified using the `default-conflation-policy` property in the `conflation` section of `etc/Server.xml`. Alternatively it can be set programmatically at any time using the `setDefaultPolicy` method on `ConflationConfig`.

If no default policy is set, conflation will not occur for topics that have no explicit mappings even when conflation is enabled.

Mapping topics to policies

Having defined one or more conflation policies, you can map topics to the conflation policies that are to be used for them.

Either a full topic name or a topic selector pattern can be used when mapping to a conflation policy.

As the use of topic selectors makes it possible for more than one mapping to potentially apply to the same topic, the last mapping defined that matches a specific topic is the one that is used for conflating messages of that topic.

A default conflation policy can be specified which is selected to map to if no other mapping matches a topic.

Messages for topics that have no mappings (when there is no default policy) are not conflated, even if conflation is enabled for a client.

Conflation mappings can be defined using `topic-conflation` elements within the `conflation` section of the `etc/Server.xml` property file.

Conflation mappings can also be set programmatically using the `setTopicPolicy` method of `ConflationConfig`. Mappings can be set at any time during the running of a server. Mappings can also be removed at any time using `unsetTopicPolicy`.

Note: Removing a conflation policy at runtime will automatically remove any mappings to it.

Enabling conflation

Specify conflation for a queue-definition by setting the `conflates` property to `true`. This queue definition can then be used wherever required, for example by connectors that have conflation enabled for all clients.

Configuring authentication handlers

Authentication handlers and the order that the Diffusion server calls them in are configured in the `Server.xml` configuration file.

To configure authentication handlers for your server, edit the `Server.xml` configuration file to include the following elements:

```
<security>
  <authentication-handlers>
    <authentication-handler class="com.example.LocalLDAPHandler" />
    <system-authentication-handler/>
    <control-authentication-handler handler-name="RemoteHandler" />
  </authentication-handlers>
</security>
```

Ordering your configuration handlers

The order of handler elements within the `<authentication-handlers>` element defines the order in which the authentication handlers are called. In the preceding example, `localLDAPHandler` is called first. If `localLDAPHandler` returns an `ABSTAIN` result, the system authentication handler is called next. If the system authentication handler returns an `ABSTAIN` result, `RemoteHandler` is called next.

Order your authentication handlers from least to most restrictive and configure your handlers to abstain unless they are to explicitly allow or deny the authentication request.

For more information, see [Authentication](#) on page 140.

Configuring local authentication handlers

Configure local authentication handlers by using the `<authentication-handler/>` element. The value of the attribute `class` is the class name for the handler.

You can configure any number of distinct local authentication handlers in the `Server.xml` file.

Configuring the system authentication handler

You can configure Diffusion to use the system authentication handler by using the `<system-authentication-handler/>` element. The system authentication handler uses information in the system authentication store to make authentication decisions.

You can configure the system authentication handler to be called at most once. This restriction is not enforced by the XSD for the `Server.xml` file, but the Diffusion server does enforce this restriction on the configuration.

Configuring control authentication handlers

Configure control authentication handlers are configured by using the `<control-authentication-handler/>` element. The value of the attribute `handler-name` is the name by which the handler was registered by the control client. Control clients use the `AuthenticationControl` feature to register the handler and passing the binding name as a parameter.

If no control client has registered a control authentication handler with the name defined in the configuration file, the response for that handler is `ABSTAIN`.

Multiple control clients can register a control authentication handler with the same name. Registering a control authentication handler from multiple clients gives the following advantages:

- If one of the control clients becomes unavailable, another can handle the authentication request.
- Control clients can be changed or updated without affecting the authentication behavior.
- Authentication requests can be load balanced between the control clients.

You can configure any number of distinct control authentication handlers in the `Server.xml` file.

Note: To register a control authentication handler, an authenticating client must first connect to and authenticate with the server. We recommend that you configure a local authentication handler or the system authentication handler in the `Server.xml` file to authenticate the control client.

Related concepts

[User-written authentication handlers](#) on page 143

You can implement authentication handlers that authenticate clients that connect to the Diffusion server or perform an action that requires authentication.

[Local authentication handlers](#) on page 508

You can implement authentication handlers that authenticate client connections to the Diffusion server.

[Example: Register an authentication handler](#) on page 395

The following examples use the Diffusion Unified API to register a control authentication handler with the Diffusion server. The examples also include a simple or empty authentication handler.

[Authenticating clients](#) on page 395

A client can use the `AuthenticationControl` feature to authenticate other client sessions.

Related reference

[System authentication handler](#) on page 145

Diffusion provides an authentication handler that uses principal, credential, and roles information stored in the Diffusion server to make its authentication decision.

Authentication API

[Server.xml](#) on page 563

This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

Configuring performance

Use the `Server.xml` configuration file to configure behaviors and parameters that affect the performance of the Diffusion server.

For more information on the factors to consider when configuring the performance of your Diffusion server, see the [Tuning](#) on page 1052 section of this guide.

Server.xml

This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

server

All server properties

The following table lists the elements that an element of type `server` can contain:

Name	Type	Description	Min occurs	Max occurs
server-name	push:string	The server name is used to identify this server if running in a cluster. If not specified, the local hostname is used.	0	1
max-message-size	push:bytes	The maximum message size in bytes. This defines the maximum message size (including headers) that can be received.	1	1
default-load-message-capacity	push:bytes	The default capacity of a load message if not explicitly specified. If not supplied, a default of 4096 is used.	0	1
default-delta-message-capacity	push:bytes	The default capacity of a delta message if not explicitly specified. If not specified, a default of 1024 is used.	0	1
log-message-data	push:boolean	Indicates whether the data part of messages is logged as part of routine diagnostic message logging (FINE and FINEST levels). If this is false, credentials message headers are also hidden. If a value is not specified, a default of true is used.	0	1
message-length-size	push:int	DEPRECATED: since 5.2. This value is no longer used.	0	1
charset	push:string	The default character set to use for Diffusion message character conversions. See Java Encodings for	0	1

Name	Type	Description	Min occurs	Max occurs
		the full list. If a value is not specified, a default of "UTF-8" is used. DEPRECATED: Since 5.9 - in future releases only UTF-8 will be supported.		
multiplexers	multiplexers	DEPRECATED: These configuration options are retained for backwards compatibility. Using this element causes a warning to be output in the logs. In future, these configuration options will be removed. The multiplexers used by Diffusion can be configured using the multiplexer element.	0	1
multiplexer	multiplexer	Properties that define the multiplexer.	0	1
write-selectors	write-selectors	DEPRECATED: These configuration options are no-ops retained for backwards compatibility. All types of selector have now been unified. The selectors used by Diffusion can be configured using selector-thread-pool-definition.	0	1
security	security	Properties relating to security (optional).	0	1
conflation	conflation	Conflation policies and topic to policy mappings.	0	1
client-queues	client-queues	Definitions of client queues.	1	1
connection-timeouts	connection-timeouts	Timeout values relating to connections. If a value is not specified, defaults are used.	0	1
date-formats	date-formats	Date and time formats. If a value is not specified, default formats are used.	0	1
thread-pools	thread-pools	Definitions of thread pools	1	1
selector-thread-pools	selector-thread-pools	Definitions of thread pools	0	1
whois	whois	Definition of the Whois lookup service. If a value is not specified, no Whois service runs.	0	1
auto-deployment	auto-deployment	Automatic deployment properties (optional).	0	1
geo-ip	geo-ip	Properties relating to the Geo IP lookup facility. If a value is not specified, defaults are used.	0	1
usr-lib	usr-lib	User libraries (optional).	0	1
hooks	hooks	User hooks used in the server (optional)	0	1

Name	Type	Description	Min occurs	Max occurs
fanout	fanout	Properties relating to fan-out (optional). If not specified then the server will not be enabled as a fan-out secondary server.	0	1

multiplexers

DEPRECATED: These configuration options are retained for backwards compatibility. Using this element causes a warning to be output in the logs. In future, these configuration options will be removed. The multiplexers used by Diffusion can be configured using the multiplexer element.

The following table lists the elements that an element of type `multiplexers` can contain:

Name	Type	Description	Min occurs	Max occurs
client	push:string	Name of the client multiplexer definition. If this is not specified, the first multiplexer defined is used.	0	1
multiplexer-definition	multiplexer-definition	Multiplexer definition.	1	unbounded

multiplexer-definition

DEPRECATED: These configuration options are retained for backwards compatibility. Using this element causes a warning to be output in the logs. In future, these configuration options will be removed. The multiplexers used by Diffusion can be configured using the multiplexer element.

The following table lists the attributes that an element of type `multiplexer-definition` can have:

Name	Type	Description	Required
name	push:string	The multiplexer name.	true

The following table lists the elements that an element of type `multiplexer-definition` can contain:

Name	Type	Description	Min occurs	Max occurs
size	push:positiveInt	This is the number of multiplexer instances that start in readiness for clients to be assigned to. If there are going to be a large number of users, increase this number. If a value is not specified, a default value equal to the number of available processors is used.	0	1
thread-priority	push:positiveInt	DEPRECATED: Since 5.8. The multiplexer thread priority can no longer be set. This configuration option has no effect, and is retained only for backwards compatibility.	0	1

Name	Type	Description	Min occurs	Max occurs
load-balancer	push:string	DEPRECATED: This is the load balancer to use for for assigning connecting clients to multiplexer instances. There are currently two implemented load balancers, 'RoundRobin' and 'LeastClients'. If a value is not specified, a default of 'RoundRobin' is used. In future, only the 'RoundRobin' load balancing policy will be provided. If you have found the 'LeastClients' algorithm useful, contact Push Technology Support and reference case 11098.	0	1
latency-warning	push:millis	Multiplexers are critical to the operation of Diffusion. If there are too many clients assigned to too few multiplexer instances, there is a possibility of message latency. This is an optional flag which can be set to issue a warning if the multiplexer instance is taking too long in its operational cycle (see ServerNotificationListener in the publisher API). If this value is 0, this feature is not enabled. If this value is not supplied, a default of 0 is used.	0	1
max-event-queue-size	push:positiveInt	This specifies the maximum size of the multiplexer event queue. This is the queue on which events from publishers are queued for multiplexers and the default value is normally more than adequate. If this queue fills, it can cause the publisher threads to block until they can enqueue events and in this case it might be necessary to increase the value. Typically, leave this value at the default of 128k.	0	1

multiplexer

Multiplexer definition.

The following table lists the elements that an element of type `multiplexer` can contain:

Name	Type	Description	Min occurs	Max occurs
size	push:positiveInt	This is the number of multiplexer instances that start in readiness for clients to be assigned to. If there are going to be a large number of users, increase this number. If a value is not specified, a default value equal to the number of available processors is used.	0	1

Name	Type	Description	Min occurs	Max occurs
thread-priority	push:positiveInt	DEPRECATED: Since 5.8. The multiplexer thread priority can no longer be set. This configuration option has no effect, and is retained only for backwards compatibility.	0	1
latency-warning	push:millis	The multiplexer latency warning threshold. Multiplexers are critical to the operation of Diffusion. If there are too many clients assigned to too few multiplexer instances, there is a possibility of message latency. This is an optional setting which can be set to issue a warning if the multiplexer instance is taking too long to complete its operational cycle. If this value is not supplied, a default of 1000 (1 second) is used. Warnings are logged to the server log and reported to the publisher MultiplexerLatencyListener event API.	0	1
max-event-queue-size	push:positiveInt	This specifies the maximum size of the multiplexer event queue. This is the queue on which events from publishers are queued for multiplexers and the default value is normally more than adequate. If this queue fills, it can cause the publisher threads to block until they can enqueue events and in this case it might be necessary to increase the value. Typically, leave this value at the default of 128k.	0	1

write-selectors

DEPRECATED: These configuration options are no-ops retained for backwards compatibility. All types of selector have now been unified. The selectors used by Diffusion can be configured using selector-thread-pool-definition.

The following table lists the elements that an element of type `write-selectors` can contain:

Name	Type	Description	Min occurs	Max occurs
thread-priority	push:positiveInt	DEPRECATED: This configuration setting is a no-op.	0	1
size	push:positiveNonZeroInt	DEPRECATED: This configuration setting is a no-op.	0	1
timeout	push:millis	DEPRECATED: This configuration setting is a no-op.	0	1
load-balancer	push:string	DEPRECATED: This configuration setting is a no-op.	0	1

Name	Type	Description	Min occurs	Max occurs
queue-size	push:positiveNon	DEPRECATED: This configuration setting is a no-op.	0	1

hooks

User hooks used in the server.

The following table lists the elements that an element of type `hooks` can contain:

Name	Type	Description	Min occurs	Max occurs
startup-hook	push:string	This is the class name of a class that implements the interface <code>com.pushtechology.diffusion.api.publisher.ServerStartupHook</code> . If specified, the hook is instantiated and the <code>serverStarting</code> method called when the server is starting, before the loading of publishers.	0	1
shutdown-hook	push:string	This is the class name of a class that implements the interface <code>com.pushtechology.diffusion.api.publisher.ServerShutdownHook</code> . If specified, the hook is instantiated and the <code>serverStopping</code> method called when the server is stopping.	0	1

security

Server security properties.

The following table lists the elements that an element of type `security` can contain:

Name	Type	Description	Min occurs	Max occurs
authorisation-handler-class	push:string	This is the full name of a class, on the classpath, that implements the <code>AuthorisationHandler</code> interface in the Java publisher API. If specified, the handler is instantiated when the server starts and is called to authorize client connections, subscriptions, and fetch requests.	0	1
authentication-handlers	authentication-handlers		0	1

authentication-handlers

Authentication handlers, in order of decreasing precedence. The authentication handlers are called to authenticate new connections and changes to the principal associated with a session. Authentication handlers are configured in precedence order. Authentication succeeds if a handler returns "allow" and all higher precedence handlers (earlier in the order) return "abstain". Authentication fails if a handler

returns "deny" and all higher precedence handlers return "abstain". If all authentication handlers return "abstain", the request is denied. After the outcome is known, the server might choose not to call the remaining handlers.

The following table lists the elements that an element of type `authentication-handlers` can contain:

Name	Type	Description	Min occurs	Max occurs
authentication-handler	server-authentication-handler	An authentication handler hosted by the server.	0	unbounded
control-authentication-handler	control-authentication-handler	An authentication handler registered by a client.	0	unbounded
system-authentication-handler	system-authentication-handler	An authentication handler that uses the configured system authentication store to validate principals and to define an action for anonymous logins. The XSD does not prevent you from configuring the system authentication multiple times. However, the Diffusion server restricts this and will not start if you define the system authentication handler more than once.	0	unbounded

server-authentication-handler

An authentication handler hosted by the server. The handler is instantiated when the server starts.

The following table lists the attributes that an element of type `server-authentication-handler` can have:

Name	Type	Description	Required
class	push:string	The class attribute specifies the fully qualified name of a handler implementation class that implements the <code>com.pushtechology.diffusion.client.security.authentication.AuthenticationHand</code> interface. The class must be available on the classpath.	true

system-authentication-handler

The system authentication handler uses the configured system authentication store to validate principals and to define an action for anonymous logins. If the system handler is specified then it will check if a principal is specified in the store and if so will validate its credentials against the store. The store may also specify additional assigned roles to be granted to a principal. If a principal is not specified in the store then the handler will abstain. The store may indicate whether to allow, deny or abstain for anonymous logins.

control-authentication-handler

Client sessions register control authentication handlers using an identifying name. A `<control-authentication-handler>` must be configured with a matching `handler-name`. Configure at most one `<control-authentication-handler>` for a `handler-name`.

The following table lists the attributes that an element of type `control-authentication-handler` can have:

Name	Type	Description	Required
handler-name	push:string	The handler name attribute must match the identifying name used by the client session to register a control authentication handler.	true

conflation

Conflation policies and topic to policy mappings.

The following table lists the elements that an element of type `conflation` can contain:

Name	Type	Description	Min occurs	Max occurs
default-conflation-policy	push:string	The default conflation policy. This specifies a conflation policy that is used for any topics that do not have explicit conflation policy mappings defined. If this is not specified, conflation does not occur for topics that do not have a policy mapping defined. If this value is specified, it must be the name of a defined policy.	0	1
conflation-policy	conflation-policy	Conflation policy.	0	unbounded
topic-conflation	topic-conflation	A mapping between a topic (or topic selector pattern) and a policy.	0	unbounded

conflation-policy

Conflation policy.

The following table lists the attributes that an element of type `conflation-policy` can have:

Name	Type	Description	Required
name	push:string	The conflation policy name.	true

The following table lists the elements that an element of type `conflation-policy` can contain:

Name	Type	Description	Min occurs	Max occurs
mode	push:string	The conflation mode. This can have the value 'replace' or 'append'. If a value is not specified, a default of 'replace' is assumed. The value 'replace' means that when a matching message is found, the message is replaced by the new (or merged) message in its current queue position. The value 'append' means that when a matching message is found, the message is removed from its current queue position and the new (or merged)	0	1

Name	Type	Description	Min occurs	Max occurs
		message is appended to the end of the queue. This option preserves message ordering but there is the danger that messages are constantly sent to the end of the queue.		
matcher	push:string	The full class name of a message matcher of type <code>com.pushtechology.diffusion.api.conflation.MessageMatcher</code> . If a class is not supplied, a default matcher that matches by topic name is used.	0	1
merger	push:string	The full class name of a message merger of type <code>com.pushtechology.diffusion.api.conflation.MessageMerger</code> . If a class is not supplied, no merging with the new message occurs. The latest message either replaces the existing one or the existing one is removed and the new one appended to the end of the queue depending upon the mode. The default merging behavior is suitable for single value topics, but not for any topics that have a complex structure, such as record topics. If the messages being conflated are delta messages, information can be lost when intermediate deltas are discarded in favor of the latest delta message.	0	1

topic-conflation

A mapping between a topic (or topic selector pattern) and conflation policy.

The following table lists the elements that an element of type `topic-conflation` can contain:

Name	Type	Description	Min occurs	Max occurs
topic	push:string	The name of a topic or a topic selector pattern that indicates the topic or topics that the specified conflation policy is applied to.	1	1
policy	push:string	The name of a configured conflation policy that is applied to the specified topic or topics.	1	1

client-queues

Client queue definitions.

The following table lists the elements that an element of type `client-queues` can contain:

Name	Type	Description	Min occurs	Max occurs
default-queue-definition	push:string	The name of the queue definition to use by default. Connectors that do not explicitly specify a queue definition use the one specified here.	1	1
queue-definition	queue-definition	Queue definition.	1	unbounded

queue-definition

This defines the properties of a client queue.

The following table lists the attributes that an element of type `queue-definition` can have:

Name	Type	Description	Required
name	push:string	The queue definition name.	true

The following table lists the elements that an element of type `queue-definition` can contain:

Name	Type	Description	Min occurs	Max occurs
max-depth	push:positiveInt	The maximum depth of the queue. If the number of messages queued for a client exceeds this number, the server disconnects the client.	1	1
conflates	push:boolean	Specifies whether conflation is applied to all clients using this queue definition. If this value is not specified, conflation is not applied by default.	0	1
upper-threshold	push:percent	This specifies a percentage of the maximum queue size and if this value is reached then any listeners (see <code>ClientListener</code> in the publisher API) are notified. Notification occurs only once and does not occur again until the queue has returned to the lower threshold. If this value is not specified, no upper limit notification occurs.	0	1
lower-threshold	push:percent	This specifies a percentage of the maximum queue size and indicates the level at which listeners (see <code>ClientListener</code> in the publisher API) are notified after an upper limit notification has occurred and the queue size has dropped back to the specified lower limit. If this value is not specified, no lower limit notification occurs.	0	1
auto-fragment	push:boolean	DEPRECATED : since 5.1.6. Topic message fragmentation has been removed. This setting has no effect.	0	1

connection-timeouts

Connection-related timeouts.

The following table lists the elements that an element of type `connection-timeouts` can contain:

Name	Type	Description	Min occurs	Max occurs
write-timeout	<code>push:millis</code>	The write timeout in milliseconds for blocking write operations. Most write operations are non-blocking and are not affected by this timeout. Blocking writes include connection responses to new clients, and HTTP responses to web server requests. If this value is not specified, a default of 2 seconds is used. If this exceeds one hour (3600000ms) a warning will be logged and the time-out will be set to one hour.	0	1
connection-timeout	<code>push:millis</code>	The time in milliseconds allowed for a connection to complete its handshake processing, including the time taken to call any configured authentication handlers and look up location details. If this value is not specified, a default of 2 seconds is used. If this exceeds one hour (3600000ms) a warning will be logged and the time-out will be set to one hour.	0	1

date-formats

Date and time formats.

The following table lists the elements that an element of type `date-formats` can contain:

Name	Type	Description	Min occurs	Max occurs
date	<code>push:string</code>	The format used when displaying dates. Specify the format according to the Java SimpleDateFormat specification. If a format is not specified, a default of "yyyy-MM-dd" is used.	0	1
time	<code>push:string</code>	The format used when displaying times. Specify the format according to the Java SimpleDateFormat specification. If a format is not specified, a default of "HH:mm:ss" is used.	0	1
date-time	<code>push:string</code>	The format used when displaying date and time. Specify the format according to the Java SimpleDateFormat specification. If a format is not specified, a default of "yyyy-MM-dd HH:mm:ss" is used.	0	1

Name	Type	Description	Min occurs	Max occurs
timestamp	push:string	The format used when displaying a timestamp - for example, in a log - to millisecond precision. Specify the format according to the Java SimpleDateFormat specification. If a format is not specified, a default of "yyyy-MM-dd HH:mm:ss.SSS" is used.	0	1

thread-pools

Thread pools.

The following table lists the elements that an element of type `thread-pools` can contain:

Name	Type	Description	Min occurs	Max occurs
inbound	push:string	Name of the inbound thread pool definition.	1	1
outbound	push:string	DEPRECATED : This property is no longer used.	0	1
background-thread-size	push:int	Number of threads to use for the background thread pool. If a value is not specified, a default of 10 is used.	0	1
thread-pool-definition	thread-pool-definition	Thread pool definition.	1	unbounded

thread-pool-definition

Thread pool definition.

The following table lists the attributes that an element of type `thread-pool-definition` can have:

Name	Type	Description	Required
name	push:string	Name of the thread pool definition.	true

The following table lists the elements that an element of type `thread-pool-definition` can contain:

Name	Type	Description	Min occurs	Max occurs
core-size	push:positiveInt	The core number of threads to have running in the thread pool. Whenever a thread is required a new thread is created until this number is reached even if there are idle threads already in the pool.	1	1
max-size	push:positiveInt	The maximum number of threads that can be created in the thread pool before tasks are queued. Such threads are	0	1

Name	Type	Description	Min occurs	Max occurs
		released immediately after execution. If not set, the value defaults to the core-size.		
queue-size	push:positiveInt	The thread pool queue size. When the max-size is reached, tasks are queued. If the value is 0, the queue is unbounded. If the value is not 0, it must be at least 10.	1	1
keep-alive	push:millis	The time to keep inactive threads alive for. This does not apply to core threads. If this value is not specified, a default of 0 is used.	0	1
priority	push:positiveInt	DEPRECATED: Since 5.8. The thread priority can no longer be set. This configuration option has no effect, and is retained only for backwards compatibility.	0	1
thread-pool-listener	thread-pool-listener	DEPRECATED: Since 5.8 configuration of thread pool listeners is not supported	0	1
rejection-handler-class	push:string	The name of a class implementing the ThreadPoolRejectionHandler interface which is called if a task cannot be executed by the Thread Pool. If this value is not specified, a default rejection policy is used so that rejected tasks are executed in the calling thread. The default rejection policy is implemented by the class com.pushtechnology.diffusion.api.threads.ThreadService.CallerRunsRejectionP A thread is rejected if all the threads are in use and the queue is full.	0	1

thread-pool-listener

Thread pool listener details.

The following table lists the elements that an element of type `thread-pool-listener` can contain:

Name	Type	Description	Min occurs	Max occurs
queue-notification-handler-class	push:string	The name of a class implementing the ThreadPoolNotificationHandler interface which is instantiated to handle notifications on the thread pool.	1	1
queue-upper-threshold	push:percent	The size of the thread pool queue at which the notification handler is called on the queueUpperThresholdReached	1	1

Name	Type	Description	Min occurs	Max occurs
		method. The method is called once only until the queue size drops below the specified lower threshold.		
queue-lower-threshold	push:percent	The size of the thread pool queue at which the notification handler is called on the queueLowerThresholdReached method if the upper threshold has previously been breached.	1	1

selector-thread-pools

Thread pools.

The following table lists the elements that an element of type `selector-thread-pools` can contain:

Name	Type	Description	Min occurs	Max occurs
default	push:string	Name of the default selector thread pool definition.	0	1
selector-thread-pool-definition	selector-thread-pool-definition	Selector thread pool definition.	1	unbounded

selector-thread-pool-definition

Selector thread pool definition.

The following table lists the attributes that an element of type `selector-thread-pool-definition` can have:

Name	Type	Description	Required
name	push:string	Name of the selector thread pool definition.	true

The following table lists the elements that an element of type `selector-thread-pool-definition` can contain:

Name	Type	Description	Min occurs	Max occurs
size	push:positiveInt	The number of selector threads to have running in the thread pool. The number of selector threads created is the maximum of the value defined here and the number of acceptors defined in the Connectors.xml file. This number is fixed and does not change at runtime.	0	1

whois

Whois service details.

The following table lists the elements that an element of type `whois` can contain:

Name	Type	Description	Min occurs	Max occurs
provider	push:string	Name of the Whois provider class that must be on the classpath and must implement the API class WhoisProvider. If a provider is not specified, WhoisDefaultProvider is used.	0	1
threads	push:int	The number of background threads that process Whois resolver requests. If a value is not specified, a default of 2 is used. If the value is set to 0, the service is not started.	0	1
host	push:string	The hostname of a Whois provider that adheres to the RFC3912 Whois protocol. If a hostname is not specified, a default of "whois.ripe.net" is used.	0	1
port	push:port	The port number that the Whois provider listens on. If a value is not specified, the normal value of 43 is used.	0	1
whois-cache	whois-cache	Details of the Whois service cache that is used to cache Whois lookup results. If a value is not specified, the default values are used.	0	1

whois-cache

Details of the Whois service cache that is used to cache Whois lookup results.

The following table lists the elements that an element of type `whois-cache` can contain:

Name	Type	Description	Min occurs	Max occurs
maximum	push:int	The maximum size of the Whois cache. When the cache size exceeds this number it is tidied. A value of 0 means the cache grows indefinitely unless entries are removed because they have exceeded their retention time. If a value is not specified, a default of 1000 is used.	0	1
retention	push:millis	The time for which Whois cache entries are retained before being deleted. A value of 0 means entries are retained indefinitely or until the cache reaches its maximum size. If a value is not specified, a default of 0 is used.	0	1
tidy-interval	push:millis	The interval at which the Whois cache tidier task checks if any cache entries have passed their retention time or if the cache has exceeded its maximum size.	0	1

Name	Type	Description	Min occurs	Max occurs
		This is ignored if both maximum and retention are 0. If a value is not specified, a default of 1 minute is used.		

auto-deployment

Auto deployment details.

The following table lists the elements that an element of type `auto-deployment` can contain:

Name	Type	Description	Min occurs	Max occurs
directory	<code>push:string</code>	The name of the automatic deployment directory.	1	1
scan-frequency	<code>push:millis</code>	The frequency at which the deployment directory is scanned for new deployments. If a value is not specified, a default of 5 seconds is used.	0	1

geo-ip

GeoIP details.

The following table lists the attributes that an element of type `geo-ip` can have:

Name	Type	Description	Required
enabled	<code>push:boolean</code>	Set to true to enable GeoIP lookup. This needs to be set to true if you are going to use connection or subscription validation policies. If a value is not specified, a default of true is used.	false

The following table lists the elements that an element of type `geo-ip` can contain:

Name	Type	Description	Min occurs	Max occurs
file-name	<code>push:string</code>	The name of the Maxmind GeoCityIP city file. If a value is not specified, a default of <code>"../data/GeoLiteCity.dat"</code> is used.	0	1

usr-lib

A list of user libraries from which user code is loaded.

The following table lists the elements that an element of type `usr-lib` can contain:

Name	Type	Description	Min occurs	Max occurs
directory	<code>push:string</code>	Directory to load classes from. When the server starts, this folder is traversed, including subdirectories and all jars or zip files added to the class loader.	1	unbounded

fanout

Specifies fan-out connections to establish with primary servers. Typically there is a single connection but it is possible to replicate topics from more than one primary server as long as they do not overlap. All such connections are automatically established when the secondary server starts and will recover as configured.

The following table lists the elements that an element of type `fanout` can contain:

Name	Type	Description	Min occurs	Max occurs
connection	<code>fanout-connection</code>	A fan out connection.	0	unbounded

fanout-connection

Represents a fan-out connection from a secondary server to a primary server.

The following table lists the elements that an element of type `fanout-connection` can contain:

Name	Type	Description	Min occurs	Max occurs
url	<code>push:string</code>	The connection URL which specifies the primary server to connect to.	1	1
principal	<code>push:string</code>	The principal used to connect to the primary server. If not specified, an anonymous connection is assumed.	0	1
password	<code>push:string</code>	The password to use for the connection. If not specified, no credentials are assumed.	0	1
retry-delay	<code>push:millis</code>	This is the time to wait after failing to connect or losing a connection before trying to connect again. The value is specified in milliseconds. If this value is not specified, a default of 1s is used.	0	1
retry-interval	<code>push:int</code>	DEPRECATED : This value is no longer used. Use <code>retry-delay</code> instead.	0	1
reconnect-timeout	<code>push:millis</code>	This is the total time in milliseconds that will be allowed to reconnect a failed connection to the primary server. For reconnection to work the primary server connector must have been configured to support reconnection. If this is not specified, a value of 0 is assumed which means that reconnection will not be attempted (this does not affect retry). If reconnection is configured and a load balancer is in use then it must be configured for sticky routing.	0	1
recovery-buffer-size	<code>push:int</code>	If the primary server is configured to support reconnection, a session established with a non-zero <code>reconnect-timeout</code> retains a buffer of sent	0	1

Name	Type	Description	Min occurs	Max occurs
		messages. If the session disconnects and reconnects, this buffer is used to re-send messages that the server has not received. The default value is 10,000 messages. If reconnect-timeout is 0 then this value is ignored.		
input-buffer-size	push:bytes	Specifies the size of the input buffer to use for the connection with the primary server. This is used to receive messages from the primary server. Set this to the same size as the output buffer used at the primary server. If not specified, maximum message size is assumed.	0	1
output-buffer-size	push:bytes	The size of the output buffer to use for the connection with the primary server. This is used to send messages to the primary server. Set this to the same size as the input buffer used by the primary server. If not specified, maximum message size is assumed.	0	1
maximum-queue-size	push:int	The maximum number of messages that can be queued to send to the primary server. If this number is exceeded, the connection will be closed. This must be sufficient to cater for messages that may be queued whilst disconnected (awaiting reconnect). The default value is 10,000 messages.	0	1
connection-timeout	push:millis	This specifies the connection timeout value (in milliseconds). If a value is not specified, a default of 2s is used.	0	1
write-timeout	push:millis	This specifies the write timeout value (in milliseconds). If a value is not specified, a default of 2s is used.	0	1
link	fanout-link	Specifies a link to a selection of topics at the primary server that are to be replicated at the secondary server.	1	unbounded

fanout-link

Represents a selection of topics from the primary topic tree to be replicated to the secondary server.

The following table lists the elements that an element of type `fanout-link` can contain:

Name	Type	Description	Min occurs	Max occurs
selector	push:string	A topic selector specifying the topics to be replicated. This must not overlap (select the same topics as) any other link within this or any other connection configured for the secondary server.	1	1

Related concepts

[User-written authentication handlers](#) on page 143

You can implement authentication handlers that authenticate clients that connect to the Diffusion server or perform an action that requires authentication.

Configuring connectors

A connector provides a connection point for external applications to connect to the Diffusion server over a TCP connection. Use the `connectors.xml` configuration file to configure your connectors.

Each connector has a socket server thread which reacts to an incoming connection. The socket information is defined by the connector. Suitable connectors must be defined for inbound connections expected by the Diffusion server.

The following properties are common to all connectors:

Table 61: Connectors properties

Name	A name by which the connector can be identified.
Port	A port number on which to accept requests (or policy requests).
Host	The host to accept requests (only relevant on a multi-homed machine).
DEPRECATED: Number of acceptor threads	The number of acceptor threads. This can be tuned for performance reasons.
Input buffer size	The size of the socket input buffer to use for each connection.
Output buffer size	The size of the socket output buffer to use for each connection.

Socket buffer sizes are very important in achieving the best performance. For more information, see [Tuning](#) on page 1052.

Restricting connection types

By default a connector can accept any type of connection handled by Diffusion, but you can configure a connector so that it accepts only one type of connection as follows:

Table 62: Connection restrictions

client	Client connections.
policy	Policy file requests.
all	Any type of connection.

Client connections

Connectors can accept connections from any type of client. Any number of connectors can be defined to provide different connection points with different properties.

Each client connection has an input buffer to receive messages from the client. The configured input buffer size must be large enough to accommodate the largest message expected from the client. If the maximum message size and the input buffer size are configured as different values, the larger of the two is used as the input buffer size.

The output buffer size is used to assign an output buffer per client multiplexer into which messages are dequeued prior to transmission. This can have an important effect on performance. For more information, see [Tuning](#) on page 1052.

Enabling client reconnection

Specify a reconnection timeout, maximum queue depth, and recovery buffer size by using the `<reconnect>` element in the `etc/Connectors.xml` configuration file.

Reconnection timeout (`keep-alive`)

How long a disconnected client's session remains available on the server before being closed. By default, this is 60 seconds.

Maximum queue depth (`max-depth`)

Optional maximum limit on the number of messages to queue for a disconnected client session. By default, this is the same as the queue depth for a connected client session, which is defined by the queue definitions in `Connectors.xml` and `Server.xml`.

Recovery buffer size (`recovery-buffer-size`)

The maximum number of sent messages to keep in a buffer. These messages can then be recovered on reconnection.

```
<connector>
  ...
  <reconnect>
    <keep-alive>60s</keep-alive>
    <max-depth>1000</max-depth>
    <recovery-buffer-size>64</recovery-buffer-size>
  </reconnect>
  ...
</connector>
```

A client can reconnect to the server through this connector within 60 seconds of becoming disconnected. While the client is disconnected, up to 1000 messages are queued for it. These messages are delivered to the client when it reconnects. A buffer of up to 64 sent messages are retained in the recovery buffer. When a client reconnects, the Diffusion server use this buffer to re-send any messages that the client has not received.

Policy connections

Connectors are used to serve policy file requests to plugin clients. Flash and Silverlight require policy files to be served from different ports (Flash on 843 and Silverlight on 943) so if plugin clients are in use it will be necessary to define a separate connector for each type of plugin client.

The connector configuration specifies the path of an XML policy file to be sent to the client.

Related reference

[Connectors.xml](#) on page 583

This file specifies the schema for the connectors properties.

Connectors.xml

This file specifies the schema for the connectors properties.

connectors

Connectors

The following table lists the elements that an element of type `connectors` can contain:

Name	Type	Description	Min occurs	Max occurs
connector	connector	Connector definition	0	unbounded

connector

Connector definition

The following table lists the attributes that an element of type `connector` can have:

Name	Type	Description	Required
name	<code>push:string</code>	The connector name	true

The following table lists the elements that an element of type `connector` can contain:

Name	Type	Description	Min occurs	Max occurs
required	<code>push:boolean</code>	This setting specifies if the connector is required on server start-up. If set to true and the connector does not start, the server is shut down.	0	1
type	connectorType	The type of connection supported. By default 'all' types are supported but the connector can be restricted to one of the following specific types - 'client' (Clients only), or 'policy' (Policy File Requests only).	0	1
api-type	connectorApiType	This setting constrains the API that can be used with this connector. The allowed values are 'all', 'classic', and 'unified'. The value 'unified' indicates that clients must use the Unified API. The value 'classic' indicates that clients must use the Classic API. The value 'all' indicates that clients can use either API. The default value is 'all'. DEPRECATED: since 5.9 as classic client APIs are also deprecated.	0	1
host	<code>push:string</code>	The name or the IP address that the connector binds to. This is optional.	0	1

Name	Type	Description	Min occurs	Max occurs
port	push:port	The port on which the connector accepts connections.	1	1
acceptors	push:positiveNon	DEPRECATED: This value is no longer used.	0	1
backlog	push:positiveNon	The maximum queue length for incoming clients. If a connection indication arrives when the queue is full, the connection is refused. If a value is not specified, a default of 1000 is used.	0	1
socket-conditioning	socket-conditioning	Describes the properties associated with TCP socket connections.	1	1
web-server	push:string	If this connector is required to serve HTTP requests, this element specifies a web-server entry in WebServer.xml. If a value is not specified, the connector cannot serve HTTP requests.	0	1
policy-file	push:string	The location/name of the policy file if this connector is required to act as a policy file server (type='all' or 'policy').	0	1
validation-policy-file	push:string	The location/name of a connection validation policy file to use for this connector. Applies only to type 'all' or 'client'.	0	1
key-store	key-store-definition	Keystore details for any connector that is to support secure (SSL) connections. If this is not specified, SSL connections are not supported.	0	1
queue-definition	push:string	An optional queue definition to use for this connector. This applies only to connectors of type 'all' or 'client'. The definition must exist in Server.xml. If this is not specified, the default queue definition specified in Server.xml is used.	0	1
reconnect	reconnect	Optional reconnection properties which apply only to connectors that accept 'client' connections. If this is not specified, reconnection of client connections is not supported.	0	1
ignore-errors-from	ignore-errors-from	Specifies addresses from which connection errors can be ignored. This is useful for masking errors that might be reported due to the connector port being pinged by some known external entity.	0	1

Name	Type	Description	Min occurs	Max occurs
thread-pool-definition	push:string	Optionally, this can be used to specify a thread pool definition to be used for this connector to create its own inbound thread pool. If specified, the thread pool definition must exist in Server.xml. If a value is not specified, the default inbound thread pool is used.	0	1
selector-thread-pool-definition	push:string	Optionally, this can be used to specify a selector thread pool definition to be used for this connector to deal with NIO operations. If specified, the selector thread pool definition must exist in Server.xml. If a value is not specified, the default selector thread pool is used.	0	1
system-ping-frequency	push:millis	This indicates the interval at which clients are pinged by the server to ensure that they are still connected. If a response is not received from the client before the expiry of another interval period, the client is assumed to be disconnected. If this is not specified or a value of 0 is supplied, clients are not automatically pinged.	0	1
fetch-policy	fetch-policy	Specifies a policy for batching fetch requests. If a value is not specified, no policy is applied and fetches are not batched.	0	1
proxy-protocol	proxyProtocol	Indicates the proxy protocol required for connection. Can have the values 'NONE' or 'HA_PROXY'. The default value is 'NONE'. Only connections with the protocol specified are allowed. On publicly accessible connectors, ensure that this value is set to NONE. 'HA_PROXY' refers to the proxy protocol that was first implemented by HAProxy but it is also supported by others including Amazon's Elastic Load Balancer.	0	1
connection-timeout	push:millis	This is the time in milliseconds allowed for a connection to take place and complete its handshake processing. If this value is not specified for a connector, the value set in Server.xml is used.	0	1

socket-conditioning

Describes properties associated with TCP socket connections.

The following table lists the elements that an element of type `socket-conditioning` can contain:

Name	Type	Description	Min occurs	Max occurs
input-buffer-size	<code>push:bytes</code>	Specifies the size of the socket input buffer to use for each connection. If a value is not specified, a default of 128k is used. The greater of this value and the max-message-size set in <code>Server.xml</code> is used when setting the socket input buffer size.	0	1
output-buffer-size	<code>push:bytes</code>	This value specifies the size of the output buffer to use for each connection. This must be large enough to accommodate the largest message to be sent. Messages are 'batched' into this buffer and so the larger the buffer, the more messages can be sent in a single write. If a value is not specified, a default of 64k is used.	0	1
keep-alive	<code>push:boolean</code>	This enables or disables TCP keep-alive. If a value is not specified, a default of true is used.	0	1
no-delay	<code>push:boolean</code>	This enables or disables TCP_NODELAY (disable/enable Nagle's algorithm). If a value is not specified, a default of true is used.	0	1
reuse-address	<code>push:boolean</code>	When a TCP connection is closed the connection can remain in a timeout state for a period of time after the connection is closed (typically known as the TIME_WAIT state or 2MSL wait state). For applications using a well-known socket address or port, it might not be possible to bind a socket to the required SocketAddress if there is a connection in the timeout state involving the socket address or port. Enabling this feature allows the socket to be bound even though a previous connection is in a timeout state. If this value is not specified, the feature is enabled.	0	1

reconnect

Reconnect properties.

The following table lists the elements that an element of type `reconnect` can contain:

Name	Type	Description	Min occurs	Max occurs
keep-alive	push:millis	This specifies the reconnection timeout. During this period a disconnected client can reconnect to the same client session. Messages for the client continue to be queued during this period.	1	1
max-depth	push:positiveInt	As messages continue to be queued for a client whilst it is disconnected, this enables you to specify a larger maximum queue size that is used during the period that the client is disconnected. When the client reconnects, the maximum reverts back to its previous size (once any backlog had been cleared). If the specified size is not greater than the current maximum size, this has no effect. If this value is not specified, a default of 0 is used which means that no attempt is made to extend the queue size when a client is disconnected.	0	1
recovery-buffer-size	push:positiveInt	If the keep-alive time is not zero, this connector supports reconnection. For each client connected via this connector, the server will retain a buffer of up to recovery-buffer-size sent messages. If a client disconnects and reconnects, the server uses the buffer to re-send messages that the client has not received. The default value is 128 messages. Higher values increase the chance of successful reconnection, but increase the per-client memory footprint.	0	1

key-store-definition

The keystore definition that allows SSL connection to a connector.

The following table lists the attributes that an element of type `key-store-definition` can have:

Name	Type	Description	Required
mandatory	push:boolean	If this is set to true, all connections must use this keystore and SSL connection is mandatory. If a value is not specified, a default of false is used, meaning that the connector accepts either SSL or non-SSL connections.	false

The following table lists the elements that an element of type `key-store-definition` can contain:

Name	Type	Description	Min occurs	Max occurs
file	push:string	The keystore file path.	1	1
password	push:string	The password for the keystore.	1	1

ignore-errors-from

Some external monitors cause the Diffusion server to log errors, as it is not a valid Diffusion connection. Adding the remote IP address to this list ensure that the errors are not logged.

The following table lists the elements that an element of type `ignore-errors-from` can contain:

Name	Type	Description	Min occurs	Max occurs
ip-address	push:string	An IP address or unknown if the remote IP address is being masked.	1	unbounded

fetch-policy

This is the policy for batching fetch requests. This can be used when fetches on topic sets might be large and lead to an excessive number of fetch reply messages being queued for a client at one time. The policy can define that the replies are sent in periodic batches to allow the client time to process them and prevent client queues filling.

The following table lists the elements that an element of type `fetch-policy` can contain:

Name	Type	Description	Min occurs	Max occurs
batch-size	push:positiveInt	Specifies the maximum number of fetch reply messages to send per batch. If this is set to 0, no batching occurs.	1	1
delay	push:millis	Specifies the time period between submissions of batches. If a batch size is specified, this must be a positive value.	1	1

Related reference

[Configuring connectors](#) on page 581

A connector provides a connection point for external applications to connect to the Diffusion server over a TCP connection. Use the `Connectors.xml` configuration file to configure your connectors.

Configuring user security

You can use the `Security.store` and `SystemAuthentication.store` files in the `etc` directory of your Diffusion server to configure the security roles and how they are assigned.

These files can also be updated on the running Diffusion server by clients that provide the capability to update the security settings. For more information, see [#unique_78/unique_78_Connect_42_managing_security](#) on page .

Related concepts

[Updating the security store](#) on page 420

A client can use the SecurityControl feature to update the security store. The information in the security store is used by the Diffusion server to define the permissions assigned to roles and the roles assigned to anonymous sessions and named sessions.

[Role-based authorization](#) on page 130

Diffusion restricts the ability to perform actions to authorized principals. Roles are used to map permissions to principals.

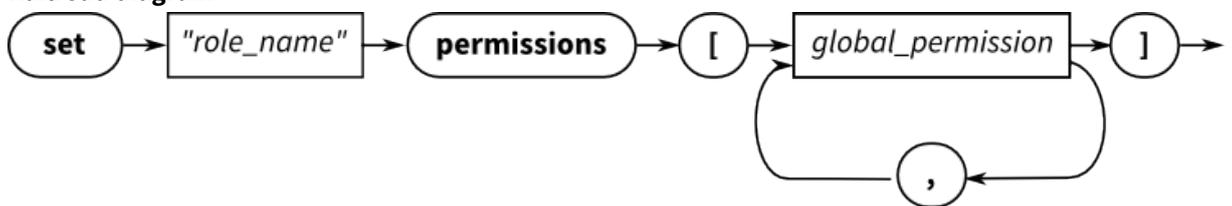
Security.store

The `Security.store` file defines the security roles and the permissions associated with them. It also defines the default set of roles that are assigned to named or anonymous client sessions.

The following sections each describe the syntax for a single line of the script file.

Assigning global permissions to a role

Railroad diagram



Backus-Naur form

```
set "role_name" permissions [ '[' global_permission [ , global_permission ] ' ] ]
```

Example

```
set "ADMINISTRATOR" permissions [CONTROL_SERVER, VIEW_SERVER,
VIEW_SECURITY, MODIFY_SECURITY]
set "CLIENT_CONTROL" permissions [VIEW_SESSION, MODIFY_SESSION,
REGISTER_HANDLER]
```

Assigning default topic permissions to a role

Railroad diagram



Backus-Naur form

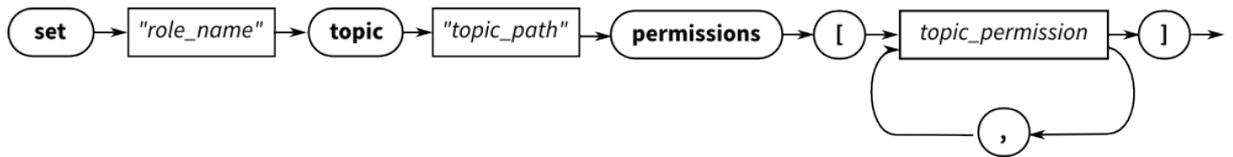
```
set "role_name" default topic permissions [ '[' topic_permission [ , topic_permission ] ' ] ]
```

Example

```
set "CLIENT" default topic permissions [READ_TOPIC ,
SEND_TO_MESSAGE_HANDLER]
```

Assigning topic permissions associated with a specific topic path to a role

Railroad diagram



Backus-Naur form

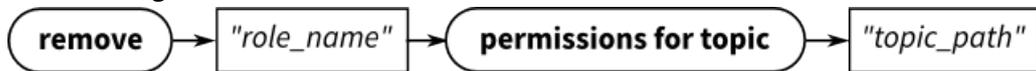
set "role_name" topic "topic_path" permissions ['[' topic_permission [, topic_permission]]]

Example

```
set "CLIENT" topic "foo/bar" permissions [READ_TOPIC,
SEND_TO_MESSAGE_HANDLER]
set "ADMINISTRATOR" topic "foo" permissions [ MODIFY_TOPIC ]
set "CLIENT_CONTROL" topic "foo" permissions [ ]
```

Removing all topic permissions associated with a specific topic path to a role

Railroad diagram



Backus-Naur form

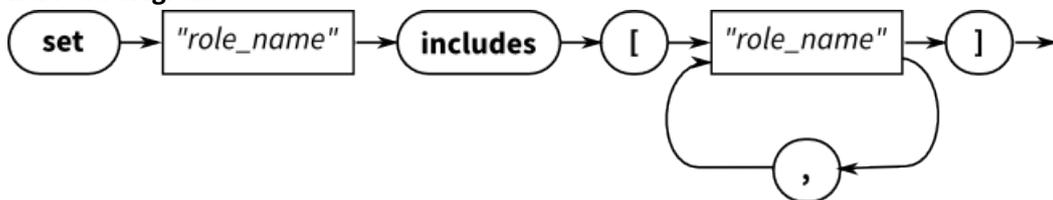
remove "role_name" permissions for topic "topic_path"

Example

```
remove "CLIENT" permissions for topic "foo/bar"
```

Including roles within another role

Railroad diagram



Backus-Naur form

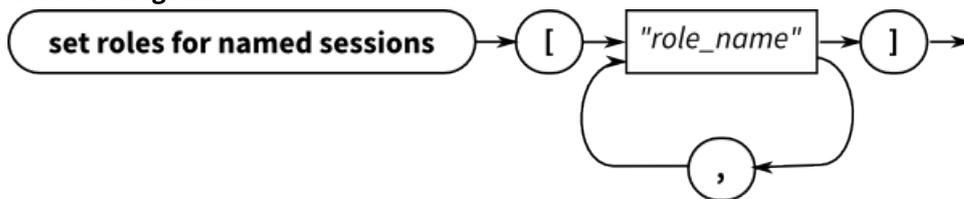
set "role_name" includes ['[' "role_name" [, "role_name"]]]

Example

```
set "ADMINISTRATOR" includes ["CLIENT_CONTROL" , "TOPIC_CONTROL"]
set "CLIENT_CONTROL" includes ["CLIENT"]
```

Assigning roles to a named session

Railroad diagram



Backus-Naur form

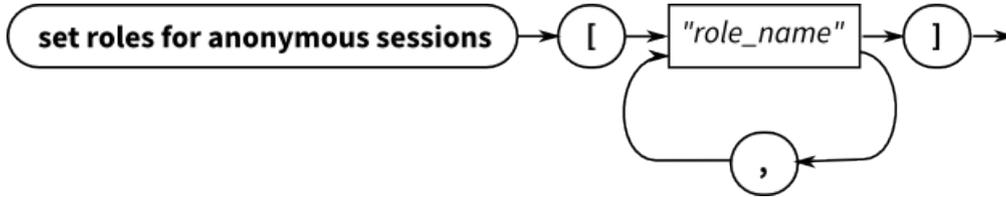
```
set roles for named sessions [' ' "role_name" [ , "role_name" ] ]'
```

Example

```
set roles for named sessions ["CLIENT"]
```

Assigning roles to an anonymous session

Railroad diagram



Backus-Naur form

```
set roles for anonymous sessions [' ' "role_name" [ , "role_name" ] ]'
```

Example

```
set roles for anonymous sessions ["CLIENT"]
```

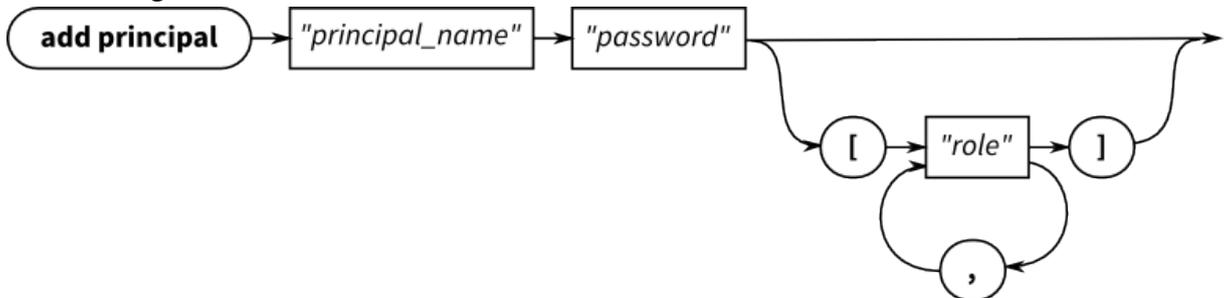
SystemAuthentication.store

The `SystemAuthentication.store` file defines the roles that are assigned by the system authentication handler to client sessions that have authenticated with a specific security principal. It also defines whether anonymous connections are allowed or denied.

The following sections each describe the syntax for a single line of the file.

Adding a principal

Railroad diagram



Backus-Naur form

```
add principal "principal_name" "password" [ [' ' "role" [ , "role" ] ] ]'
```

Example

```
add principal "user6" "passw0rd"  
add principal "user13" "passw0rd" ["CLIENT", "TOPIC_CONTROL"]
```

The password is passed in as plain text, but is stored in the system authentication store as a secure hash.

Removing a principal

Railroad diagram



Backus-Naur form

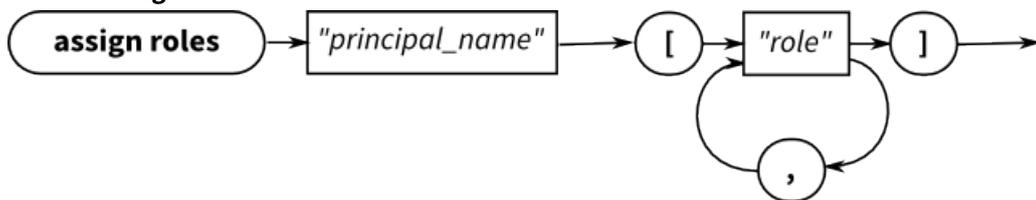
`remove principal " principal_name "`

Example

```
remove principal "user25"
```

Assigning roles to a principal

Railroad diagram



Backus-Naur form

`assign roles " principal_name " '[' " role " [, " role "]'`

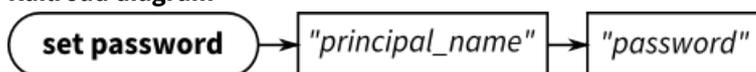
Example

```
assign roles "agent77" ["CLIENT", "CLIENT_CONTROL"]
```

When you use this command to assign roles to a principal, it overwrites any existing roles assigned to that principal. Ensure that all the roles you want the principal to have are listed in the command.

Setting the password for a principal

Railroad diagram



Backus-Naur form

`set password " principal_name " " password "`

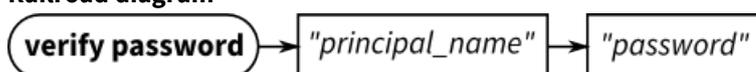
Example

```
set password "user1" "passw0rd"
```

The password is passed in as plain text, but is stored in the system authentication store as a secure hash.

Verifying the password for a principal

Railroad diagram



Backus-Naur form

`verify password " principal_name " " password "`

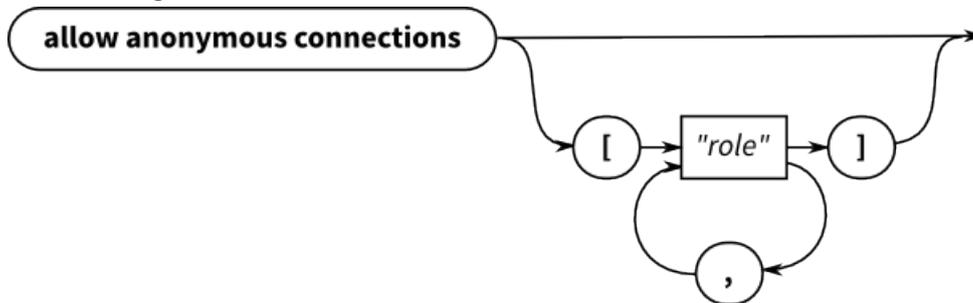
Example

```
verify password "user1" "passw0rd"
```

The password is passed in as plain text, but is stored in the system authentication store as a secure hash.

Allowing anonymous connections

Railroad diagram



Backus-Naur form

```
allow anonymous connections ['["role" [, "role"]']']
```

Example

```
allow anonymous connections [ "CLIENT" ]
```

Denying anonymous connections

Railroad diagram



Backus-Naur form

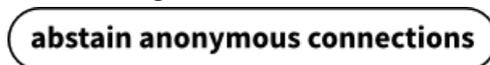
```
deny anonymous connections
```

Example

```
deny anonymous connections
```

Abstaining from providing a decision about anonymous connections

Railroad diagram



Backus-Naur form

```
abstain anonymous connections
```

Example

```
abstain anonymous connections
```

Configuring logging on the Diffusion server

Your Diffusion installation provides a default logging framework and the log4j2 logging framework. Configure the Diffusion server to use your preferred framework.

The Diffusion server uses the JAR file located at `lib/slf4j-binding.jar` as its logging framework. When you first install your Diffusion server, the logging framework used is the Diffusion default logging.

Use log4j2

The `log4j-slf4j-impl-version.jar` file controls the log4j2 logging. This file is included in the Diffusion installation in the `lib/thirdparty` directory.

To use log4j2 instead of the default Diffusion logging implementation, copy `lib/thirdparty/log4j-slf4j-impl-version.jar` to `lib/slf4j-binding.jar`.

Configure the log4j2 logging framework with the `log4j2.xml` configuration file.

Use the default logging

To revert to the standard Diffusion logging implementation, copy `lib/diffusion-slf4j-binding.jar` to `lib/slf4j-binding.jar`.

Configure the default logging framework with the `Logs.xml` configuration file.

Use another SLF4J implementation

To use an alternative SLF4J implementation, remove the `lib/slf4j-binding.jar` and add the appropriate classes for the alternative implementation to the Diffusion server classpath.

Note: Alternative implementations of SLF4J are not supported for production use.

Related reference

[Logging](#) on page 784

Diffusion uses the Simple Logging Facade for Java (SLF4J) API to log messages from the Diffusion server or from publishers running on the Diffusion server. SLF4J separates the logging of messages in the Diffusion server from the logging framework. This separation enables you to configure an independent back-end implementation to format and write out the log messages.

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

Configuring default logging

To use the default logging, ensure that the Diffusion logging JAR is at `lib/slf4j-binding.jar`. The default logging implementation is already located here when you first install the Diffusion server. Use the `Logs.xml` configuration file to configure the behavior of the Diffusion default logging.

Log messages created by the Diffusion server, and by publishers deployed to the server, are filtered by the configuration in `etc/Logs.xml`.

You can configure the following aspects of logging:

- The level of logging to the console
- The level of logging to a file
- The name and location of the file
- Whether the log files rotate based on time or file size or both
- The time interval to use to rotate the files
- The file size to use to rotate the files
- The number of old log files to keep

Warning: Logging can use considerable CPU resources. In a production environment, enable only significant log messages (`INFO` and above). Performance degrades significantly when running at finer logging levels as more messages are produced, each requiring processing.

Logging on the Diffusion server cannot be configured using the configuration API. The `LoggingConfig` object is read-only.

Related reference

[Logging](#) on page 784

Diffusion uses the Simple Logging Facade for Java (SLF4J) API to log messages from the Diffusion server or from publishers running on the Diffusion server. SLF4J separates the logging of messages in the Diffusion server from the logging framework. This separation enables you to configure an independent back-end implementation to format and write out the log messages.

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

Logs.xml

This file specifies the schema for the log properties used to configure the Diffusion default logging back-end. If you use a different logging back-end, this file is ignored.

logs

Properties defining logging options.

The following table lists the elements that an element of type `logs` can contain:

Name	Type	Description	Min occurs	Max occurs
console-log-level	push:log-level	The log level to start console logging at. Can be ERROR, WARN, INFO, DEBUG, or TRACE. If a value is not specified, a default of INFO is used.	0	1
log-message-data	push:boolean	DEPRECATED since 5.5 : Use log-message-data in Server.xml instead.	0	1
server-log	push:string	The log to use for the server. This must specify the name of a configured log definition.	1	1
default-log-directory	push:string	The default log folder for all logs, although this can be over-ridden for each log.	1	1
async-logging	push:boolean	Indicates whether logging is asynchronous. Asynchronous logging is performed by a separate thread as opposed to being performed in-line by the logging thread. This is normally set to true for performance reasons, but asynchronous logging might cause problems in some OS environments. This element provides the option to turn asynchronous logging off, if so advised. If a value is not specified, a default of true is used.	0	1
logging-queue-size	push:positiveInt	The size of the asynchronous logging queue. In normal cases, leave this value at the default value of 128k entries.	0	1
thread-name-logging	push:boolean	Indicates whether the thread name is logged with messages. If this is not specified, thread names are logged.	0	1
log	log	A log definition.	0	unbounded

log

A log definition.

The following table lists the attributes that an element of type `log` can have:

Name	Type	Description	Required
name		Name of the log definition	true
rotation-period	push:positiveNonZeroInt	Time period that the log exists for before being rotated. This is a positive non-zero integer, with unit specified by rotation-unit. If a rotation-period is specified, the value of file-append must be false.	false
rotation-unit	push:timeunit	A time unit to specify the unit used alongside rotation-period. This can be "day(s)", "hour(s)", "minute(s)".	false

The following table lists the elements that an element of type `log` can contain:

Name	Type	Description	Min occurs	Max occurs
log-directory	<code>push:string</code>	The name of the directory to which this log file is written. If a value is not specified, the default-log-directory is used.	0	1
file-pattern	<code>push:string</code>	This is used to specify the name of the log file. The following values can be used within the pattern. "/" - the local pathname separator. "%t" - the system temporary directory. "%g" - the generation number to distinguish rotated logs. "%h" - the value of the "user.home" system property. "%s" - the system type - for example, 'Diffusion'. "%n" - the system name as defined in Server.xml. "%d" - the date as specified in diffusion.properties (date.format), this is included when using daily rotation. "%%" - translates to a single percent sign "%". If a log file name is not specified, a default of "%s.log" is used.	0	1
level	<code>push:log-level</code>	Specifies the starting log level. This can be ERROR, WARN, INFO, DEBUG, or TRACE. If a value is not specified, a default of INFO is used.	0	1
xml-format	<code>push:boolean</code>	Indicates whether the log file is output in XML format. If a value is not specified, a default of false is used.	0	1
date-format	<code>push:string</code>	Specifies a date format to name a log. Specify the format according to the Java SimpleDateFormat specification. If a format is not specified, a default of "yyyy-MM-dd" is used.	0	1
file-limit	<code>push:bytes</code>	Specifies an approximate maximum amount to write (in bytes) to any one log file. If this is zero, there is no limit. If a value is not specified, a default of 0 is used.	0	1
file-append	<code>push:boolean</code>	Specifies whether log records are appended to existing log files. If a rotation-period is specified, the value of file-append must be false. If a value is not specified, a default of false is used and log files are overwritten.	0	1
file-count	<code>push:positiveNon</code>	Specifies the number of log files to use. Must be at least 1. If a value is not specified, a default of 1 is used.	0	1

Name	Type	Description	Min occurs	Max occurs
rotate-daily	push:boolean	DEPRECATED: Indicates whether the log is to rotate on a daily basis. This is superseded by rotation-period.	0	1

Configuring log4j2

To use log4j2, replace the default logging JAR file with the log4j2 JAR file. Use the `log4j2.xml` configuration file to configure the behavior of log4j2.

When the Diffusion server is configured to use the log4j2 logging framework, the Diffusion server ignores the configuration in the `Logs.xml` file. Instead, it uses the `log4j2.xml` configuration file.

The `log4j2.xml` configuration file is located in the `etc` directory of your Diffusion installation. For more information about how to use this file to configure log4j2, see the log4j2 documentation: <http://logging.apache.org/log4j/2.x/manual/configuration.html>

By default, the provided `log4j2.xml` file is configured to output log messages in the same format as used by the default logging framework. In your configuration file, create a property that defines the format to output log messages in:

```
<Property name="pattern">%date{yyyy-MM-dd HH:mm:ss.SSS} | %level |
%thread | %marker | %replace{%msg}{\|}{ } | %logger%n%xEx</Property>
```

You can use this property to specify the format used by your appenders. The property `%marker` indicates the message code. For more information, see [Logging reference](#) on page 786.

By default, the provided `log4j2.xml` file is configured to append log output to the console and to a file. This is the same behavior as the default logging framework.

```
<Loggers>
  <AsyncRoot level="info" includeLocation="false">
    <AppenderRef ref="console" />
    <AppenderRef ref="file" />
  </AsyncRoot>
</Loggers>
```

You can configure other appenders to output to the log messages to different destinations. For more information about using appenders, see <https://logging.apache.org/log4j/2.x/manual/appenders.html>.

Related reference

[Logging](#) on page 784

Diffusion uses the Simple Logging Facade for Java (SLF4J) API to log messages from the Diffusion server or from publishers running on the Diffusion server. SLF4J separates the logging of messages in the Diffusion server from the logging framework. This separation enables you to configure an independent back-end implementation to format and write out the log messages.

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

Log4j2.xml

Use the `Log4j2.xml` configuration file to configure the behavior of the log4j2 logging framework.

```
<Configuration status="warn" name="Diffusion">
  <Properties>
    <Property name="diffusion.log.dir">../logs</Property>

    <!-- The log directory can be overridden using the
    system property 'diffusion.log.dir'. -->
    <Property name="log.dir">${sd:diffusion.log.dir}</
Property>

    <Property name="pattern">%date{yyyy-MM-dd HH:mm:ss.SSS} |
%level| %thread| %marker| %replace{%msg}{\|}{ }| %logger%n%xEx
</Property>
  </Properties>

  <Appenders>
    <Console name="console">
      <PatternLayout pattern="{pattern}" />
    </Console>

    <RollingRandomAccessFile name="file"
immediateFlush="false" fileName="{log.dir}/diffusion.log"
      filePattern="{log.dir}/${date:yyyy-MM}/diffusion-
%d{MM-dd-yyyy}-%i.log.gz">
      <PatternLayout pattern="{pattern}" />

      <Policies>
        <OnStartupTriggeringPolicy />
        <TimeBasedTriggeringPolicy />
        <SizeBasedTriggeringPolicy size="250 MB" />
      </Policies>

      <DefaultRolloverStrategy max="20" />
    </RollingRandomAccessFile>
  </Appenders>

  <Loggers>
    <AsyncRoot level="info" includeLocation="false">
      <AppenderRef ref="console" />
      <AppenderRef ref="file" />
    </AsyncRoot>
  </Loggers>
</Configuration>
```

Related reference

[Logging](#) on page 784

Diffusion uses the Simple Logging Facade for Java (SLF4J) API to log messages from the Diffusion server or from publishers running on the Diffusion server. SLF4J separates the logging of messages in the Diffusion server from the logging framework. This separation enables you to configure an independent back-end implementation to format and write out the log messages.

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

Logging using another SLF4J implementation

You can use other implementations of SLF4J for your logging. However, this is not supported for production use.

To use an alternative SLF4J implementation, remove the `lib/slf4j-binding.jar` and add the appropriate classes for the alternative implementation to the Diffusion server classpath.

Alternative implementations of SLF4J are not supported for production use.

Related reference

[Logging](#) on page 784

Diffusion uses the Simple Logging Facade for Java (SLF4J) API to log messages from the Diffusion server or from publishers running on the Diffusion server. SLF4J separates the logging of messages in the Diffusion server from the logging framework. This separation enables you to configure an independent back-end implementation to format and write out the log messages.

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

Configuring JMX

Use the `Management.xml` configuration file to configure Diffusion to be manageable through JMX. Use the `Publishers.xml` configuration file to configure the JMX adapter to make MBeans available through topics.

Configuring the Diffusion JMX connector server

Connect to JMX through the Diffusion connector server. This connector server is integrated with the Diffusion server and enables you to use role-based access control to define how connecting users can use the MBeans.

About this task

Diffusion binds to the specified ports to listen for connections from JMX clients such as JConsole and Java VisualVM.

Procedure

1. Optional: If you are running Diffusion on a Linux server, check that the host name is not `127.0.1.1`.

You can do this by running the following command:

```
hostname -i
```

If the output to this command is `127.0.1.1`, add an entry to `/etc/hosts` that defines the host name.

2. Edit the `etc/Management.xml` configuration file to enable and configure the management features:
 - a) Set the value of the `enabled` attribute in the `management` element to `true`.

```
<management enabled="true">
```

- b) Specify the hostname to allow JMX connections on in the `host` element.

```
<host>localhost</host>
```

The default value is `localhost`. If you set the contents of the `host` element to a value, connections are only allowed to that value. For example, a JMX connection to `localhost` is allowed, but connecting to the same system by IP address is not.

To allow JMX connections on any applicable hostname or IP address, leave the `host` element blank.

- c) Optional: Specify the ports to use for the JMX service.

```
<!-- The RMI Registry port -->  
<registry-port>1099</registry-port>  
<!-- The JMX service port -->  
<connection-port>1100</connection-port>
```

These two ports can be set to the same value, which can simplify firewall configuration.

You can use the default values:

- **1099** The RMI registry port

- **1100** The JMX service port
3. Configure the principals that are allowed to use the JMX service. You can do this in one of the following ways.
 - Update the system authentication store to assign a role with the required permissions to the principal and configure the Diffusion server to call the system authentication handler.
For more information, see [System authentication handler](#) on page 145.
 - Implement a custom authentication handler that assigns a role with the required permissions to the principal and configure the Diffusion server to call your custom authentication handler.
For more information, see [User-written authentication handlers](#) on page 143.
 4. **Note:** If you are using a firewall that employs NAT, you might still be unable to connect to Diffusion even when the JMX ports are left open.

Optional: To make a secure connection or a connection through a firewall, you can use SSH tunnelling:

- a) Establish an SSH connection to the fire-walled Diffusion server.
- b) Tunnel the RMI registry port and JMX service port through SSH.
- c) Use JMX to connect to the local ends of the tunneled ports.

Results

Use the ports you have configured to connect a JMX management console to the Diffusion server.

This connection cannot be made through SSL. However, you can use SSH tunnelling to secure your connection. For more information, see step 4 on page 602.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Using Java VisualVM](#) on page 736

You can manage Diffusion using the JMX system management console Java VisualVM.

[Using JConsole](#) on page 738

You can manage Diffusion using the JMX system management console JConsole.

Configuring a remote JMX server connector

Connect to JMX through a remote connector to the JVM that runs the Diffusion. This connector is not integrated with the Diffusion server security and you must configure additional security in the JVM.

About this task

Important: We recommend that you use the Diffusion connector server to connect to the JMX service. For more information, see [Configuring the Diffusion JMX connector server](#) on page 601.

The JVM that runs Diffusion accepts remote connections from JMX clients such as JConsole and Java VisualVM.

Procedure

1. Configure security for your remote JMX connection.

For more information, see <https://docs.oracle.com/javase/8/technotes/guides/management/agent.html>.

The security users and roles defined for the JVM do not integrate with the security provided by the Diffusion server

2. When starting Diffusion, set the properties required for your remote JMX connection.

For more information, see <https://docs.oracle.com/javase/8/technotes/guides/management/agent.html>.

3. **Note:** If you are using a firewall that employs NAT, you might still be unable to connect to Diffusion even when the JMX ports are left open.

Optional: To make a secure connection or a connection through a firewall, you can use SSH tunnelling:

- a) Establish an SSH connection to the fire-walled Diffusion server.
- b) Tunnel the RMI registry port and JMX service port through SSH.
- c) Use JMX to connect to the local ends of the tunneled ports.

Results

Use the ports you have configured to connect a JMX management console to the Diffusion server. These connections can be made over SSL.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Using Java VisualVM](#) on page 736

You can manage Diffusion using the JMX system management console Java VisualVM.

[Using JConsole](#) on page 738

You can manage Diffusion using the JMX system management console JConsole.

Configuring a local JMX connector server

Connect to JMX through a local connector to the JVM that runs the Diffusion. This connector is not integrated with the Diffusion server security and you must configure additional security in the JVM.

About this task

The JVM that runs Diffusion accepts local connections from JMX clients such as JConsole and Java VisualVM.

Procedure

Review the JVM documentation for any actions to take before connecting your JMX client.

For more information, see <https://docs.oracle.com/javase/8/technotes/guides/management/agent.html>.

Results

You can connect a JMX management console running on the same server as Diffusion to the JVM.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Using Java VisualVM](#) on page 736

You can manage Diffusion using the JMX system management console Java VisualVM.

[Using JConsole](#) on page 738

You can manage Diffusion using the JMX system management console JConsole.

Management.xml

This file specifies the schema for the management properties that enable JMX access over an RMI JMXConnectorServer.

management

The management configuration.

The following table lists the attributes that an element of type `management` can have:

Name	Type	Description	Required
enabled	push:boolean	Specifies if an RMI JMXConnectorServer is enabled, making JMX remotely available.	true

The following table lists the elements that an element of type `management` can contain:

Name	Type	Description	Min occurs	Max occurs
host	push:string	The local interface used for the RMI registry and the JMX service. Empty values declare that the RMI registry binds to all local network interfaces.	0	1
registry-port	push:port	The RMI registry port. If a value is not specified, a default of 1099 is used.	0	1
connection-port	push:port	The JMX service port. If a value is not specified, a default of 1100 is used.	0	1
assigned-roles	assigned-roles	Additional roles granted to principals that authenticate successfully. Configuring roles for JMX-only users here is DEPRECATED since 5.9. Used to grant JMX-only users a base set of permissions.	0	1
users	users	The management users. Registering JMX-only users here is DEPRECATED. Instead configure appropriate users in the system authentication store, or with a custom authentication handler, and assign the users the role ADMINISTRATOR or VIEW_SERVER based upon whether they need full access to JMX operations or only to monitor the server.	0	1

assigned-roles

Assigned security roles.

The following table lists the elements that an element of type `assigned-roles` can contain:

Name	Type	Description	Min occurs	Max occurs
role	<code>push:string</code>	A security role.	1	unbounded

users

Management users.

The following table lists the elements that an element of type `users` can contain:

Name	Type	Description	Min occurs	Max occurs
user	<code>user</code>	A user that can use the JMX interface.	0	unbounded

user

Management user.

The following table lists the elements that an element of type `user` can contain:

Name	Type	Description	Min occurs	Max occurs
name	<code>push:string</code>	User name for JMX credentials.	1	1
password	<code>push:string</code>	Password for JMX credentials.	1	1
read-only	<code>push:boolean</code>	Specify if user has read-only access. If this value is false, the user is assigned their standard roles, the roles configured for assigned-roles, and the JMX_ADMINISTRATOR role. By default, JMX_ADMINISTRATOR is configured to provide full administrator access. Otherwise, the user is assigned only their standard roles and the assigned-roles, which may not grant sufficient permissions to perform operations that affect the state of the server.	1	1

Configuring the JMX adapter

The JMX adapter can reflect JMX MBeans their properties and notifications as topics. Configure the JMX adapter using the `Publishers.xml` configuration file.

Before you begin

The JMX adapter is packaged in the Diffusion publisher. The Diffusion publisher must be running for the JMX adapter be enabled.

About this task

You can configure the adapter to reflect the state of JMX MBeans and MXBeans as topics. These MBeans can be built-in, Diffusion, or third-party in origin.

Many statistics are available as MBean properties, for example, CPU load, OS version, number of file-descriptors, threads. Making these statistics available as topics to Diffusion clients makes possible the implementation of system monitoring solutions to the web, and all other Diffusion platforms.

Note: Publishing MBean data to topics can constitute a security risk. Ensure that crucial information about your Diffusion server is protected by permissions.

Procedure

1. Add the following properties to the `<publisher name="Diffusion">` section of the `Publishers.xml` configuration file located in the `etc` directory of your Diffusion installation.
 - a) Use the `JMSAdapter.enabled` property to enable the JMX adapter.

```
<property name="JMSAdapter.enabled">true</property>
```

- b) Use the `JMSAdapter.refreshFrequency` property to specify how often, in milliseconds, the data on the topics is updated.

```
<property name="JMSAdapter.refreshFrequency">3000</property>
```

The default value is 3 seconds.

- c) Use the `JMSAdapter.mbeans` property to specify which MBeans to reflect as topics.

Specify the MBeans using `ObjectName` format. For more information, see <https://docs.oracle.com/javase/7/docs/api/javax/management/ObjectName.html>

Specify each `ObjectName` on a new line.

```
<property name="JMSAdapter.mbeans">java.nio:*
java.lang:*
java.util.logging:*
com.pushtechology.diffusion:*</property>
```

2. Restart the Diffusion server to reload the configuration.

Results

The specified MBeans and MXBeans are reflected as topics in the Diffusion/MBeans branch of the topic tree.

Related concepts

[The JMX adapter](#) on page 751

The JMX adapter reflects JMX MBeans and their properties and notifications as topics.

Publishers.xml

This file specifies the schema for the publisher properties.

publishers

The set of publishers that the Diffusion server is aware of at startup.

The following table lists the elements that an element of type `publishers` can contain:

Name	Type	Description	Min occurs	Max occurs
publisher	publisher	A publisher definition.	0	unbounded

publisher

A publisher definition.

The following table lists the attributes that an element of type `publisher` can have:

Name	Type	Description	Required
name	push:string	The publisher name.	true

The following table lists the elements that an element of type `publisher` can contain:

Name	Type	Description	Min occurs	Max occurs
topics	push:string	An optional, comma-separated list of topic names specifying topics to be automatically created for the publisher as it is started. This technique does not allow for topics to be set up with data and so it is more usual to define the topics you require in the <code>initialLoad</code> method of the <code>Publisher</code> . This property remains mostly for backwards compatibility.	0	1
class	push:string	The full class name of a Java class that implements the publisher. This class must extend the Java API <code>Publisher</code> class and provide implementations of methods as required. The class file must be available on the classpath of the Diffusion server (or in the configured <code>usr-lib</code> or <code>ext</code> folder).	1	1
enabled	push:boolean	By default, the publisher is loaded as the server starts. By setting this to false, the publisher is not loaded.	0	1
start	push:boolean	By default, the publisher is started after it is loaded. By specifying this as false, the publisher can be loaded but not started and then can be started later using JMX.	0	1
topic-aliasing	push:boolean	Specifies whether topic aliasing is turned on for all topics created by the publisher. If the value is true, a short topic alias is transmitted in delta messages instead of the full topic name. By default, this is true, but because there are certain limitations when using topic aliasing there might be situations where you might want to turn it off.	0	1

Name	Type	Description	Min occurs	Max occurs
ack-timeout	push:millis	This specifies the default ACK (message acknowledgment) timeout value (in milliseconds) to use for messages sent from the publisher that require acknowledgment and do not have a timeout explicitly specified. If a value is not specified, a default of 1s is used.	0	1
auto-ack	push:boolean	Indicates whether to automatically acknowledge messages sent from clients to the publisher requiring acknowledgment. By default, this is false so messages requiring acknowledgment must be manually acknowledged by the publisher.	0	1
subscription-policy-file	push:string	Path of a subscription validation policy file. If this value is specified, the file is used to validate client subscriptions to topics owned by the publisher.	0	1
stop-server-if-not-loaded	push:boolean	If this is set to true and the publisher fails to load, the Diffusion server stops. By default, this is false.	0	1
log-level	push:log-level	Specifies the log level for the publisher. If this value is not specified, the publisher logs at the default log level.	0	1
server	server	A specification of a server that is automatically connected to by the publisher when it starts. DEPRECATED : Since 5.9 - will be removed at a future release.	0	unbounded
web-server	web-server	If the publisher has associated web content, it can be deployed with the publisher by specifying this property.	0	1
launch	launch	Launch detail describes how the publisher might be accessed externally, if it has an associated webpage.	0	unbounded
property	property	A property available to the publisher. This can be used to configure publisher-specific variables or parameters.	0	unbounded

launch

Launch detail.

The following table lists the attributes that an element of type `launch` can have:

Name	Type	Description	Required
name	push:string	The launcher name.	true

Name	Type	Description	Required
category	push:string	An optional category to which this launcher belongs. For example, "demo" for the Diffusion demo landing page.	false

The following table lists the elements that an element of type `launcher` can contain:

Name	Type	Description	Min occurs	Max occurs
description	push:string	A short description of this launcher.	0	1
url	push:string	The URL at which a webpage associated with this publisher can be found.	1	1
icon	push:string	A URL or path at which an icon representing this launcher can be reached.	0	1

property

A publisher property.

The following table lists the attributes that an element of type `property` can have:

Name	Type	Description	Required
name	push:string	The property value	true
type	push:string	An optional property type. Usage of this is implementation specific.	false

credentials

Credentials for server connection.

The following table lists the elements that an element of type `credentials` can contain:

Name	Type	Description	Min occurs	Max occurs
username	push:string	User name.	0	1
password	push:string	Password.	0	1

server

The following table lists the attributes that an element of type `server` can have:

Name	Type	Description	Required
name	push:string	Server definition name.	true

The following table lists the elements that an element of type `server` can contain:

Name	Type	Description	Min occurs	Max occurs
host	push:string	The host name or IP address of the server.	1	1

Name	Type	Description	Min occurs	Max occurs
port	push:port	The port number that the server is listening on for publisher client connections from other publishers.	1	1
ssl	push:boolean	If this value is true, the connection to the server is a secure connection over SSL. In this case the specified port must represent an SSL client connector at the server. The keystore properties must also be supplied for secure connections. By default, this is false.	0	1
keystore-file-location	push:string	The path of the keystore file defining the SSL context. This is ignored if ssl=false, but mandatory if it is true.	0	1
keystore-password	push:string	The keystore password. This is ignored if ssl=false, but mandatory if it is true.	0	1
input-buffer-size	push:bytes	Specifies the size of the input buffer to use for the connection with the server. This is used to receive messages from the server. Set this to the same size as the output buffer used at the server.	1	1
output-buffer-size	push:bytes	The size of the output buffer to use for the connection with the server. This is used to send messages to the server. Set this to the same size as the input buffer used by the server.	1	1
fail-policy	push:string	This specifies what happens if the publisher fails to connect to the server. 'default' means that if unable to connect, no action is taken and it is the publisher's responsibility to handle this. 'close' means that if unable to connect to the server, the publisher closes. 'retry' means that if unable to connect, the connection is automatically retried at intervals as specified by the retry-interval property.	1	1
retry-interval	push:millis	If the fail-policy for a server is 'retry', this is the interval at which the connection to the server is retried. If this value is not specified, a default of 5s is used.	0	1
credentials	credentials	Credentials to use for the server connection. If this value is not specified, no credentials are passed on connection.	0	1
queue-definition	push:string	Optional outbound queue definition for this server connection. The definition must exist in Server.xml. This defines the	0	1

Name	Type	Description	Min occurs	Max occurs
		queue to use for outbound messages from the publisher to the server. If this value is not specified, the default queue definition in Server.xml is used.		

web-server

A web server definition.

The following table lists the elements that an element of type `web-server` can contain:

Name	Type	Description	Min occurs	Max occurs
virtual-host	<code>push:string</code>	The name of the virtual host to deploy to. If this value is not supplied, <code>default-files-default</code> is used.	0	1
alias-file	<code>push:string</code>	The alias file to use for this publisher	1	1

Configuring replication

Use the `Replication.xml` configuration file to configure the Diffusion server to replicate sessions and topics.

You can also use the `hazelcast.xml` configuration file to configure your datagrid provider.

Topic aliasing

Topic aliasing is not supported with replication. Ensure that it is disabled on all servers in the cluster.

When starting any servers in the cluster ensure that `diffusion.publishers.v5.topic.aliasing.disabled` is set to `true`. Edit the `diffusion.sh` or `diffusion.bat` file to set it as a system property when starting the Diffusion server:

```
-Ddiffusion.publishers.v5.topic.aliasing.disabled=true
```

Configuring the Diffusion server to use replication

You can configure replication by editing the `etc/Replication.xml` files of your Diffusion servers.

About this task

Ensure that you use the same replication configuration on all of the Diffusion servers in your cluster.

Procedure

1. Edit the `Replication.xml` file to configure replication.

```
<replication enabled="true">
  <provider>HAZELCAST</provider>
  <sessionReplication enabled="true" />
  <topicReplication enabled="true">
    <topics>
      <topicPath>foo/bar</topicPath>
    </topics>
  </topicReplication>
</replication>
```

```
</topics>
  </topicReplication>
</replication>
```

- In the `replication` element, set `enabled` to `true` to enable replication.
- In the `sessionReplication` element, set `enabled` to `true` to configure the server to reflect client session information into the datagrid.
- In the `topicReplication` element, set `enabled` to `true` to configure the server to reflect topic information and topic data into the datagrid.
- Inside the `topics` element, use one or more `topicPath` elements to define the topics to which to apply topic replication and failover of the active update source.

The content of the `topicPath` element is a path to a single topic. Topic replication and failover of the active update source are applied to the topic defined by the path and all topics below it in the topic tree.

Unlike a topic selector, the topic path does not contain any leading or trailing characters. For example, use `<topicPath>foo/bar</topicPath>` to select the topic `foo/bar`.

The `topicPath` elements also define which sections of the topic tree are have update sources created for them by the server.

2. Restart the Diffusion server to load the configuration.
3. Ensure that your clients are configured to reconnect if they lose their connection to the server.

Related reference

[Session replication](#) on page 105

You can use session replication to ensure that if a client connection fails over from one server to another the state of the client session is maintained.

[Topic replication](#) on page 108

You can use topic replication to ensure that the structure of the topic tree, topic definitions, and topic data are synchronized between servers.

[Failover of active update sources](#) on page 110

You can use failover of active update sources to ensure that when a server that is the active update source for a section of the topic tree becomes unavailable, an update source on another server is assigned to be the active update source for that section of the topic tree. Failover of active update sources is enabled for any sections of the topic tree that have topic replication enabled.

[Configuring your datagrid provider](#) on page 612

You can configure how the built-in Hazelcast datagrid replicates data within your solution architecture.

[Replication.xml](#) on page 614

This file specifies the schema for the replication properties.

Configuring your datagrid provider

You can configure how the built-in Hazelcast datagrid replicates data within your solution architecture.

Configuring Hazelcast

By default, the Hazelcast node in your Diffusion server multicasts to all other Hazelcast nodes in your network.

We recommend that in a production environment you disable multicast and explicitly define the nodes in your Hazelcast cluster. This configuration is more secure and removes the risk of nodes in your development environment connecting to the production environment and interfering with the production data.

To define which Hazelcast nodes can communicate with each other, use the `hazelcast.xml` configuration file.

The following example shows the structure of the `hazelcast.xml` file:

```
<hazelcast xsi:schemaLocation="http://www.hazelcast.com/schema/config
  http://www.hazelcast.com/schema/config/hazelcast-config-3.5.xsd"
  xmlns="http://www.hazelcast.com/schema/config" xmlns:xsi="http://
  www.w3.org/2001/XMLSchema-instance">

  <properties>
    <property name="hazelcast.logging.type">slf4j</property>
    <property name="hazelcast.version.check.enabled">>false</
  property>
  </properties>

  <network>
    <join>
      <!-- <multicast enabled="true" /> -->
      <tcp-ip enabled="true">
        <member>node1.example.com</member>
        <member>203.0.113.1</member>
        <member>203.0.113.2:5757</member>
        <member>203.0.113.3-7</member>
      </tcp-ip>
    </join>

  </network>
</hazelcast>
```

This example configuration disables the multicast capability and defines the Hazelcast nodes that can be connected to.

The Hazelcast nodes can be defined by hostname, by IP address, or by IP range. The default port used by Hazelcast is 5701. If you want to connect on a different port, you can specify this when you define the node, using the format `host:port`.

Ensure that the `hazelcast.xml` file is on the Diffusion server classpath. For example, by putting the file in the `diffusion_installation/data` directory. Restart the Diffusion server to load the configuration.

For more information about using the `hazelcast.xml` file to configure Hazelcast, see the [Hazelcast™ Reference Manual](#).

Diagnosing problems with Hazelcast

If you enable logging for Hazelcast, you can use the log files to diagnose problems with Hazelcast.

To enable logging, include the following line in your `hazelcast.xml` file:

```
<property name="hazelcast.logging.type">slf4j</property>
```

Ensure that the `hazelcast.xml` file is on the Diffusion server classpath. For example, by putting the file in the `diffusion_installation/data` directory. Restart the Diffusion server to load the configuration.

You can also enable logging by starting the Diffusion server that contains the node with the following parameter `-Dhazelcast.logging.type=slf4j`

You can enable JMX for your Hazelcast nodes and use a JMX tool to examine the MBeans.

To enable JMX for a Hazelcast node, include the following line in your `hazelcast.xml` file:

```
<property name="hazelcast.jmx">true</property>
```

Ensure that the `hazelcast.xml` file is on the Diffusion server classpath. For example, by putting the file in the `diffusion_installation/data` directory. Restart the Diffusion server to load the configuration.

You can also enable JMX by starting the Diffusion server that contains the node with the following parameter `-Dhazelcast.jmx=true`

For more information about using Hazelcast, see the [Hazelcast™ Reference Manual](#).

Related tasks

[Configuring the Diffusion server to use replication](#) on page 611

You can configure replication by editing the `etc/Replication.xml` files of your Diffusion servers.

Related reference

[Session replication](#) on page 105

You can use session replication to ensure that if a client connection fails over from one server to another the state of the client session is maintained.

[Topic replication](#) on page 108

You can use topic replication to ensure that the structure of the topic tree, topic definitions, and topic data are synchronized between servers.

[Failover of active update sources](#) on page 110

You can use failover of active update sources to ensure that when a server that is the active update source for a section of the topic tree becomes unavailable, an update source on another server is assigned to be the active update source for that section of the topic tree. Failover of active update sources is enabled for any sections of the topic tree that have topic replication enabled.

[Replication.xml](#) on page 614

This file specifies the schema for the replication properties.

Replication.xml

This file specifies the schema for the replication properties.

replication

Properties defining replication.

The following table lists the attributes that an element of type `replication` can have:

Name	Type	Description	Required
enabled	push:boolean	Specifies whether replication is enabled for this server.	true

The following table lists the elements that an element of type `replication` can contain:

Name	Type	Description	Min occurs	Max occurs
provider	push:string	The type of replication provider to use to replicate the data. Currently only Hazelcast is supported.	1	1
sessionReplication	sessionReplication	The definition for session replication	1	1
topicReplication	topicReplication	The definition for topic replication	1	1

sessionReplication

Properties defining session replication.

The following table lists the attributes that an element of type `sessionReplication` can have:

Name	Type	Description	Required
enabled	push:boolean	Specifies whether session replication is enabled for this server.	true

topicReplication

Properties defining topic replication.

The following table lists the attributes that an element of type `topicReplication` can have:

Name	Type	Description	Required
enabled	push:boolean	Specifies whether topic replication is enabled for this server.	true

The following table lists the elements that an element of type `topicReplication` can contain:

Name	Type	Description	Min occurs	Max occurs
topics	topics	The topics that are configured to use replication.	1	1

topics

Properties defining the topics to replicate.

The following table lists the elements that an element of type `topics` can contain:

Name	Type	Description	Min occurs	Max occurs
topicPath	push:string	A topic path that identifies the root of a tree that will be replicated by this server.	0	unbounded

Related tasks

[Configuring the Diffusion server to use replication](#) on page 611

You can configure replication by editing the `etc/Replication.xml` files of your Diffusion servers.

Related reference

[Session replication](#) on page 105

You can use session replication to ensure that if a client connection fails over from one server to another the state of the client session is maintained.

[Topic replication](#) on page 108

You can use topic replication to ensure that the structure of the topic tree, topic definitions, and topic data are synchronized between servers.

[Failover of active update sources](#) on page 110

You can use failover of active update sources to ensure that when a server that is the active update source for a section of the topic tree becomes unavailable, an update source on another server is assigned to be the active update source for that section of the topic tree. Failover of active update sources is enabled for any sections of the topic tree that have topic replication enabled.

[Configuring your datagrid provider](#) on page 612

You can configure how the built-in Hazelcast datagrid replicates data within your solution architecture.

Configuring the Diffusion web server

Use the `WebServer.xml` and `Aliases.xml` configuration files to configure the behavior of the Diffusion web server.

Diffusion can act as a web server by modifying the `Connectors.xml` configuration file to add a `web-server` definition to a connector. If a connector is required to serve HTTP requests, the connector requires a `web-server` definition. A valid `web-server` entry must also exist in the `WebServer.xml` configuration file.

The Diffusion web server is a lightweight web server with very basic features. It hosts the Diffusion landing page, monitoring console, and demos.

The Diffusion web server also provides the endpoint for clients connecting to the Diffusion server using HTTP-based transports.

Note: Do not use the Diffusion web server as the host for your production website. Instead use a third-party web server.

For more information about using Diffusion with third-party web servers, see [Web servers](#) on page 648.

Related concepts

[Configuring Diffusion web server security](#) on page 617

When configuring your Diffusion web server, consider the security of your solution.

[Running the Diffusion server inside of a third-party web application server](#) on page 652

Diffusion can run as a Java servlet inside any Java application server.

[Hosting Diffusion web clients in a third-party web server](#) on page 651

Host Diffusion web clients — clients written using the JavaScript, Flash, or Silverlight APIs — on a third-party web server to enable your customers to access them.

[Web servers](#) on page 648

Diffusion incorporates its own basic web server for a limited set of uses. The Diffusion server also interacts with third-party web servers that host Diffusion web clients. The Diffusion server is also capable of being run as a Java servlet inside a web application server.

[Diffusion web server](#) on page 649

Diffusion incorporates its own web server. This web server is required to enable a number of Diffusion capabilities, but we recommend that you do not use it to host your production web applications.

[Web servers](#) on page 120

Consider how to use web servers as part of your Diffusion solution.

Related reference

[WebServer.xml](#) on page 617

This file specifies the schema for the web server properties.

Configuring Diffusion web server security

When configuring your Diffusion web server, consider the security of your solution.

Digest authentication

Digest authentication can be utilized to negotiate credentials with a user's web browser. It is applied to specific directories on your web site. The protection of one directory automatically applies protection to all lower directories as well.

Use the `realms` element in the `WebServer.xml` configuration file to add new realms to a virtual host and to store the user's name and the passwords.

HTTP deployment

You can deploy DAR files to a Diffusion server through a web service. This web service does not run by default, but can be enabled for your test environment by editing the provided `WebServer.xml` configuration file to include the commented out `deploy-service`.

Warning: Access to the deploy web service is not restricted. Do not enable this web service in your production environment unless you restrict access to the `diffusion-url/deploy` URL by other means, for example through your firewall setup.

Related concepts

[Configuring the Diffusion web server](#) on page 616

Use the `WebServer.xml` and `Aliases.xml` configuration files to configure the behavior of the Diffusion web server.

[Web servers](#) on page 648

Diffusion incorporates its own basic web server for a limited set of uses. The Diffusion server also interacts with third-party web servers that host Diffusion web clients. The Diffusion server is also capable of being run as a Java servlet inside a web application server.

[Diffusion web server](#) on page 649

Diffusion incorporates its own web server. This web server is required to enable a number of Diffusion capabilities, but we recommend that you do not use it to host your production web applications.

[Web servers](#) on page 120

Consider how to use web servers as part of your Diffusion solution.

Related reference

[WebServer.xml](#) on page 617

This file specifies the schema for the web server properties.

WebServer.xml

This file specifies the schema for the web server properties.

web-servers

Definitions of one or more web servers.

The following table lists the elements that an element of type `web-servers` can contain:

Name	Type	Description	Min occurs	Max occurs
web-server	web-server	Web server definition.	0	unbounded

web-server

Web server definition.

The following table lists the attributes that an element of type `web-server` can have:

Name	Type	Description	Required
name	push:string	Name of the web server definition.	true

The following table lists the elements that an element of type `web-server` can contain:

Name	Type	Description	Min occurs	Max occurs
client-service	client-service	Optional client service.	0	1
http-service	http-service	HTTP service.	0	unbounded
file-service	file-service	Optional file service.	0	1

virtual-host

Virtual host definition.

The following table lists the attributes that an element of type `virtual-host` can have:

Name	Type	Description	Required
name	push:string	Virtual host name.	true
debug	push:boolean	Debug flag. Set to true for debugging. Default is false.	false

The following table lists the elements that an element of type `virtual-host` can contain:

Name	Type	Description	Min occurs	Max occurs
host	push:string	Specifies the host which the virtual host is to serve, for example, <code>download.pushtechnology.com</code> or <code>*</code> for all.	1	1
document-root	push:string	The physical directory for this virtual host. If a relative path is configured, it is resolved relative to the Diffusion home directory.	1	1
home-page	push:string	The default home page. This file is used with directory browsing.	1	1
error-page	push:string	This is used to control the 404 response. The server looks for one of these files in the directory of the request. If the file does not exist, it looks for this file in the virtual directory. If the file is not	0	1

Name	Type	Description	Min occurs	Max occurs
		supplied or the file does not exist, a standard 404 response HTML document is sent.		
static	push:boolean	If this is set to true, after loading the resource once, the file system is not checked again. This improves performance for simple static usage. By default this is false.	0	1
minify	push:boolean	Set to true to minify the html. This happens before the file is compressed. By default this is false.	0	1
cache	cache	The virtual host cache configuration.	1	1
compression-threshold	push:bytes	All HTTP responses over this size are compressed. If not specified, a default value of 512 is used.	0	1
alias-file	push:string	Optionally specifies an alias file. This allows for URL aliasing if required. If a relative path is configured, it is resolved relative to the Diffusion configuration directory.	0	1
realms	realms	Virtual host realms.	0	1

realms

Virtual host realms.

The following table lists the elements that an element of type `realms` can contain:

Name	Type	Description	Min occurs	Max occurs
realm	realm	A virtual host realm.	0	unbounded

realm

A virtual host realm.

The following table lists the attributes that an element of type `realm` can have:

Name	Type	Description	Required
name	push:string	Virtual host realm name.	true
path	push:string	Virtual host realm path.	true

The following table lists the elements that an element of type `realm` can contain:

Name	Type	Description	Min occurs	Max occurs
users	users	Virtual host realm users.	0	1

users

Virtual host realm users.

The following table lists the elements that an element of type `users` can contain:

Name	Type	Description	Min occurs	Max occurs
user	<code>user</code>	Virtual host realm user.	1	unbounded

user

Virtual host realm user.

The following table lists the attributes that an element of type `user` can have:

Name	Type	Description	Required
name	<code>push:string</code>	Virtual host realm user name.	true
password	<code>push:string</code>	Virtual host realm user password.	true

cache

Virtual host cache.

The following table lists the attributes that an element of type `cache` can have:

Name	Type	Description	Required
debug	<code>push:boolean</code>	Set true to debug the cache. If a value is not specified, a default of false is used.	false

The following table lists the elements that an element of type `cache` can contain:

Name	Type	Description	Min occurs	Max occurs
file-size-limit	<code>push:bytes</code>	If the file to be served is over this size, do not cache the entire contents, but map the file instead. If a size is not specified, a default value of 1m is used.	0	1
cache-size-limit	<code>push:bytes</code>	Total size of the cache for this web server definition. If a size is not specified, a default value of 10m is used.	0	1
file-life-time	<code>push:millis</code>	If the file has not been accessed within the time specified, remove the entry from the cache. If a time is not specified, a default value of 1d is used.	0	1

http-service

HTTP service.

The following table lists the attributes that an element of type `http-service` can have:

Name	Type	Description	Required
name	<code>push:string</code>	HTTP service name.	true

Name	Type	Description	Required
debug	push:boolean	Set true to debug the HTTP service. If a value is not specified, a default of false is used.	false

The following table lists the elements that an element of type `http-service` can contain:

Name	Type	Description	Min occurs	Max occurs
class	push:string	The user HTTP service class name. This class must implement the <code>HTTPServiceHandler</code> interface in the web server API.	1	1
url-pattern	push:string	The pattern that the URL must match for this service to be invoked.	1	1
log	push:string	An optional log file can be specified and, if so, HTTP access can be logged. The log definition must exist in <code>Logs.xml</code> .	0	1
max-inbound-request-size	push:bytes	The maximum number of bytes that the HTTP request can have. If this is not specified, a default of the maximum message size is used.	0	1
property	property	HTTP service property.	0	unbounded

property

A property.

The following table lists the attributes that an element of type `property` can have:

Name	Type	Description	Required
name	push:string	Property name.	true
type	push:string	Optional property type.	false

file-service

File service.

The following table lists the attributes that an element of type `file-service` can have:

Name	Type	Description	Required
name	push:string	File service name.	true

The following table lists the elements that an element of type `file-service` can contain:

Name	Type	Description	Min occurs	Max occurs
virtual-host	virtual-host	Virtual host.	1	unbounded
write-timeout	push:millis	Write timeout for serving files. This does not affect HTTP clients. If a value is not specified, a default value of 3s is used.	0	1

client-service

Client service.

The following table lists the attributes that an element of type `client-service` can have:

Name	Type	Description	Required
name	push:string	Client service name.	true
debug	push:boolean	Set true to debug the client service. If a value is not specified, a default of false is used.	false

The following table lists the elements that an element of type `client-service` can contain:

Name	Type	Description	Min occurs	Max occurs
message-sequence-timeout	push:millis	This is used with HTTP clients to indicate how long to wait for a missing message in a sequence of messages before assuming it has been lost and closing the client session. If a value is not specified, a default of 2 seconds is used. If this exceeds one hour (3600000ms) a warning will be logged and the time-out will be set to one hour.	0	1
websocket-origin	push:string	This is used to control access from client web socket to Diffusion. This is a regular expression pattern that matches the origin of the request. A value of <code>"*"</code> matches anything, so all requests are allowed. If this is not specified, the service is unable to handle web socket requests.	0	1
cors-origin	push:string	This is used to control access from client web (XHR) to Diffusion. This element will enable Cross Origin Resource Sharing (CORS). This is a regular expression pattern that matches the origin of the request. A value of <code>"*"</code> matches anything, so all requests are allowed. If a value is not specified, the service cannot handle CORS requests.	0	1
websocket-secure-response	push:boolean	DEPRECATED: the value is ignored. This configuration applied to behavior defined by draft versions of the WebSocket specification that is no longer supported.	0	1
close-callback-requests	push:boolean	For Diffusion client requests this specifies whether to obey the keep-alive header or close all requests. If this is set to true, all requests are closed. If a value is not specified, a default of false is	0	1

Name	Type	Description	Min occurs	Max occurs
		used. DEPRECATED: since Diffusion 5.7. This setting will be removed in a future release.		
compression-threshold	push:bytes	Enable compression for HTTP client responses over this size. If a value is not specified, a default of 512 bytes is used.	0	1
max-inbound-request-size	push:bytes	The maximum number of bytes that the HTTP request can have. If a value is not specified, a default of the maximum message size is used.	0	1
comet-bytes-before-new-poll	push:bytes	This parameter enables you to specify the number of bytes after which a Comet connection is forced to re-establish itself. This can help to reduce the potential for memory leaks in the browser due to the long-lived nature of a Comet connection, at the expense of degraded performance due to more frequent HTTP handshakes. If this is not specified, a default value of 30 kilobytes is used.	0	1
comet-initial-message-padding	push:bytes	Some browsers do not pass on data received through a Comet connection until a minimum number of bytes have been received. This means that in the case where an initial topic load message is small, the client might never receive it. To work around this restriction, you can set a value here which ensures that the first message received is padded with extra bytes that are automatically discarded by the client library. If a value is not specified, a default of 1k is used.	0	1
disable-cookies	push:boolean	Set true to disable session cookie from being in the "Set-Cookie" header. If a value is not specified, cookies are enabled.	0	1

Related concepts

[Configuring the Diffusion web server](#) on page 616

Use the `WebServer.xml` and `Aliases.xml` configuration files to configure the behavior of the Diffusion web server.

[Configuring Diffusion web server security](#) on page 617

When configuring your Diffusion web server, consider the security of your solution.

[Web servers](#) on page 648

Diffusion incorporates its own basic web server for a limited set of uses. The Diffusion server also interacts with third-party web servers that host Diffusion web clients. The Diffusion server is also capable of being run as a Java servlet inside a web application server.

[Diffusion web server](#) on page 649

Diffusion incorporates its own web server. This web server is required to enable a number of Diffusion capabilities, but we recommend that you do not use it to host your production web applications.

[Web servers](#) on page 120

Consider how to use web servers as part of your Diffusion solution.

Aliases.xml

This file specifies the schema for the aliases properties used in a web server.

aliases

List of aliases

The following table lists the elements that an element of type `aliases` can contain:

Name	Type	Description	Min occurs	Max occurs
alias	alias	An alias definition	0	unbounded

alias

An alias definition

The following table lists the attributes that an element of type `alias` can have:

Name	Type	Description	Required
name	push:string	A name for the alias.	true

The following table lists the elements that an element of type `alias` can contain:

Name	Type	Description	Min occurs	Max occurs
source	push:string	The source URL, which can be expressed as a regular expression.	1	1
destination	push:string	The destination path.	1	1

ConnectionValidationPolicy.xml

This file specifies the schema for the connection validation policy.

connection-validation-policies

Connection validation policies

The following table lists the elements that an element of type `connection-validation-policies` can contain:

Name	Type	Description	Min occurs	Max occurs
policy	policy	A connection validation policy.	0	unbounded

policy

A connection validation policy.

The following table lists the attributes that an element of type `policy` can have:

Name	Type	Description	Required
name	push:string	Each policy must be supplied with a unique name for easy reference.	true
type	push:string	The policy type should be either "blacklist" or "whitelist". A blacklist indicates that if any of the policy rules in this policy match the incoming connection, that connection is to be rejected. A whitelist requires that at least one policy rule matches for the connection to be accepted.	true
automatic	push:boolean	Policies which are set to automatic are applied by Diffusion and the publishers do not need to perform any checks themselves. If this attribute is set to false, the policy is not applied unless it is done so by the publisher. If a value is not specified, a default of true is used.	false

The following table lists the elements that an element of type `policy` can contain:

Name	Type	Description	Min occurs	Max occurs
addresses	addresses	Connection validation policy addresses. These are addresses that are blacklisted/whitelisted.	0	1
locale	locale	Connection validation policy locale. This is locale details that are blacklisted/whitelisted.	0	unbounded

addresses

The following table lists the elements that an element of type `addresses` can contain:

Name	Type	Description	Min occurs	Max occurs
address	push:string	An IP address (or regular expression) of a connecting client.	0	unbounded
hostname	push:string	The hostname (or regular expression) of a connecting client.	0	unbounded
resolved-name	push:string	The resolved hostname (or regular expression) of a connecting client, as returned by the Whois service.	0	unbounded

locale

The following table lists the elements that an element of type `locale` can contain:

Name	Type	Description	Min occurs	Max occurs
country	push:string	The ISO country code of the connecting client, as returned by the Whois service.	0	1
language	push:string	The ISO language code of the connecting client, as returned by the Whois service.	0	1

Env.xml

This file specifies the schema for the environment properties.

env

The following table lists the elements that an element of type `env` can contain:

Name	Type	Description	Min occurs	Max occurs
property	property	Environment variable value	0	unbounded

property

The following table lists the attributes that an element of type `property` can have:

Name	Type	Description	Required
name	<code>xsd:token</code>	Name of the environment variable.	true

Mime.xml

This file specifies the schema for the mime properties.

mimes

The following table lists the elements that an element of type `mimes` can contain:

Name	Type	Description	Min occurs	Max occurs
mime	mime	Mime.	0	unbounded

mime

The following table lists the attributes that an element of type `mime` can have:

Name	Type	Description	Required
type	<code>push:string</code>	Mime type.	true

Name	Type	Description	Required
extension	push:string	Mime extension.	true

Publishers.xml

This file specifies the schema for the publisher properties.

publishers

The set of publishers that the Diffusion server is aware of at startup.

The following table lists the elements that an element of type `publishers` can contain:

Name	Type	Description	Min occurs	Max occurs
publisher	<code>publisher</code>	A publisher definition.	0	unbounded

publisher

A publisher definition.

The following table lists the attributes that an element of type `publisher` can have:

Name	Type	Description	Required
name	push:string	The publisher name.	true

The following table lists the elements that an element of type `publisher` can contain:

Name	Type	Description	Min occurs	Max occurs
topics	<code>push:string</code>	An optional, comma-separated list of topic names specifying topics to be automatically created for the publisher as it is started. This technique does not allow for topics to be set up with data and so it is more usual to define the topics you require in the <code>initialLoad</code> method of the <code>Publisher</code> . This property remains mostly for backwards compatibility.	0	1
class	<code>push:string</code>	The full class name of a Java class that implements the publisher. This class must extend the Java API <code>Publisher</code> class and provide implementations of methods as required. The class file must be available on the classpath of the Diffusion server (or in the configured <code>usr-lib</code> or <code>ext</code> folder).	1	1
enabled	<code>push:boolean</code>	By default, the publisher is loaded as the server starts. By setting this to false, the publisher is not loaded.	0	1

Name	Type	Description	Min occurs	Max occurs
start	push:boolean	By default, the publisher is started after it is loaded. By specifying this as false, the publisher can be loaded but not started and then can be started later using JMX.	0	1
topic-aliasing	push:boolean	Specifies whether topic aliasing is turned on for all topics created by the publisher. If the value is true, a short topic alias is transmitted in delta messages instead of the full topic name. By default, this is true, but because there are certain limitations when using topic aliasing there might be situations where you might want to turn it off.	0	1
ack-timeout	push:millis	This specifies the default ACK (message acknowledgment) timeout value (in milliseconds) to use for messages sent from the publisher that require acknowledgment and do not have a timeout explicitly specified. If a value is not specified, a default of 1s is used.	0	1
auto-ack	push:boolean	Indicates whether to automatically acknowledge messages sent from clients to the publisher requiring acknowledgment. By default, this is false so messages requiring acknowledgment must be manually acknowledged by the publisher.	0	1
subscription-policy-file	push:string	Path of a subscription validation policy file. If this value is specified, the file is used to validate client subscriptions to topics owned by the publisher.	0	1
stop-server-if-not-loaded	push:boolean	If this is set to true and the publisher fails to load, the Diffusion server stops. By default, this is false.	0	1
log-level	push:log-level	Specifies the log level for the publisher. If this value is not specified, the publisher logs at the default log level.	0	1
server	server	A specification of a server that is automatically connected to by the publisher when it starts. DEPRECATED : Since 5.9 - will be removed at a future release.	0	unbounded
web-server	web-server	If the publisher has associated web content, it can be deployed with the publisher by specifying this property.	0	1

Name	Type	Description	Min occurs	Max occurs
launch	launch	Launch detail describes how the publisher might be accessed externally, if it has an associated webpage.	0	unbounded
property	property	A property available to the publisher. This can be used to configure publisher-specific variables or parameters.	0	unbounded

launch

Launch detail.

The following table lists the attributes that an element of type `launch` can have:

Name	Type	Description	Required
name	push:string	The launcher name.	true
category	push:string	An optional category to which this launcher belongs. For example, "demo" for the Diffusion demo landing page.	false

The following table lists the elements that an element of type `launch` can contain:

Name	Type	Description	Min occurs	Max occurs
description	push:string	A short description of this launcher.	0	1
url	push:string	The URL at which a webpage associated with this publisher can be found.	1	1
icon	push:string	A URL or path at which an icon representing this launcher can be reached.	0	1

property

A publisher property.

The following table lists the attributes that an element of type `property` can have:

Name	Type	Description	Required
name	push:string	The property value	true
type	push:string	An optional property type. Usage of this is implementation specific.	false

credentials

Credentials for server connection.

The following table lists the elements that an element of type `credentials` can contain:

Name	Type	Description	Min occurs	Max occurs
username	push:string	User name.	0	1
password	push:string	Password.	0	1

server

The following table lists the attributes that an element of type `server` can have:

Name	Type	Description	Required
name	push:string	Server definition name.	true

The following table lists the elements that an element of type `server` can contain:

Name	Type	Description	Min occurs	Max occurs
host	push:string	The host name or IP address of the server.	1	1
port	push:port	The port number that the server is listening on for publisher client connections from other publishers.	1	1
ssl	push:boolean	If this value is true, the connection to the server is a secure connection over SSL. In this case the specified port must represent an SSL client connector at the server. The keystore properties must also be supplied for secure connections. By default, this is false.	0	1
keystore-file-location	push:string	The path of the keystore file defining the SSL context. This is ignored if <code>ssl=false</code> , but mandatory if it is true.	0	1
keystore-password	push:string	The keystore password. This is ignored if <code>ssl=false</code> , but mandatory if it is true.	0	1
input-buffer-size	push:bytes	Specifies the size of the input buffer to use for the connection with the server. This is used to receive messages from the server. Set this to the same size as the output buffer used at the server.	1	1
output-buffer-size	push:bytes	The size of the output buffer to use for the connection with the server. This is used to send messages to the server. Set this to the same size as the input buffer used by the server.	1	1
fail-policy	push:string	This specifies what happens if the publisher fails to connect to the server. 'default' means that if unable to connect, no action is taken and it is the publisher's responsibility to handle this. 'close' means that if unable to connect	1	1

Name	Type	Description	Min occurs	Max occurs
		to the server, the publisher closes. 'retry' means that if unable to connect, the connection is automatically retried at intervals as specified by the retry-interval property.		
retry-interval	push:millis	If the fail-policy for a server is 'retry', this is the interval at which the connection to the server is retried. If this value is not specified, a default of 5s is used.	0	1
credentials	credentials	Credentials to use for the server connection. If this value is not specified, no credentials are passed on connection.	0	1
queue-definition	push:string	Optional outbound queue definition for this server connection. The definition must exist in Server.xml. This defines the queue to use for outbound messages from the publisher to the server. If this value is not specified, the default queue definition in Server.xml is used.	0	1

web-server

A web server definition.

The following table lists the elements that an element of type `web-server` can contain:

Name	Type	Description	Min occurs	Max occurs
virtual-host	push:string	The name of the virtual host to deploy to. If this value is not supplied, default-files-default is used.	0	1
alias-file	push:string	The alias file to use for this publisher	1	1

Statistics.xml

This file specifies the schema for the properties defining statistics collection. The statistics are broken into sections: client, topic, server and publisher.

statistics

Properties defining statistics collection.

The following table lists the attributes that an element of type `statistics` can have:

Name	Type	Description	Required
enabled	push:boolean	A global switch to toggle collection of all statistics.	true

The following table lists the elements that an element of type `statistics` can contain:

Name	Type	Description	Min occurs	Max occurs
client-statistics	client-statistics	Optional client statistics: configures Diffusion to periodically output client statistics to a log file defined in Logs.xml. It gives a count of all of the different client types. Each counter is reset according to the configured frequency.	0	1
topic-statistics	topic-statistics	Optional topic statistics.	0	1
server-statistics	server-statistics	Optional server statistics.	0	1
publisher-statistics	publisher-statistics	Optional publisher statistics.	0	1
reporters	reporters	Optional set of StatisticsReporters to be loaded with Diffusion, which are registered with the internal StatisticsService and used to generate output.	0	1

client-statistics

The following table lists the attributes that an element of type `client-statistics` can have:

Name	Type	Description	Required
enabled	<code>push:boolean</code>	Specifies if aggregate client statistics are enabled.	true

The following table lists the elements that an element of type `client-statistics` can contain:

Name	Type	Description	Min occurs	Max occurs
log-name	<code>push:string</code>	Definition of the log in Logs.xml.	0	1
output-frequency	<code>push:millis</code>	Specifies the output frequency of the log. There is one entry per specified interval. If this is not specified, a default of 1h is used.	0	1
reset-frequency	<code>push:millis</code>	Specifies when the counters are reset. The reset interval must be a multiple of the output frequency. Zero specifies that the counters are never reset. If this is not specified, a default of 1h is used.	0	1
monitor-instances	<code>push:boolean</code>	Specifies if individual client statistics are enabled.	0	1

topic-statistics

The following table lists the attributes that an element of type `topic-statistics` can have:

Name	Type	Description	Required
enabled	<code>push:boolean</code>	Specifies if aggregate topic statistics are enabled.	true

The following table lists the elements that an element of type `topic-statistics` can contain:

Name	Type	Description	Min occurs	Max occurs
monitor-instances	<code>push:boolean</code>	Specifies if individual topic statistics are enabled.	0	1

publisher-statistics

The following table lists the attributes that an element of type `publisher-statistics` can have:

Name	Type	Description	Required
enabled	<code>push:boolean</code>	Specifies if aggregate publisher statistics are enabled.	true

The following table lists the elements that an element of type `publisher-statistics` can contain:

Name	Type	Description	Min occurs	Max occurs
monitor-instances	<code>push:boolean</code>	Specifies if individual publisher statistics are enabled.	0	1

server-statistics

The following table lists the attributes that an element of type `server-statistics` can have:

Name	Type	Description	Required
enabled	<code>push:boolean</code>	Specifies whether to enable server statistics. This enables high-level aggregate statistics for the system.	true

reporters

The set of `StatisticsReporters` that the Diffusion server is aware of at startup. Used to output the statistics gathered for clients, publishers, or topics.

The following table lists the elements that an element of type `reporters` can contain:

Name	Type	Description	Min occurs	Max occurs
reporter	<code>reporter</code>	A reporter definition.	0	unbounded

reporter

A `StatisticsReporter` definition.

The following table lists the attributes that an element of type `reporter` can have:

Name	Type	Description	Required
name	<code>push:string</code>	The reporter name.	true

Name	Type	Description	Required
enabled	push:boolean	Whether the reporter is enabled. If this is set to true, the reporter is automatically loaded when Diffusion starts. Otherwise, you must manually load the reporter config at run-time using the statistics API.	true

The following table lists the elements that an element of type `reporter` can contain:

Name	Type	Description	Min occurs	Max occurs
type	push:string	The type of reporter to be used. Currently options are: TOPIC - exposes metrics in the Diffusion topic tree; JMX - exposes metrics on the local JMX server.	1	1
property	property	A property available to the reporter. This can be used to configure reporter-specific variables or parameters.	0	unbounded

property

A StatisticsReporter property. Currently accepted values: 'interval' - used by the topic reporter. Specifies an integer value, in seconds, used to set the period of update publishing.

The following table lists the attributes that an element of type `property` can have:

Name	Type	Description	Required
name	push:string	The property value	true
type	push:string	An optional property type. Usage of this is implementation specific.	false

SubscriptionValidationPolicy.xml

This file specifies the schema for the subscription validation policy.

subscription-validation-policies

Subscription validation policies

The following table lists the elements that an element of type `subscription-validation-policies` can contain:

Name	Type	Description	Min occurs	Max occurs
topics	topics	A map of topics to policies.	0	1
policy	policy	A subscription validation policy.	0	unbounded

topics

A map of topics to policies.

The following table lists the elements that an element of type `topics` can contain:

Name	Type	Description	Min occurs	Max occurs
topic	topic	A topic to policy mapping.	0	unbounded

topic

The following table lists the attributes that an element of type `topic` can have:

Name	Type	Description	Required
policy	push:string	The name of the policy to apply to this topic.	true

policy

A subscription validation policy.

The following table lists the attributes that an element of type `policy` can have:

Name	Type	Description	Required
name	push:string	Each policy must be supplied with a unique name for easy reference.	true
type	push:string	The policy type is either "blacklist" or "whitelist". A blacklist indicates that if any of the policy rules in this policy match the incoming connection, that connection is to be rejected. A whitelist requires that at least one policy rule matches for the connection to be accepted.	true
automatic	push:boolean	Policies which are set to automatic are applied by Diffusion and the publishers do not need to perform any checks themselves. If this is set to false, the policy is not applied unless it is done by the publisher. If this value is not specified, a default of true is used.	false
validate-children	xsd:boolean	Controls whether to perform validation on child topics if the parent topic fails validation. If a value is not specified, a default of false is used.	false

The following table lists the elements that an element of type `policy` can contain:

Name	Type	Description	Min occurs	Max occurs
addresses	addresses	Subscription validation policy addresses. These are addresses that are blacklisted/whitelisted.	0	1
locale	locale	Connection validation policy locale. This is locale details that are blacklisted/whitelisted.	0	unbounded

addresses

The following table lists the elements that an element of type `addresses` can contain:

Name	Type	Description	Min occurs	Max occurs
address	push:string	An IP address (or regular expression) of a subscribing client.	0	unbounded
hostname	push:string	The hostname (or regular expression) of a subscribing client.	0	unbounded
resolved-name	push:string	The resolved hostname (or regular expression) of a subscribing client, as returned by the Whois service.	0	unbounded

locale

The following table lists the elements that an element of type `locale` can contain:

Name	Type	Description	Min occurs	Max occurs
country	push:string	The ISO country code of the subscribing client, as returned by the Whois service.	0	1
language	push:string	The ISO language code of the subscribing client, as returned by the Whois service.	0	1

Additional XML files

The `etc` directory contains additional XML files. The format of these XML files is not defined by Push Technology.

The following XML files are included in the `etc` directory. For more information, see [Cross domain policies](#) on page 656.

```
crossdomain.xml
FlashMasterPolicy.xml
FlashPolicy.xml
```

Use these files to grant a web client permission to handle data across multiple domains.

Starting the Diffusion server

After you have installed and configured your Diffusion server, you can start it using one of a number of methods.

Use the provided Diffusion start scripts

Your Diffusion installation includes The `diffusion.sh` or `diffusion.bat` command (issued in the `bin` directory) starts Diffusion. An optional properties directory can be specified as a parameter to be used instead of the default `../etc` directory.

Important: Do not run your Diffusion server as root on Linux or UNIX. To run the Diffusion server on a port number of 1024 or lower, use another means. For some examples of ways of doing this, see <http://www.debian-administration.org/articles/386>.

Use a script in `init.d`

On Linux, Diffusion can be started using a script in your `/etc/init.d` folder that starts your Diffusion server when the host server starts.

If you installed your Diffusion server using RPM, this script already exists in your `/etc/init.d` folder.

If you installed your Diffusion using another method, you can use the sample script files in the `tools/init.d` directory of your Diffusion. Edit the sample script file to include the location of your installation and make any other changes that are required. Copy the edited script file to `/etc/init.d`. Ensure that the file is executable.

When your host server starts, it starts your Diffusion server.

Use Docker

Diffusion is provided as a Docker image on DockerHub. When you use Docker to run this image, the Diffusion server inside the image is started.

For more information, see [Installing the Diffusion server using Docker](#) on page 541.

Run embedded in a Java process

You can run the Diffusion server from within a Java process by including the `diffusion.jar` on the classpath of the Java process.

For more information, see [Running from within a Java application](#) on page 637.

Running from within a Java application

To run Diffusion from within a Java application instantiate, configure and start a `DiffusionServer` object.

Creating a server

`DiffusionServer` is available in the `com.pushtechnology.diffusion.api.server`. You can instantiate it with one of the following constructors:

Default configuration

```
DiffusionServer server = new DiffusionServer();
```

This instantiates the server with default configuration options. The default configuration is read from the XML configuration files in the `etc` directory of your Diffusion installation. Required aspects of the server must be configured before it is started. These can be configured programmatically. For more information, see [Programmatic configuration](#) on page 553.

Bootstrap properties

```
DiffusionServer server = new  
DiffusionServer(bootstrapProperties);
```

This specifies a set of properties inside a `Properties` object. The following properties are supported:

Property	Description	Default
diffusion.home	The base installation directory	Calculated from the location of the <code>diffusion.jar</code> file.
diffusion.config.dir	The configuration directory, where the XML configuration files are located.	<code>diffusion.home/etc</code>
diffusion.license.file	The license file	<code>diffusion.config.dir/licence.lic</code>
diffusion.keystore.file	The keystore file required to decrypt the license	<code>diffusion.config.dir/licence.keystore</code>

These properties can also be set as system properties.

Whichever approach to instantiation that you use, a full set of XML configuration files can be present in the configuration directory and tuned as required or just a partial set of the files can be present and all missing configuration supplied programmatically.

Configuring the Diffusion server

Once the server object has been instantiated some properties can be configured. The root configuration object can be obtained from the server object as follows:

```
ServerConfig config = server.getConfig();
```

Alternatively the root can be obtained using `ConfigManager.getServerConfig()`.

The configuration is populated from the XML configuration files in the installation or in the configuration directory specified by `diffusion.config.dir`. You can further customize the configuration to fit your requirements. For an example, see the following sample code:

```
DiffusionServer server = new DiffusionServer();

ServerConfig config = server.getConfig();

// Publisher
PublisherConfig publisher = config.addPublisher("My
  Publisher", "com.company.MyPublisherClass");

// Connector ConnectorConfig connector = config.addConnector("Client
  Connector");
// Configure connector as required....

// Thread Pools
ThreadsConfig threads = config.getThreads();
ThreadPoolConfig inbound = threads.addPool("Inbound");
inbound.setCoreSize(3);
inbound.setMaximumSize(10);
inbound.setQueueSize(2000);
threads.setInboundPool(inbound.getName());
threads.setBackgroundPoolSize(2);

// Queues QueuesConfig queues = config.getQueues();
QueueConfig queue = queues.addQueue("DefaultQueue");
queue.setMaximumDepth(10000);
queues.setDefaultQueue("DefaultQueue");
```

```
// Multiplexer MultiplexerConfig multiplexer =
config.addMultiplexer("Multiplexer");
multiplexer.setSize(4);
```

Note: The logging configuration cannot be changed using the `ServerConfig` object. By the time the configuration objects are available to your Java application, the logging properties are locked. The `LoggingConfig` object is read-only.

Monitoring the Diffusion server lifecycle

The `DiffusionServer` class provides methods to add and remove a lifecycle listener on the server instance.

```
addLifecycleListener(LifecycleListener stateCallback);
```

```
removeLifecycleListener(LifecycleListener stateCallback);
```

The lifecycle listener is a callback that is called whenever the server state changes. The Diffusion server can have the following states:

INITIAL

An instance of the `DiffusionServer` exists, but has not been started.

STARTING

The server is starting.

STARTED

The server has started and all publishers are deployed.

STOPPING

The server is stopping.

STOPPED

The server has stopped.

Starting the server

After the server configuration has been completed, the server can be started using `server.start()`.

The declared publishers are then loaded and connectors start to listen on the configured ports.

Stopping the server

The server can be stopped using `server.stop()` at which point the server is no longer available.

Run requirements

A simple way to use Diffusion as a library within your application is to install Diffusion and include the path to `diffusion.jar` in your CLASSPATH.

If you want to repackage the Diffusion code more extensively, be aware of the following concerns:

- `diffusion.jar` depends on other library files in the installation's `lib` directory and are referenced in the jar's manifest `Class-Path` entry. You must also make the code in these libraries available.

- You still require a Diffusion installation. The installation provides the configuration, licence, and log directories. The installation directory is calculated from the location of `diffusion.jar`. If `diffusion.jar` is not loaded from a URL classloader, or has been moved from the product installation, use the bootstrap properties constructor and set the `diffusion.home` system property to the installation directory.

Limitations

Currently only one Diffusion server can be instantiated in a Java VM and it can be started only once.

Deploying publishers on your Diffusion server

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

All publishers that run on the Diffusion server must be defined in the `Publishers.xml` configuration file located in the `etc` directory

If a publisher is defined in the Diffusion server configuration, you can deploy it on the Diffusion server by using either *classic deployment* or *hot deployment*. Classic deployment is deploying a publisher to a stopped Diffusion server. The publisher then starts when the Diffusion server starts. Hot deployment is deploying a publisher to a running Diffusion server.

Classic deployment

Installing publishers into a stopped Diffusion instance.

The publishers that are started when Diffusion starts must be defined in the configuration file `etc/Publishers.xml`. Publishers that do not start with Diffusion can also be defined in the `etc/Publishers.xml` and these can be started later using JMX.

The publishers must be present on the classpath of the Diffusion server. The recommended way to do this is to compile the publisher source code with the `diffusion.jar` they run with on the classpath. Package the publisher class files into a JAR file. This JAR file must be deployed to the `ext/` directory of the Diffusion installation. The Diffusion server will search the `ext/` directory and load all the JAR files it finds.

Related concepts

[Writing a publisher](#) on page 492

How to approach writing a publisher

[Build server application code with Maven](#) on page 518

The Diffusion API for server application code is not available in the Push Technology public Maven repository. To build server components, you must install the product locally and depend on `diffusion.jar` using a Maven system scope.

[Build publishers with Maven](#) on page 514

The Diffusion API for publishers is not available in the Push Technology public Maven repository. To build publishers, you must install the product locally and depend on `diffusion.jar` using a Maven system scope.

Related tasks

[Building a publisher with mvndar](#) on page 516

Use the Maven plugin *mvndar* to build and deploy your publisher DAR file. This plugin is available from the Push Public Maven Repository.

Hot deployment

Installing publishers into a running Diffusion instance.

In addition to starting publishers by defining them in `etc/Publishers.xml`, you can install them into an already running Diffusion instance by a process known as hot deployment. Publishers can also be undeployed and redeployed, providing they implement the `isStoppable` method, and it returns true. You can also deploy dependent JAR files, configuration files and associated web pages for a publisher. All artifacts required for deployment are packaged within a DAR file.

Related concepts

[Build server application code with Maven](#) on page 518

The Diffusion API for server application code is not available in the Push Technology public Maven repository. To build server components, you must install the product locally and depend on `diffusion.jar` using a Maven system scope.

[Build publishers with Maven](#) on page 514

The Diffusion API for publishers is not available in the Push Technology public Maven repository. To build publishers, you must install the product locally and depend on `diffusion.jar` using a Maven system scope.

Related tasks

[Building a publisher with mvndar](#) on page 516

Use the Maven plugin *mvndar* to build and deploy your publisher DAR file. This plugin is available from the Push Public Maven Repository.

Deployment methods

There are two ways to deploy a DAR file: file copy or HTTP.

File copy

To use this method, copy your DAR file to the deployment directory on the file system. If you enable auto-deployment in the `Server.xml` configuration file, Diffusion periodically scans a directory for new or updated DAR files and deploys them. In the case of an updated DAR, the existing publisher is undeployed (if possible) before being redeployed.

HTTP

If the deploy web service is running, you can `POST` the DAR file over HTTP. For example, you can use command line tools such as `curl` to deploy the publisher:

```
curl --data-binary @MyPublisher.dar http://localhost:8080/deploy
```

Warning: We recommend you use the HTTP method of deployment in your test environments only. If you enable the deploy web service in your production environment, you must take additional security measures to prevent unauthorized or malicious access to the web service URL. For example, by setting up restrictions in your firewall.

To enable deployment through HTTP, you must enable the web service in the `WebServer.xml` configuration file. For example, include the following XML:

```
<http-service name="deploy-service" debug="true">
  <class>com.pushtechnology.diffusion.service.DeploymentService</class>
  <url-pattern>^/deploy.*</url-pattern>
  <max-inbound-request-size>128m</max-inbound-request-size>
</http-service>
```

Ensure that the HTTP connector is configured to have an input buffer large enough to contain the entire DAR file. You can configure this in the `Connectors.xml` configuration file.

Undeployment

For publishers deployed using the file copy method, you can delete the DAR file from the deployment directory and on the next scan the server undeploys the publisher. A DAR file can be undeployed only if all of the publishers it contains are stoppable. If a DAR file fails to be undeployed, any future modifications to it are ignored.

It is important that any files that the deployment process has extracted from the DAR are not deleted until the publisher has been successfully undeployed. Publishers can also be undeployed through JMX by invoking the undeploy operation on associated MBean, for example

```
localhost/Server/com.pushtechnology.diffusion - Publisher -
MyPublisher - undeploy()
```

Related concepts

[Build server application code with Maven](#) on page 518

The Diffusion API for server application code is not available in the Push Technology public Maven repository. To build server components, you must install the product locally and depend on `diffusion.jar` using a Maven system scope.

[Build publishers with Maven](#) on page 514

The Diffusion API for publishers is not available in the Push Technology public Maven repository. To build publishers, you must install the product locally and depend on `diffusion.jar` using a Maven system scope.

Related tasks

[Building a publisher with mvndar](#) on page 516

Use the Maven plugin `mvndar` to build and deploy your publisher DAR file. This plugin is available from the Push Public Maven Repository.

Load balancers

Load balancers provide many capabilities that are key to creating a seamless Diffusion solution. We recommend that you use a load balancer with Diffusion.

In addition to balancing client connections across multiple Diffusion servers, you can use load balancers to composite URL spaces or do SSL offloading.

Connections between Diffusion clients and Diffusion servers have specific requirements. If your load balancer handles Diffusion connections incorrectly, this can cause problems for your solution.

Ensure that you fully review this section of the user guide when using load balancers with Diffusion.

Routing strategies at your load balancer

Your load balancer can present a number of different strategies for choosing which Diffusion server a new client connection is routed to. After a client connection has been routed to a Diffusion server, ensure that the client is always routed to the Diffusion server that its session exists on.

The routing strategies that are available to you depend on the load balancer that you choose to use. The following table lists some examples of routing strategies:

Table 63: Examples of routing strategies

Name	Description
Round-robin	Each available Diffusion server is chosen in turn, with none favored.
Fewest clients	The Diffusion server with the fewest number of client connections in progress is chosen.
Least loaded	The Diffusion server with the lowest CPU load is chosen.

Routing connection that use HTTP protocols

To route HTTP traffic, the load balancer must be able to inspect the HTTP headers and extract session information.

Diffusion sets an HTTP cookie named `session` with a connection-specific ID specifically for this purpose. Your load balancer can use this cookie to maintain a table of client to server mappings.

The session cookie is flagged with `HttpOnly`, which prevents scripts accessing the cookie. The session cookie is not flagged with `Secure`, which prevents the cookie from being sent over non-secure connections.

Sample HTTP conversation, cookie highlighted:

```
POST /diffusion/ HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Content-Length: 0
tt: 90
Origin: http://localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
  (KHTML, like Gecko) Ubuntu Chromium/48.0.2564.82 Chrome/48.0.2564.82
  Safari/537.36
m: 0
ty: B
v: 4
Content-Type: text/plain;charset=UTF-8
Accept: */*
Referer: http://localhost:8080/tools/DhtmlClient.html
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6

HTTP/1.1 200 OK
Set-Cookie: session=c04815df73a1646d-0000000000000000; HttpOnly
Access-Control-Allow-Origin:http://localhost:8080
Cache-Control:no-store, no-cache
Content-Type:text/plain; charset=UTF-8
Content-Length:41

4.100.4.c04815df73a1646d-0000000000000000
```

```

POST /diffusion/ HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Content-Length: 0
Origin: http://localhost:8080
c: c04815df73a1646d-0000000000000000
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
  (KHTML, like Gecko) Ubuntu Chromium/48.0.2564.82 Chrome/48.0.2564.82
  Safari/537.36
m: 1
Content-Type: text/plain;charset=UTF-8
Accept: */*
Referer: http://localhost:8080/tools/DhtmlClient.html
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Cookie: session=c04815df73a1646d-0000000000000000

```

Diffusion uses a session ID for the cookie. This enables the load balancer to maintain a map of each session ID to its target Diffusion server.

Instead, you can disable the Diffusion cookie and configure the load balancer to set a cookie that identifies the target server instead of the session. While the overhead of transmitting a cookie is still present between client and load balancer, the identifier can be smaller because there are a smaller number of servers than client sessions. Load balancers that use the cookie to identify the Diffusion server can send less data down the wire and consume fewer resources.

If you want to configure your load balancer to inject its own cookie, you can disable this Diffusion cookie. To disable the Diffusion cookie, set the `<disable-cookie>` element in the `WebServer.xml` configuration file of your Diffusion to `true`.

Routing connections that use streaming protocols

Streaming protocols that open a single socket and remain connected until they are no longer required appear immune to requiring any special routing considerations. However, in the event that connection keep-alive is enabled to handle reconnections in case of temporary connection loss, it is important that the reconnection attempt is routed to the original server.

Without the ability to parse headers (and indeed, the absence of a session ID at all), the most common method for routing a streaming protocol such as WebSocket is to create a client/server mapping based on the IP addresses of the endpoints. This technique is generally referred to as Sticky-IP, and has the advantage of also working with HTTP transports, if required.

For F5's Sticky IP, ensure that the Source Address Translation option is set to Auto Map.



Figure 40: Sticky-IP in F5 BIG-IP

The drawback of this approach is that multiple users masquerading behind a proxy or access point can have the same IP address, and all requests from clients with that IP address are routed to the same Diffusion instance. Load balancing still occurs, but some hosts might be unfairly loaded.

Monitoring available Diffusion servers from your load balancer

To route your client connections most effectively, your load balancer must know which Diffusion servers are available to accept connections.

There are a number of ways to determine the availability of a Diffusion server:

- Implement a custom monitor using a scripting language that is supported by your load balancer.

For example, create a custom Diffusion client that connects and subscribes to a status topic.

This is the most effective way of determine availability as can check the connector used by your client applications.

- Use a TCP probe against the client connector.

A TCP probe checks whether the Diffusion server is able to accept new connections without doing any processing.

If your load balancer's TCP probe does not complete the full 3-way handshake and instead responds with a TCP reset after the server acks the probe, you might see IO exceptions on the the Diffusion server. You can use the `ignore-error-from` element in the `Connectors.xml` configuration file to tell Diffusion to ignore these errors.

- Use an HTTP probe against the built-in web server.

This has the advantage of being simple; most system administrators are familiar with HTTP requests. In the simplest case, a `GET` request can be made against the root context of the web server, for example:

```
GET / HTTP/1.0\r\n
```

However, this only tests the availability of the Diffusion server as a whole, and not the applications within it.

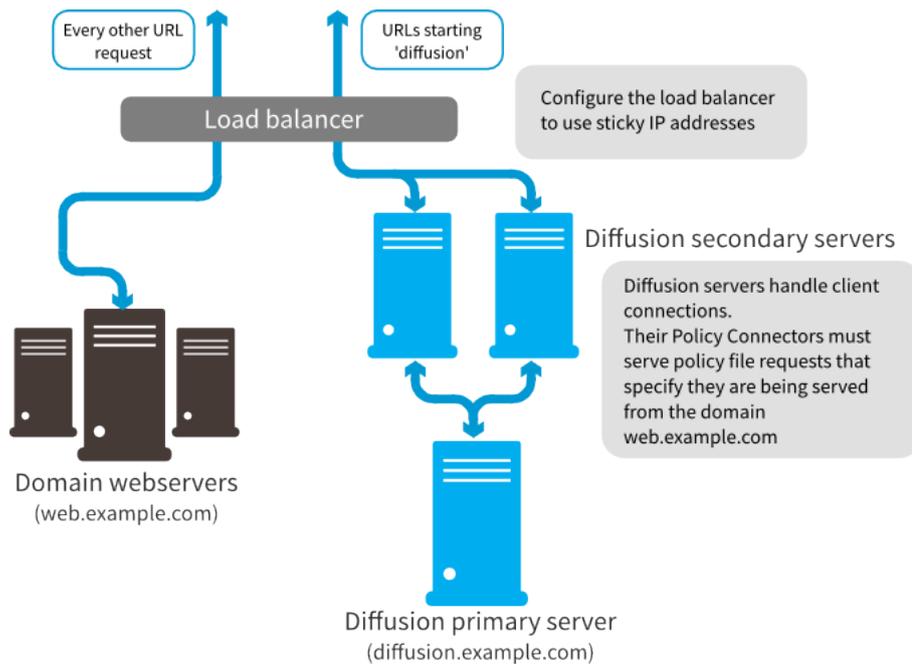
- Implement an `http-service` on your Diffusion servers.

This service can query the state of the topic tree or any publishers, and return availability on receipt of a `GET` request.

Compositing URL spaces using your load balancer

If your Diffusion servers are located at a different URL to the Diffusion browser clients hosted by your web servers, you can use a load balancer to composite the URL spaces.

Security features in some browsers prevent web-based Diffusion clients from making requests to your Diffusion server if your Diffusion server is in a different URL space to the web server you use to host your client.



- Your web content is hosted on `web.example.com` and your Diffusion servers are hosted on `diffusion.example.com`.
- The Diffusion servers are configured to serve policy files that specify they are being served from the domain `web.example.com`

For more information about serving policy files, see [Configuring connectors](#) on page 581.

- The load balancer composites the URL space so that requests to Diffusion at `web.example.com` are routed to Diffusion servers hosted on `diffusion.example.com`
- To the client, both the web and the Diffusion content appear to be hosted on `web.example.com`. This avoids any cross-domain security issues.

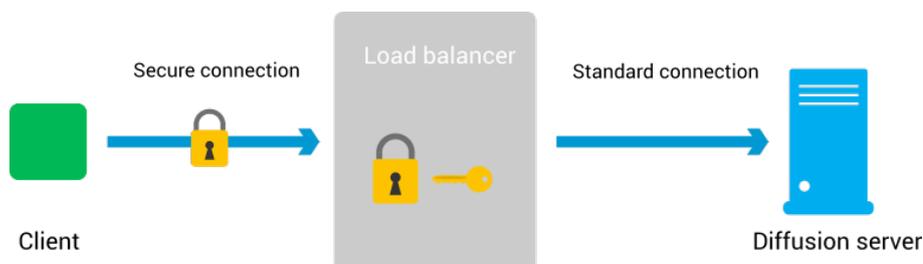
If you choose not to use your load balancer to composite the URL spaces, you can set up cross-domain policy files that allow requests to the different URL spaces.

For more information about serving policy files, see [Configuring connectors](#) on page 581.

Secure Sockets Layer (SSL) offloading at your load balancer

Diffusion clients can connect to your solution using TLS or SSL. The TLS/SSL can terminate at your load balancer or at your Diffusion server.

Note: If you have sensitive client data that you must secure, use a secure connection all the way from the client to the Diffusion server. Either do not perform SSL offloading at the load balancer or re-encrypt the connection between load balancer and the Diffusion server.



SSL offloading is when you terminate the TLS at the load balancer. The processing burden of encrypting and/or decrypting a Diffusion client connection made over SSL can then be offloaded to a component that can perform SSL termination more efficiently.

After the SSL connection has been decrypted, the client connection can travel between the load balancer and the Diffusion server using an unsecured transport. Doing this reduces CPU cost on your Diffusion servers.

Using load balancers for resilience

An important part of creating your Diffusion solution is ensuring that it is resilient if one of its components fails.

Load balancer redundancy

If you only have one load balancer in your Diffusion solution, this load balancer can become a single point of failure. For a more resilient solution, have more than one load balancer.

You can have a cluster of active load balancers, each with a different IP address, behind a DNS that uses a round-robin strategy to direct client connections to a hostname to the IP of each load balancer in turn. With a round-robin routing strategy at your DNS, there can be a lag between a load balancer becoming unavailable and this being detected.

Alternatively, you can configure your network so that all of your load balancers are available on the same IP address.

If you use multiple load balancers, ensure that all load balancers have access to any client/server mapping information.

Refer to the documentation for your load balancer for further information about load balancer redundancy.

Using load balancers with Diffusion replication

Diffusion replication is designed to be used with Diffusion servers that are load balanced. If a Diffusion server in your solution becomes unavailable, your load balancer must re-route any of that server's client connections to other Diffusion servers in your solution.

For more information, see [High availability](#) on page 104.

Common issues when using a load balancer

There are some configuration options on your load balancer that can cause problems or inefficient behavior in your Diffusion solution.

Connection pooling

Many load balancers include a connection pooling feature where connections between the load balancer and the Diffusion server are kept alive and reused by other clients. In fact, multiple clients can be multiplexed through a single server-side connection.

In Diffusion, a client is associated with a single TCP/HTTP connection for the lifetime of that connection. If a Diffusion server closes a client, the connection is also closed. Diffusion makes no distinction between a single client connection and a multiplexed connection, so when a client sharing a multiplexed connection closes, the connection between the load balancer and Diffusion is closed, and subsequently all of the client-side connections multiplexed through that server-side connection are closed.

For this reason, it is required that load balancers are not configured to pool connections when working with Diffusion.

Reuse TCP connection

If your load balancer is configured to create a new TCP connection between the load balancer and the Diffusion server for each request from a specific client, this can be expensive. Creating a new TCP connection per request, increases the time each request takes to be processed and increases the amount of traffic between the load balancer and the Diffusion server.

To avoid this, ensure that your load balancer is configured to reuse a TCP connection for requests from the same client.

Sticky-IP

We recommend that you use the sticky-by-IP routing strategy when your clients connect using streaming protocols. This ensures that client connections are always routed to the Diffusion server where their sessions are located.

However, the drawback of this approach is that multiple users masquerading behind a proxy or access point can have the same IP address, and all requests from clients with that IP address are routed to the same Diffusion server. Load balancing still occurs, but some hosts might be unfairly loaded.

TCP retransmission timeout

If you use Diffusion failover, the TCP retransmission timeout on your load balancer's host server can cause long waits for clients whose connections failover from one Diffusion server to another. When a Diffusion server becomes unavailable, the load balancer can hold open existing client connections to this server. These connections can continue to receive and buffer data from the client for the duration of the timeout, before being closed. This data is discarded when the connection closes.

You can avoid this problem by changing the TCP retransmission timeout of the host server of your load balancer or by configuring the load balancer to shutdown connections to Diffusion servers it knows are unhealthy.

Web servers

Diffusion incorporates its own basic web server for a limited set of uses. The Diffusion server also interacts with third-party web servers that host Diffusion web clients. The Diffusion server is also capable of being run as a Java servlet inside a web application server.

Related concepts

[Diffusion web server](#) on page 649

Diffusion incorporates its own web server. This web server is required to enable a number of Diffusion capabilities, but we recommend that you do not use it to host your production web applications.

[Web servers](#) on page 120

Consider how to use web servers as part of your Diffusion solution.

[Running the Diffusion server inside of a third-party web application server](#) on page 652

Diffusion can run as a Java servlet inside any Java application server.

[Hosting Diffusion web clients in a third-party web server](#) on page 651

Host Diffusion web clients — clients written using the JavaScript, Flash, or Silverlight APIs — on a third-party web server to enable your customers to access them.

[Configuring the Diffusion web server](#) on page 616

Use the `WebServer.xml` and `Aliases.xml` configuration files to configure the behavior of the Diffusion web server.

[Configuring Diffusion web server security](#) on page 617

When configuring your Diffusion web server, consider the security of your solution.

Related reference

[WebServer.xml](#) on page 617

This file specifies the schema for the web server properties.

Diffusion web server

Diffusion incorporates its own web server. This web server is required to enable a number of Diffusion capabilities, but we recommend that you do not use it to host your production web applications.

Any Diffusion connector can be configured to act as a web server and provide the following capabilities:

- Providing an endpoint for the HTTP-based transports used by Diffusion clients
- Hosting the Diffusion server landing page
- Hosting the Diffusion demos
- Hosting the Diffusion monitoring console
- Serving policy files
- Optionally, hosting a static page you can use to check the status of the Diffusion server

For more information about configuring the Diffusion web server for these uses, see [Configuring the Diffusion web server](#) on page 616.

Related concepts

[Web servers](#) on page 648

Diffusion incorporates its own basic web server for a limited set of uses. The Diffusion server also interacts with third-party web servers that host Diffusion web clients. The Diffusion server is also capable of being run as a Java servlet inside a web application server.

[Web servers](#) on page 120

Consider how to use web servers as part of your Diffusion solution.

[Running the Diffusion server inside of a third-party web application server](#) on page 652

Diffusion can run as a Java servlet inside any Java application server.

[Hosting Diffusion web clients in a third-party web server](#) on page 651

Host Diffusion web clients — clients written using the JavaScript, Flash, or Silverlight APIs — on a third-party web server to enable your customers to access them.

[Configuring the Diffusion web server](#) on page 616

Use the `WebServer.xml` and `Aliases.xml` configuration files to configure the behavior of the Diffusion web server.

[Configuring Diffusion web server security](#) on page 617

When configuring your Diffusion web server, consider the security of your solution.

Related reference

[WebServer.xml](#) on page 617

This file specifies the schema for the web server properties.

Server-side processing

A basic level of server-side processing can be utilized with any file hosted on the Diffusion web server that has a text mime type and JavaScript.

There are three server-side tags: Include, Publisher and Topic Data. These tags are stored in HTML comments so as to not interfere with normal HTML

Include Tag

Include stubs load the file specified in the file attribute and are loaded as is into the parent HTML document. They do not necessarily have to be valid HTML. They can be positioned anywhere within the HTML file.

These includes are synonymous with #Include statements of ANSI C.

Below is an example of the syntax:

```
<!--@DiffusionTag type="Include" file="stub.html" -->
```

Include files can be nested so an include file can contain an include tag

Publisher tag

Publisher tags enable a publisher to interact with the web page during the serving process. Again these tags can appear anywhere within the HTML document. In the case below the publisher method `processHTMLTag` of the `Trade` publisher is called with the tag argument of `table`. The publisher can return a String of HTML that is inserted into the document at the position of the tag and the tag is removed. The `processHTMLTag` method is also called with the HTTP Request, although the request cannot be written to. Below is an example of the syntax

```
<!--@DiffusionTag type="publisher" publisher="Trade" tagid="table" -->
```

TopicData

Topic data tags allow for `SingleValueTopicData` items to be rendered in the HTML page. Again these tags can appear anywhere within the HTML document. The following example shows the syntax:

```
<!--@DiffusionTag type="TopicData" name="Assets/FX/EURUSD/0" -->
```

HTTP listener

Publishers can listen to all file HTTP requests by registering as a `HTTPRequestListener`. This exposes the interface

```
void handleHTTPRequest(HTTPVirtualHost virtualHost, HTTPRequest request)
```

This enables for more detailed statistics to be captured from the HTTP request

Hosting a status page on the Diffusion web server

You can host a simple status page on the Diffusion.

When setting up your Diffusion server to act as a web server for a status page, ensure that the web service uses a different connector to the Diffusion clients. This enables the web server to use a different thread pool and ensures that requests for status are not slowed by heavy client traffic.

You can use server-side tags to include topic data or call on publisher methods from within the status page. For more information, see [Server-side processing](#) on page 650.

Receiving no response from the status page might not indicate that the server hosting it is down. If you use a non-response from the status page as an indicator for failing over to another Diffusion server, ensure that you kill all processes belonging to the non-responsive Diffusion server before failing over.

Hosting Diffusion web clients in a third-party web server

Host Diffusion web clients — clients written using the JavaScript, Flash, or Silverlight APIs — on a third-party web server to enable your customers to access them.

If your Diffusion clients are web clients, they must be hosted on a web server to enable your customers to access them. We recommend that you use a third-party web server to host your clients instead of the built-in web server provided by Diffusion.

This approach requires additional configuration of your solution to account for cross-origin requests.

Cross-origin requests

Cross-origin requests occur when your web client requests resources (for example, data from the Diffusion server) that are hosted on a different domain, or in some cases a different port on the same domain, to your web client.

Some browsers do not support cross-origin resource sharing. For more information, see [Cross-origin resource sharing limitations](#) on page 57.

You can use one of the following approaches to enable cross-origin requests for your solution:

- Define a cross domain policy. For more information, see [Cross domain policies](#) on page 656.
- Use a load balancer to composite the URL spaces.

Related concepts

[Web servers](#) on page 648

Diffusion incorporates its own basic web server for a limited set of uses. The Diffusion server also interacts with third-party web servers that host Diffusion web clients. The Diffusion server is also capable of being run as a Java servlet inside a web application server.

[Diffusion web server](#) on page 649

Diffusion incorporates its own web server. This web server is required to enable a number of Diffusion capabilities, but we recommend that you do not use it to host your production web applications.

[Running the Diffusion server inside of a third-party web application server](#) on page 652

Diffusion can run as a Java servlet inside any Java application server.

[Configuring the Diffusion web server](#) on page 616

Use the `WebServer.xml` and `Aliases.xml` configuration files to configure the behavior of the Diffusion web server.

[Web servers](#) on page 120

Consider how to use web servers as part of your Diffusion solution.

Running the Diffusion server inside of a third-party web application server

Diffusion can run as a Java servlet inside any Java application server.

When running the Diffusion server inside a third-party web application server, the Diffusion server can have a different port number to clients that are hosted on the same server. This can cause cross-origin .

Some browsers do not support cross-origin resource sharing. For more information, see [Cross-origin resource sharing limitations](#) on page 57.

You can use one of the following approaches to enable cross-origin requests for your solution:

- Define a cross domain policy. For more information, see [Cross domain policies](#) on page 656.
- Use a load balancer to composite the URL spaces.

When using a third-party web server at least some of the functionality of the built-in Diffusion web server can be disabled. The file-service, and two http-service entries can be removed as Tomcat™ provides this functionality. The client-service is needed to support WebSocket, HTTPC and HTTP connection protocols. If these are not used, the client-service can be disabled as well.

Related concepts

[Web servers](#) on page 648

Diffusion incorporates its own basic web server for a limited set of uses. The Diffusion server also interacts with third-party web servers that host Diffusion web clients. The Diffusion server is also capable of being run as a Java servlet inside a web application server.

[Diffusion web server](#) on page 649

Diffusion incorporates its own web server. This web server is required to enable a number of Diffusion capabilities, but we recommend that you do not use it to host your production web applications.

[Hosting Diffusion web clients in a third-party web server](#) on page 651

Host Diffusion web clients — clients written using the JavaScript, Flash, or Silverlight APIs — on a third-party web server to enable your customers to access them.

[Configuring the Diffusion web server](#) on page 616

Use the `WebServer.xml` and `Aliases.xml` configuration files to configure the behavior of the Diffusion web server.

[Web servers](#) on page 120

Consider how to use web servers as part of your Diffusion solution.

Example: Deploying the Diffusion server within Tomcat

Run the Diffusion server inside Tomcat as a Java servlet.

About this task

The Tomcat servlet container and the Diffusion server run in the same Java process and can communicate directly through shared memory. Tomcat and the Diffusion server listen on different ports. Clients can connect directly to the Diffusion server without going through Tomcat.

Procedure

1. Configure an installation of the Diffusion server for how you want your Diffusion servlet to behave.
Ensure, when editing the configuration files in the `etc` directory, that all paths are expressed as absolute paths.
Ensure that a valid license file is present in the `etc` directory.
Place any additional JARs that are required by your servlet in the `ext` directory of your Diffusion installation.
2. Use the `war.xml` Ant script in the `tools` directory of your Diffusion to package the Diffusion server as a WAR file.

```
ant -f war.xml
```

The script creates the `diffusion.war` file in the `build` directory of your Diffusion installation.

The `diffusion.war` file includes the following files and directories:

META-INF/manifest.xml

The manifest file for the WAR

WEB-INF/web.xml

This file contains information about the servlet.

WEB-INF/classes

This directory contains the configuration files for the Diffusion server. These files are copied from the `etc` directory of the Diffusion installation.

WEB-INF/lib/diffusion.jar

The `diffusion.jar` file contains the Diffusion server

WEB-INF/lib

This directory also contains JAR files copied from the `ext` directory of the Diffusion installation.

WEB-INF/lib/thirdparty

This directory contains the third-party libraries that are required by the Diffusion server. These files are copied from the `lib/thirdparty` directory of the Diffusion installation.

lib/DIFFUSION

This directory contains the browser API libraries. These files are copied from the `html/lib/DIFFUSION` directory of the Diffusion installation.

Additional files and directories

The WAR file contains additional files and directories that are not listed here.

The top level of the WAR file contains resources that can be served by Tomcat.

3. Verify the WAR file.
 - a) Check that the WAR structure is the same as described in the previous step and that all necessary files have been copied into the WAR structure.
 - b) Check that the `WEB-INF/web.xml` file contains the following information.

```
<servlet>
  <servlet-name>Diffusion</servlet-name>
  <display-name>Diffusion Servlet</display-name>
  <servlet-
class>com.pushtechology.diffusion.servlet.DiffusionServlet</
servlet-class>
  <load-on-startup>1</load-on-startup>
```

```
</servlet>
```

The WAR is now ready to be deployed inside a Java web application server. The rest of this task describes how to run the WAR inside of Tomcat, but you can use any Java web application server.

4. Define the Tomcat connectors for incoming connections in the `Server.xml` file.

A connector defines the port, protocol, and various properties of how the connection is handled. The following is an example connector for handling HTTP 1.1 connections on port 8080:

```
<Connector port="8080"
  connectionTimeout="20000"
  URIEncoding="UTF-8"
  maxThreads="3"
  protocol="HTTP/1.1" />
```

See the Tomcat documentation for more information.

5. When starting Tomcat, ensure that the following parameters are set:

- a) Set the `diffusion.home` parameter to the path to the Diffusion JAR file.

```
-Ddiffusion.home=diffusion_installation/lib
```

Tomcat must be aware of Diffusion.

- b) Set the `java.util.prefs.userRoot` parameter to a directory that Tomcat can write to.

For example:

```
-Djava.util.prefs.userRoot=/var/lib/tomcat/diffusion/prefs/user
```

Diffusion uses the `java.util.prefs` mechanism to store preference information. If Tomcat, does not set this parameter, the Diffusion server logs warning messages.

What to do next

Accessing publishers from Tomcat

Diffusion started within Tomcat allows Tomcat to access the publishers. Tomcat can be used to serve JSP files providing dynamically generated content. These files can access publishers using the publishers class static methods.

```
<%@ page import="java.util.List,com.pushtechology.api.publisher.*"
%>
<html>
  <head>
    <title>Publisher Information</title>
  </head>
  <body>
    <table>
      <tr>
        <th>Publisher Name</th>
        <th>Topics</th>
      </tr>
      <% for (Publisher pub : Publishers.getPublishers()) { %>
      <tr>
        <td><%= pub.getPublisherName() %></td>
        <td><%= pub.getNumberOfTopics() %></td>
      </tr>
      <% } %>
    </table>
  </body>
</html>
```

The above is the content of a JSP file that return a list of the publisher Diffusion is running with the number of topics each publisher owns.

Other considerations when running the Diffusion server inside of a third-party web application server

Diffusion can run as a Java servlet inside any Java application server.

Apache Mod Proxy installation

Apache Mod Proxy can be used to forward HTTP requests from an Apache web server to Diffusion. It does not support persistent connections or WebSocket so the WebSocket and HTTPC connections do not work. Make sure that you include the following into the Apache configuration file (Virtual host setting).

```
ProxyPass /diffusion/ http://localhost:8080/diffusion/
```

For more information, see the Apache Mod Proxy documentation.

Apache AJP13 Installation

Apache AJP can be used to forward requests from an Apache web server to Tomcat. In the Apache virtual host configuration, mount the path

```
JkMount /diffusion/*dfnjetty
```

Workers definition file

```
worker.dfnjetty.port=8009
worker.dfnjetty.host=(host IP)
worker.dfnjetty.type=ajp13
worker.dfnjetty.lbfactor=1
worker.dfnjetty.cachesize=50
worker.dfnjetty.socket_keepalive=1

worker.list=dfnjetty
```

A connector that handles the AJP/1.3 protocol is needed running on port 8009 (because of the Workers file described above). See the Tomcat documentation for more information on this.

IIS Installation

Use an ISAPI_Rewrite tool. For example, http://www.helicontech.com/isapi_rewrite

The rewrite rule is as follows:

```
RewriteEngine on
RewriteRule ^diffusion/ http://localhost:8080/diffusion/ [p]
```

Diffusion home directory

The servlet container must be aware of Diffusion. Add the path to the directory that contains the Diffusion JAR file to the Java VM arguments that you use to start the servlet container.

```
-Ddiffusion.home=diffusion_installation/lib
```

Cross domain policies

Cross domain policies grant permission to communicate with servers other than the one the client is hosted on.

Cross-domain XML file

The cross-domain policy is defined in an XML file

A cross-domain policy file is an XML document that grants a web client – such as Adobe® Flash Player, Adobe Reader, Silverlight Player – permission to handle data across multiple domains. When a client hosts content from a particular source domain and that content makes requests directed towards a domain other than its own, the remote domain must host a cross-domain policy file that grants access to the source domain, allowing the client to continue with the transaction. Policy files grant read access to data, permit a client to include custom headers in cross-domain requests, and are also used with sockets to grant permissions for socket-based connections.

For example, say that the Diffusion client is loaded from `static.example.com` and the connection URL to the Diffusion client is `http://streaming.example.com`, a `crossdomain.xml` file must be loaded from `static.example.com`

A `crossdomain.xml` is required if one of the following is true:

- You are using Diffusion as a streaming data server and a separate web server which are on different domains
- The Diffusion connection type is HTTP, HTTPS, HTTPC, or HTTPCS
- You are not using a load balancer to HTTP rewrite Diffusion traffic

Note: You cannot use the `iframe` streaming transport with cross-domain requests. This is not supported by Diffusion.

Installing the `crossdomain.xml` file for Flash/Silverlight HTTP request

- If you use Diffusion as a web server, copy the `crossdomain.xml` file from the Diffusion `install/etc` folder to the root of the `html` folder
- If you do not use Diffusion as a web server, copy the `crossdomain.xml` file from the Diffusion `install/etc` folder to the virtual root of the web server hosting the Diffusion `html lib` folder

By default, Diffusion does not have `crossdomain.xml` installed. We shipped an example which allow all domains and all ports to access the Diffusion server. This must be edited to include the correct security details for your installation.

Flash security model

Flash interacts with remote services to establish security according to the restrictions defined in the Flash policy file.

If a socket-based connection is to be used, for example Diffusion DPT type connection, the Flash player tries to get a policy file from the same host as you are trying to connect to but on port 843. If this port is not open through your firewalls or is not configured within the Diffusion connectors, the Flash player waits 2 seconds before requesting a policy file from the same port that you are trying to connect to. If the policy file request is not responded to correctly or the policy file has restricted the connection, the Flash player generates a security exception and the connection attempt stops.

If an HTTP connection is to be used, for example Diffusion HTTP type connection, a socket-based policy file is not required but a `crossdomain.xml` file might be required before the Diffusion connection is made.

Official Adobe documentation is available at the following location: [Cross-domain policy file specification](#).

Note: Ensure that none of the other services on the same system as your Diffusion server use port 843. If they do, the Flash policy connector is unable to bind to the port and cannot serve the required policy file.

FlashPolicy.xml file

When is the FlashPolicy.xml used?

When a Diffusion DPT connection is used a socket connection is made, in order that the socket connection can be established a socket policy file must be acquired from port 843 or from the port that the Diffusion client is trying to connect to.

Again this is part of the cross-domain schema, but this time the `to-ports` attribute on the `allow-access-from` element is particularly important.

FlashMasterPolicy.xml file

Use of the FlashMasterPolicy file

FlashMasterPolicy is used for requests on port 843. It is a normal `crossdomain.xml` with an extra element of

```
<site-control permitted-cross-domain-policies="master-only" />
```

The `site-control` element here specifies that only this master policy file is considered valid on this domain

Related concepts

[Silverlight security model](#) on page 657

Silverlight interacts with remote services to establish security according to the restrictions defined in the Silverlight policy file.

[JavaScript security model](#) on page 658

When a JavaScript client uses the XHR transport, this imposes security constraints on using cross domain requests.

Silverlight security model

Silverlight interacts with remote services to establish security according to the restrictions defined in the Silverlight policy file.

How Silverlight interacts with remote services to establish security

If a socket-based connection is to be used, for example Diffusion DPT type connection, the Silverlight player tries to get a policy file from the 943. If this port is not open through your firewalls or is not configured within the Diffusion connectors the Silverlight player generates a security exception and the connection attempt ceases.

If an HTTP connection is to be used, for example. Diffusion HTTP type connection, a socket-based policy file is not required but a `crossdomain.xml` file might be required before the Diffusion connection is made.

Official Microsoft documentation

[Network security access restrictions \(Silverlight\)](#)

Silverlight `clientaccesspolicy.xml` file

When is the `clientaccesspolicy.xml` used?

When a Diffusion DPT connection is used a socket connection is made, in order that the socket connection can be established a socket policy file must be acquired from port 943.

Related concepts

[Flash security model](#) on page 656

Flash interacts with remote services to establish security according to the restrictions defined in the Flash policy file.

[JavaScript security model](#) on page 658

When a JavaScript client uses the XHR transport, this imposes security constraints on using cross domain requests.

JavaScript security model

When a JavaScript client uses the XHR transport, this imposes security constraints on using cross domain requests.

The Diffusion JavaScript client library, unless otherwise configured, cascades downward through a set of transports starting with WebSocket and working its way down toward XMLHttpRequest (also known as Ajax Long Poll), and finally to hidden frames.

WebSocket, Flash, and Silverlight have few security constraints, however XHR is subject to the same origin policy. Simply put, if JavaScript code executes within a web page sourced from `www.example.com` it is only permitted to make XHR requests back to `www.example.com`. If your Diffusion server is at `push.example.com` this presents a problem when only XHR is available.

The catch-all solution

The set of web browsers in current use is both broad and heterogeneous. Rather than catering to each special case browser, this approach contains all complexity to one place: the load balancer. This presumes there is a load-balancer however, though in all reasonable production circumstances this is true. All XHR requests to Diffusion use a URL that starts `/diffusion`. Routing all such requests to one of the servers in the Diffusion pool will make available both regular and Diffusion functionality from one apparent host. This approach is suitable to all web browsers.

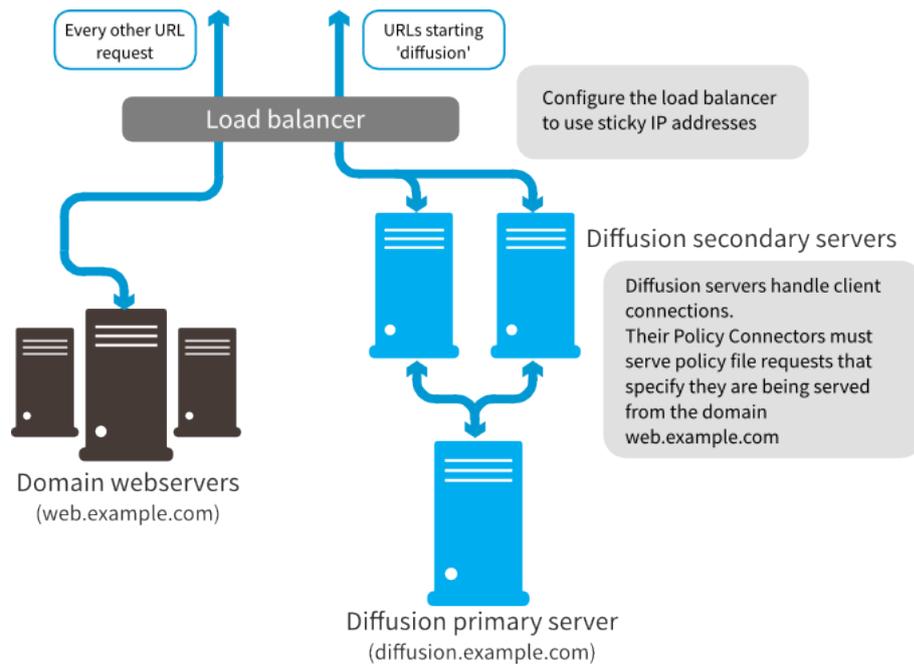


Figure 41: Using a load balancer to composite two URL spaces into one.

In circumstances where clients of Diffusion solutions cannot be depended upon to have a single IP address (for example: users with multiple aDSL connections, or smart-phones migrating between providers), each HTTP request made from a Diffusion client to a Diffusion server holds a cookie named `session` holding the unique session ID of that client. This gives load balancers an alternative means of distributing a request to one of their Diffusion servers.

Software alternatives

For test and development purposes a hardware load balancer might be an expensive means of compositing the URL-spaces of two (or more) web servers into one. Alternatives such as `mod_proxy` for the Apache web server, and an `ISAPI_Rewrite` tool can be employed to achieve the same for a lower (or zero) price-tag. Diffusion can be configured to run as a servlet with Tomcat. For more information, see [Example: Deploying the Diffusion server within Tomcat](#) on page 652.

CORS

CORS is a standard formed to address circumstances where `www.example.com` uses XHR to access resources on alternate host `push.whatnot.com`, and aims to provide sensible constraints and avoid a free-for-all.

CORS uses HTTP headers to enable the Diffusion server to indicate if it accepts traffic from web pages served from other servers. When a CORS request is made, Diffusion must respond with certain response HTTP headers for the browser to treat the request as successful. CORS requests can result in the browser sending a pre-flight request to Diffusion using the `OPTIONS` method to determine if the origin, headers, and methods of the request it is about to make are permitted. Diffusion responds with the correct values for headers and methods but the actual request is not made until the pre-flight request succeeds. The allowed origins can be configured in the `client-service` element of the `WebServer.xml` configuration file. For more information, see [WebServer.xml](#) on page 617.

Client-Side

Include the XHRURL attribute in the arguments to the DiffusionClientConnectionDetails constructor. For example:

```
var connectionDetails = {
    debug : true,
    onDataFunction : onDataEvent,
    XHRURL: "http://www.example.com:8080"
    topic : 'SYMBOLS/QUOTES/NIFTY~INDEX',
}
```

Server side

CORS filtering is governed on the server side using the cors-origin attribute found in `etc/WebServer.xml`. By default this is a very permissive `.*` regular expression, and must be set to something more specific in production. In the above example, `push.example.com` will limit requests to `push.example.com` to only those from `www.example.com`. Full details about this feature are found in the web server section of the Diffusion manual.

Related concepts

[Flash security model](#) on page 656

Flash interacts with remote services to establish security according to the restrictions defined in the Flash policy file.

[Silverlight security model](#) on page 657

Silverlight interacts with remote services to establish security according to the restrictions defined in the Silverlight policy file.

Related reference

[Cross-origin resource sharing limitations](#) on page 57

CORS allows resources to be accessed by a web page from a different domain. Some browsers do not support this capability.

Push Notification Bridge

The Push Notification Bridge is a Diffusion client that subscribes to topics on behalf of other Diffusion clients and uses a push notification network to relay topic updates to the device where the client application is located.

The Push Notification Bridge comprises the following files all located in the `pushnotification` directory of your Diffusion installation:

pn_bridge.jar

This JAR file contains the Diffusion Java client that acts as a bridge between Diffusion and push notification networks.

pn_bridge.bat and pn_bridge.sh

These scripts can be used to start the Push Notification Bridge.

PushNotifications.xml

This XML file is used to configure the Push Notification Bridge.

PushNotifications.xsd

This XSD file defines the schema of the `PushNotifications.xml` file.

How the Push Notification Bridge works

A client sends a JSON message through a request topic to the Push Notification Bridge, requesting push notifications for a specific topic.

The topic that notifications are received for must be a single value topic.

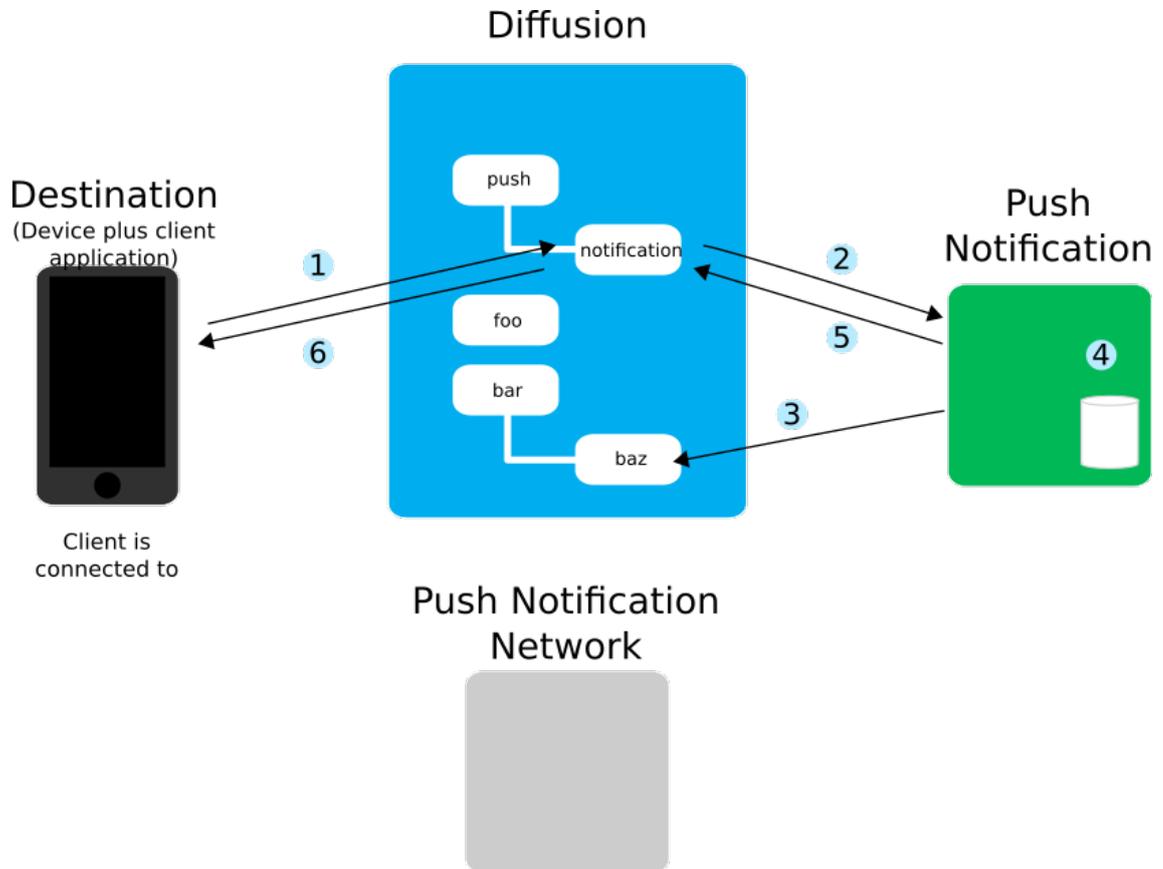


Figure 42: Requests to the Push Notification Bridge

1. The client sends a request message to the service topic path that the bridge listens on.
This topic path is defined in the `PushNotifications.xml` configuration file. For more information, see [Configuring your Push Notification Bridge](#) on page 663.
The request message is in JSON format. For more information about the request message format, see [Request and response JSON formats](#) on page 671.
2. The Push Notification Bridge receives the message through the service topic path.
3. The bridge attempts to subscribe to the topic.
4. If the subscription is successful, the bridge stores the association between the topic and the *push notification destination*. This can be represented by either an APNS device token or a GCM registration ID. The destination is the combination of the client application and the device on which the client is hosted. It is not the same as a client session.
The association between topic and destination is stored in memory, by default. You can persist this information by implementing your own persistence solution. For more information, see [Push Notification Bridge persistence plugin](#) on page 511.
5. The bridge sends a response message to the client through its service topic path.
The response message is in JSON format. For more information about the response message format, see [Request and response JSON formats](#) on page 671.
6. The client receives the response message and can act on it.

The client can also request to be unsubscribed from receiving push notifications for a topic. When an update is received on a subscribed topic, the bridge sends a push notification to the destinations associated with that topic.

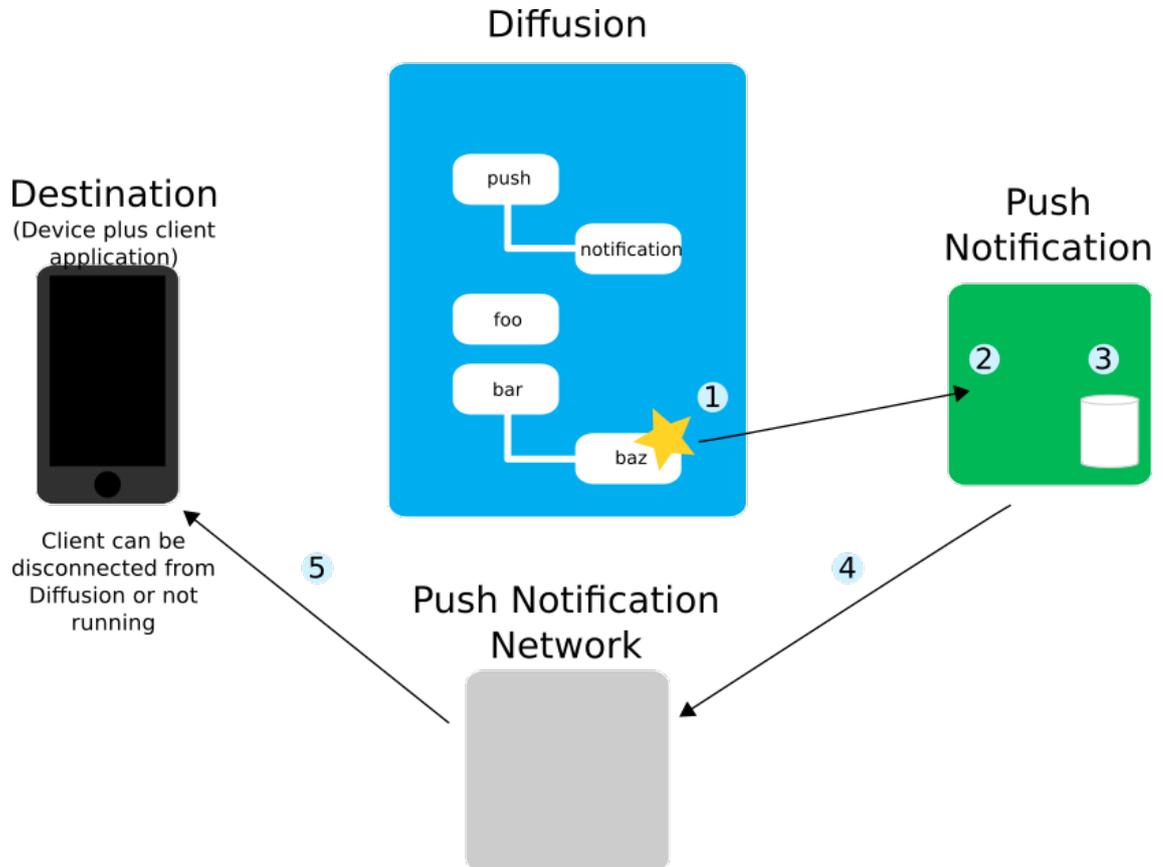


Figure 43: Notifications from the Push Notification Bridge

1. The topic is updated.
2. The Push Notification Bridge receives topic update and transforms the update into JSON according to the template that is configured for the topic.

For more information about the JSON format of notifications, see [Push notification JSON format](#) on page 674. For more information about configuring templates, see [PushNotifications.xml](#) on page 665.

3. The bridge looks up the destinations that have subscribed to receive push notifications for this topic.
4. The bridge sends the push notifications to the push notification network.

The push notification network that the bridge uses depends on the transport prefix in the destination URI provided in the subscription request message.

5. The push notification network sends the notification to the destination.

If the client is active when the topic update occurs and is subscribed to that topic, the update is received twice: once through the Diffusion server and once through the push notification network. It is the responsibility of the client to handle any duplication.

Related concepts

[Configuring your Push Notification Bridge](#) on page 663

Use the `PushNotification.xml` configuration file to define the behavior of your Push Notification Bridge.

[Running the Push Notification Bridge](#) on page 670

The Diffusion installation includes scripts that you can use to start the Push Notification Bridge.

[Push notification networks](#) on page 123

Consider whether your solution will interact with push notification networks.

[Push Notification Bridge persistence plugin](#) on page 511

The Push Notification Bridge stores subscription information in memory. To persist this information past the end of the bridge process, implement a persistence plugin.

[Example: Send a request message to the Push Notification Bridge](#) on page 372

The following examples use the Unified API to send a request message on a topic path to communicate with the Push Notification Bridge. The request message is in JSON and can be used to subscribe or unsubscribe from receiving push notifications when specific topics are updated.

Related reference

[JSON formats used by the Push Notification Bridge](#) on page 670

Requests and responses sent between clients and the Push Notification Bridge on the bridge's service topic and the push notifications sent by the bridge to devices are all JSON format.

[Push notification JSON format](#) on page 674

When a topic is updated, the Push Notification Bridge sends a notification through either APNS or GCM. This message is in JSON format. You can define the format of the message in a template in the `PushNotifications.xml` configuration file. If the update is in the correct JSON format, you can relay the update verbatim to the push notification network.

[Request and response JSON formats](#) on page 671

A client sends push notification requests to the topic path that the Push Notification Bridge listens on. The bridge responds through the same topic path. The default topic path is `push/notifications`. These requests and responses are in JSON format.

[PushNotifications.xml](#) on page 665

This file specifies the schema for the configuration required by the Push Notification Bridge.

Configuring your Push Notification Bridge

Use the `PushNotification.xml` configuration file to define the behavior of your Push Notification Bridge.

Configuring the Diffusion server connection

Consider the following when configuring the server connection:

- The URL to the Diffusion server must include both the prefix for the transport that the bridge uses to connect to the Diffusion server and the port number that the Diffusion server accepts client connections on.

The following transport prefixes are supported:

- `ws://`
- `wss://`
- `http://`
- `https://`
- DEPRECATED: `dpt://`
- DEPRECATED: `dpts://`

- The principal you use to connect to the Diffusion server must have a role that includes the following permissions:
 - `select_topic` and `read_topic` permissions for all topics that the bridge sends push notifications for and for the topic path that the bridge receives requests through.
 - `send_to_session` permission for the topic path that the bridge sends responses through.
 - `register_handler` permission

Configuring templates

You can define template notification messages within the configuration file that are associated with one or more topics. When an update is received on a topic, the associated template is used to format the update information into the JSON that is passed to the push notification networks.

Because the templates are defined in an XML configuration file, ensure that the configuration file remains valid by escaping any characters that are not valid XML. (For example, use `&` to escape `&`.)

You can also specify that topic updates be passed to the push notification network verbatim. In this case, it is the responsibility of the client or publisher updating the topic to ensure that the update content is in the correct JSON format. If the update content is not in the correct JSON format, the Push Notification Bridge logs an error.

If there is no matching template, verbatim relay is the default.

For more information about the JSON format of push notifications, see [Push notification JSON format](#) on page 674.

Configuring persistence

The Push Notification Bridge stores its data in memory. If you want the bridge to persist the notification requests your clients set up after the bridge has restarted, you must persist the data.

Use the `persistence` element to specify the implementation of the `com.pushtechnology.diffusion.pushnotifications.persistence` API to use.

For more information about implementing a persistence solution for your Push Notification Bridge, see [Push Notification Bridge persistence plugin](#) on page 511.

Configuring authentication

The Push Notification Bridge must authenticate with the push notification network you are using. Google Cloud Messaging uses an API key to authenticate your requests. Apple Push Notification Service uses a certificate. You must acquire these credentials before you can configure your Push Notification Bridge to use the push notification network.

For more information about getting the required credentials, see [Getting an Apple certificate for the Push Notification Bridge](#) on page 669 and [Getting a Google API key for the Push Notification Bridge](#) on page 669.

Related concepts

[Push Notification Bridge](#) on page 660

The Push Notification Bridge is a Diffusion client that subscribes to topics on behalf of other Diffusion clients and uses a push notification network to relay topic updates to the device where the client application is located.

[Running the Push Notification Bridge](#) on page 670

The Diffusion installation includes scripts that you can use to start the Push Notification Bridge.

Related reference

[JSON formats used by the Push Notification Bridge](#) on page 670

Requests and responses sent between clients and the Push Notification Bridge on the bridge's service topic and the push notifications sent by the bridge to devices are all JSON format.

[Push notification JSON format](#) on page 674

When a topic is updated, the Push Notification Bridge sends a notification through either APNS or GCM. This message is in JSON format. You can define the format of the message in a template in the `PushNotifications.xml` configuration file. If the update is in the correct JSON format, you can relay the update verbatim to the push notification network.

[Request and response JSON formats](#) on page 671

A client sends push notification requests to the topic path that the Push Notification Bridge listens on. The bridge responds through the same topic path. The default topic path is `push/notifications`. These requests and responses are in JSON format.

[PushNotifications.xml](#) on page 665

This file specifies the schema for the configuration required by the Push Notification Bridge.

PushNotifications.xml

This file specifies the schema for the configuration required by the Push Notification Bridge.

PNRootConfig

The mandatory root node of the Push Notification bridge.

The following table lists the elements that an element of type `PNRootConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
server	PNDiffusionConfig	Diffusion server connection details.	1	1
apns	PNAPNSConfig	Apple Push Notification Service connection and configuration details.	0	1
gcm	PNGCMConfig	Google Cloud Messaging connection and configuration details.	0	1
transformation	PNTransformationConfig	Transformation from topic updates to Push notifications.	1	1
persistence	PNPersistenceConfig	Optional Java plugin to persist topic subscriptions between bridge process lifetimes.	0	1
delivery	PNDeliveryConfig	Options relating to the delivery of notifications to all Push Notification networks.	1	1

PNDeliveryConfig

Options relating to the delivery of notifications to all Push Notification networks.

The following table lists the attributes that an element of type `PNDeliveryConfig` can have:

Name	Type	Description	Required
threads	xs:int	The number of threads used for notification delivery, processing client requests and collecting feedback from APNS.	true

PNTransformationConfig

Transformation from topic updates to Push notifications.

The following table lists the attributes that an element of type `PNTransformationConfig` can have:

Name	Type	Description	Required
default	xs:string		true

The following table lists the elements that an element of type `PNTransformationConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
templates	PNTemplatesConfig	The set of uniquely named templates.	1	1
map	PNTemplatesMappingConfig	Relates topics to one or more templates.	1	1

PNTemplatesConfig

The set of uniquely named templates.

The following table lists the elements that an element of type `PNTemplatesConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
template	PNTemplateConfig	A named template.	1	unbounded

PNTemplateConfig

A named template. May contain placeholders `${topic.name}` and/or `${topic.update}` which are replaced with real values at run time.

The following table lists the attributes that an element of type `PNTemplateConfig` can have:

Name	Type	Description	Required
name	xs:string	Name for this template, e.g. "london-underground-status-updates".	true

PNTemplatesMappingConfig

Establishes the relations between topic paths and templates. Alternatively updates can be passed through 'verbatim', meaning the topic update must be a valid push notification message.

The following table lists the elements that an element of type `PNTemplatesMappingConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
from	PNTemplatesMappingConfig	Relates updates from topics covered by a topic-selector to interpolation with a named template.	1	unbounded
verbatim	PNTemplatesMappingConfig	Relates a single topic-path to straight-through processing.	1	unbounded

PNAPNSConfig

Apple Push Notification Service connection and configuration details.

The following table lists the attributes that an element of type `PNAPNSConfig` can have:

Name	Type	Description	Required
certificate	xs:string	Name of a PKCS12 format file containing the certificate. The certificate needs to be of PKCS12 format and the keystore needs to be encrypted using the SunX509 algorithm. Both of these settings are the default.	true
passphrase	xs:string	Mandatory certificate passphrase.	true
servers	PNAPNSServers	A choice of "production", "sandbox" or "hostname:gatewayport:feedbackport"	true

PNGCMConfig

Google Cloud Messaging connection and configuration details.

The following table lists the attributes that an element of type `PNGCMConfig` can have:

Name	Type	Description	Required
apiKey	xs:string	The Google provided GCM API Key.	true
retryMax	xs:int	The number of attempts to make following an HTTP 50x response on posting. An increasing pause occurs prior to each repeated attempt.	false

PNDiffusionConfig

Diffusion server connection details.

The following table lists the attributes that an element of type `PNDiffusionConfig` can have:

Name	Type	Description	Required
url	xs:string	URL for a Diffusion connector configured to accept Unified clients.	true
principal	xs:string	The optional principal used when authenticating. The principal requires permissions 'send_to_session', 'read_topic' and 'register_handler'.	false
credentials	xs:string	The credentials matching the principal used when authenticating.	false
topicPath	xs:string	The topic path used by the Push Notification bridge to receive subscription and other service requests.	true

PNPersistenceConfig

Optional Java plugin to persist topic subscriptions between processes lifetimes.

The following table lists the attributes that an element of type `PNPersistenceConfig` can have:

Name	Type	Description	Required
saverFactory	xs:string	Name of the class present on the JVM classpath used to build a <code>com.pushtechnology.diffusion.pushnotifications.persistence.Saver</code> object.	true

PNTemplatesMappingFrom

Relates updates from topics covered by a topic-selector to interpolation with a named template.

The following table lists the attributes that an element of type `PNTemplatesMappingFrom` can have:

Name	Type	Description	Required
selector	xs:string	Diffusion topic selector expression.	true
toTemplate	xs:string	Name of a defined template.	true

PNTemplatesMappingVerbatim

The following table lists the attributes that an element of type `PNTemplatesMappingVerbatim` can have:

Name	Type	Description	Required
selector	xs:string	Relates a single topic-path to straight-through processing.	true

Related concepts

[Push Notification Bridge](#) on page 660

The Push Notification Bridge is a Diffusion client that subscribes to topics on behalf of other Diffusion clients and uses a push notification network to relay topic updates to the device where the client application is located.

[Configuring your Push Notification Bridge](#) on page 663

Use the `PushNotification.xml` configuration file to define the behavior of your Push Notification Bridge.

[Running the Push Notification Bridge](#) on page 670

The Diffusion installation includes scripts that you can use to start the Push Notification Bridge.

Related reference

[JSON formats used by the Push Notification Bridge](#) on page 670

Requests and responses sent between clients and the Push Notification Bridge on the bridge's service topic and the push notifications sent by the bridge to devices are all JSON format.

[Push notification JSON format](#) on page 674

When a topic is updated, the Push Notification Bridge sends a notification through either APNS or GCM. This message is in JSON format. You can define the format of the message in a template in the `PushNotifications.xml` configuration file. If the update is in the correct JSON format, you can relay the update verbatim to the push notification network.

[Request and response JSON formats](#) on page 671

A client sends push notification requests to the topic path that the Push Notification Bridge listens on. The bridge responds through the same topic path. The default topic path is push/notifications. These requests and responses are in JSON format.

Getting an Apple certificate for the Push Notification Bridge

For the Push Notification Bridge to connect to APNS, it must authenticate with a certificate. You can get a certificate from the Apple Developer Center.

About this task

Note:

These steps relate to the original APNS Provider API, where sandbox and production services each require a different certificate.

The Push Notification Bridge does not yet support the new APNS Provider API, which is based on HTTP/2 and requires only a single certificate for both sandbox and production roles.

Procedure

1. In the Apple Developer Center, go to the Member Center.
<https://developer.apple.com>
2. Go to **Certificates, Identities & Profiles**.
3. Click the plus sign (+) to add a new certificate.
4. From the **Development** section, select **Apple Push Notification service SSL (Sandbox)**.
After you have developed and tested your solution, you can get a production certificate.
5. Click Continue. Follow the instructions provided to generate a certificate.

What to do next

For more information, see [iOS Developer Library – App Distribution Guide](#).

Getting a Google API key for the Push Notification Bridge

For the Push Notification Bridge to connect to GCM, it must authenticate with an API key. You can get an API key from the Google Developers Console.

Procedure

1. In the Google Developers Console, create or select a project.
<https://console.developers.google.com/home/>
2. Go to **Use Google APIs**.
3. Select **Cloud Messaging for Android**.
4. Click **Enable API**.
You are prompted to create credentials.
5. Choose to skip the step and create an **API key**.
6. In the **Create a new key** dialog, select **Server key**.
7. Type a name for the key.
You can choose at this time to restrict the IP addresses that requests the use this API key can come from. If possible, restrict the key to the IP address your Push Notification Bridge uses.
8. Click **OK**.
A dialog displays your new API key.

Running the Push Notification Bridge

The Diffusion installation includes scripts that you can use to start the Push Notification Bridge.

System requirements

The Push Notification Bridge is a Java application. The bridge requires at least Java version 7. However, we recommend you use Java version 8 or above.

Starting with the scripts

The following startup scripts are provided in the `pushnotification` directory of your Diffusion installation:

- `pn_bridge.bat` for use on Windows platforms
- `pn_bridge.sh` for use on Linux and UNIX platforms

Related concepts

[Push Notification Bridge](#) on page 660

The Push Notification Bridge is a Diffusion client that subscribes to topics on behalf of other Diffusion clients and uses a push notification network to relay topic updates to the device where the client application is located.

[Configuring your Push Notification Bridge](#) on page 663

Use the `PushNotification.xml` configuration file to define the behavior of your Push Notification Bridge.

Related reference

[JSON formats used by the Push Notification Bridge](#) on page 670

Requests and responses sent between clients and the Push Notification Bridge on the bridge's service topic and the push notifications sent by the bridge to devices are all JSON format.

[Push notification JSON format](#) on page 674

When a topic is updated, the Push Notification Bridge sends a notification through either APNS or GCM. This message is in JSON format. You can define the format of the message in a template in the `PushNotifications.xml` configuration file. If the update is in the correct JSON format, you can relay the update verbatim to the push notification network.

[Request and response JSON formats](#) on page 671

A client sends push notification requests to the topic path that the Push Notification Bridge listens on. The bridge responds through the same topic path. The default topic path is `push/notifications`. These requests and responses are in JSON format.

[PushNotifications.xml](#) on page 665

This file specifies the schema for the configuration required by the Push Notification Bridge.

JSON formats used by the Push Notification Bridge

Requests and responses sent between clients and the Push Notification Bridge on the bridge's service topic and the push notifications sent by the bridge to devices are all JSON format.

Related concepts

[Push Notification Bridge](#) on page 660

The Push Notification Bridge is a Diffusion client that subscribes to topics on behalf of other Diffusion clients and uses a push notification network to relay topic updates to the device where the client application is located.

[Configuring your Push Notification Bridge](#) on page 663

Use the `PushNotification.xml` configuration file to define the behavior of your Push Notification Bridge.

[Running the Push Notification Bridge](#) on page 670

The Diffusion installation includes scripts that you can use to start the Push Notification Bridge.

Related reference

[Push notification JSON format](#) on page 674

When a topic is updated, the Push Notification Bridge sends a notification through either APNS or GCM. This message is in JSON format. You can define the format of the message in a template in the `PushNotifications.xml` configuration file. If the update is in the correct JSON format, you can relay the update verbatim to the push notification network.

[Request and response JSON formats](#) on page 671

A client sends push notification requests to the topic path that the Push Notification Bridge listens on. The bridge responds through the same topic path. The default topic path is `push/notifications`. These requests and responses are in JSON format.

[PushNotifications.xml](#) on page 665

This file specifies the schema for the configuration required by the Push Notification Bridge.

Request and response JSON formats

A client sends push notification requests to the topic path that the Push Notification Bridge listens on. The bridge responds through the same topic path. The default topic path is `push/notifications`. These requests and responses are in JSON format.

The following pieces of information are included in request messages:

destination_token

Push notification networks use binary tokens to represent an app installed on a device. This token combined with the transport prefix for the push notification network is the URI that the Push Notification Bridge uses to identify the device to send push notifications to.

topic_selector

The topic selector that the client subscribes to receive push notifications from or unsubscribes from receiving push notifications.

Note: This is not the same as the topic path that the request and response messages are sent through.

correlation_id

This value is passed through the Push Notification Bridge without being altered. The requesting client can use this to correlate a response message with its associated request message.

The results of including the contents of a subscription request with an unsubscription request are undefined.

Destination tokens

The destination token associated with the device and application to send a push notification to is provided to the application when the application registers with the push notification network.

Google Cloud Messaging

The destination token used by GCM is called *registration token* or *instance ID*. To get the instance ID for GCM, your client registers with the GCM connection servers using `instanceId.getToken`.

For more information, see [the GCM documentation](#).

Apple Push Notification service

The destination token used by APNs is called *device token*. Use the `registerForRemoteNotifications` method on your `UIApplication` instance to get a device token.

For more information, see [the APNs documentation](#).

When your app has successfully registered with the APNs, your `UIApplicationDelegate` instance is supplied with a device token through `application:didRegisterForRemoteNotificationsWithDeviceToken`.

Encode the device token in base 64 before you supply it to the Push Notification Bridge as an `apns://` URI in a bridge subscription request:

```
-(void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData
*)deviceToken {
    NSString * base64 = [deviceToken
base64EncodedStringWithOptions:0];
    NSString * destination = [@"apns://"
stringByAppendingString:base64];
    [self sendRequestWithDestination:destination];
}
```

Subscription request

The following message requests that updates to the topic at *topic_selector* be sent by APNs to the device identified by *destination_token*. The destination token is encoded in base64.

```
{
  "request": {
    "content": {
      "pns": {
        "destination": "apns://destination_token",
        "topic": "topic_selector"
      }
    },
    "correlation": "correlation_id"
  }
}
```

The following message requests that updates to the topic at *topic_selector* be sent by GCM to the device identified by *destination_token*.

```
{
  "request": {
    "content": {
      "pns": {
        "destination": "gcm://destination_token",
        "topic": "topic_selector"
      }
    },
    "correlation": "correlation_id"
  }
}
```

```
}
```

Unsubscription request

The following message requests that updates to the topic at *topic_selector* be no longer sent to the device identified by *destination_token*.

```
{
  "request": {
    "content": {
      "pnunsub": {
        "destination": "apns://destination_token",
        "topic": "topic_selector"
      }
    },
    "correlation": "correlation_id"
  }
}
```

The following message requests that updates to the topic at *topic_selector* be no longer sent to the device identified by *destination_token*.

```
{
  "request": {
    "content": {
      "pnunsub": {
        "destination": "gcm://destination_token",
        "topic": "topic_selector"
      }
    },
    "correlation": "correlation_id"
  }
}
```

Response

The following message is the response that the bridge sends to a requesting client if the request is successful.

```
{
  "response": {
    "content": result_json,
    "correlation": "correlation_id"
  }
}
```

If the response message contains a *content* element, the request was successful.

The *correlation_id* is the same value that the requesting client provided in the *correlation* field of the associated request.

Error response

The following message is the response that the bridge sends to a requesting client if an error occurs when processing the request.

```
{
  "response": {
    "error": "exception_text",
    "correlation": "correlation_id"
  }
}
```

```
}  
}
```

If the response message contains an `error` element, the request was not successful. More information about the reason for the failure, is available in the `exception_text`

The `correlation_id` is the same value that the requesting client provided in the `correlation` field of the associated request.

Related concepts

[Push Notification Bridge](#) on page 660

The Push Notification Bridge is a Diffusion client that subscribes to topics on behalf of other Diffusion clients and uses a push notification network to relay topic updates to the device where the client application is located.

[Configuring your Push Notification Bridge](#) on page 663

Use the `PushNotification.xml` configuration file to define the behavior of your Push Notification Bridge.

[Running the Push Notification Bridge](#) on page 670

The Diffusion installation includes scripts that you can use to start the Push Notification Bridge.

Related reference

[JSON formats used by the Push Notification Bridge](#) on page 670

Requests and responses sent between clients and the Push Notification Bridge on the bridge's service topic and the push notifications sent by the bridge to devices are all JSON format.

[Push notification JSON format](#) on page 674

When a topic is updated. The Push Notification Bridge sends a notification through either APNS or GCM. This message is in JSON format. You can define the format of the message in a template in the `PushNotifications.xml` configuration file. If the update is in the correct JSON format, you can relay the update verbatim to the push notification network.

[PushNotifications.xml](#) on page 665

This file specifies the schema for the configuration required by the Push Notification Bridge.

Push notification JSON format

When a topic is updated. The Push Notification Bridge sends a notification through either APNS or GCM. This message is in JSON format. You can define the format of the message in a template in the `PushNotifications.xml` configuration file. If the update is in the correct JSON format, you can relay the update verbatim to the push notification network.

Using templates

Templates are configured in the `PushNotification.xml` file. These templates are associated with specific topics. When a topic is updated, the associated template is applied to that update before the update is sent through the push notification network. The template uses the following placeholders to include topic and update information in the notification message:

`${topic.path}`

The path that the update is received on.

`${topic.update}`

The content of the update.

A template notification message can include both an `apns` section and a `gcm` section. The size of the data within each of these sections is restricted by the push notification network and currently cannot be greater than 2 KB.

After the update is transformed by the template, only the section of the notification message that is in the appropriate format for that push notification network is passed to the push notification network to be sent on to the client.

Verbatim relay

You can also specify that topic updates be passed to the push notification network verbatim. In this case, it is the responsibility of the client or publisher updating the topic to ensure that the update content is in the correct JSON format. If the update content is not in the correct JSON format, the Push Notification Bridge logs an error.

If there is no matching template, verbatim relay is the default.

APNS

The `apns` section wraps Apple's JSON format for an APNS message. For more information, see <https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html>

The `apns` section of the transformed notification message is used when the destination URI starts with `apns://` and the notification is sent through the Apple Push Notification Service

```
"apns": {
  "aps": {
    "alert": {
      "title": "notification_title",
      "body": "notification_content"
    }
  },
  User-defined key-value pairs
}
```

GCM

The `gcm` segment is a representation of Google's package `com.google.android.gcm.server`, which defines the following headers:

`collapseKey`

A unique identifier for a group of notifications that can be collapsed. When an idle device becomes active again, only the most recent notification for any given collapse key is sent.

For example, the topic path of the topic that the bridge subscribes to can be used as the collapse key and inactive devices are sent only the most recent update to that topic when they become active again.

Note: A maximum of 4 different collapse keys are stored simultaneously by GCM.

`delayWhileIdle`

When the value of this field is set to true, notifications are not be sent until the device becomes active. The default value is false.

`dryRun`

When the value of this field is set to true, you can test a request without actually sending a message. The default value is false.

`timeToLive`

How long in seconds the notification is kept in GCM storage if the device is offline. The maximum time to live supported is 4 weeks.

The `gcm` segment also contains a `data` section that contains a JSON payload. This JSON payload must be a dictionary of key-value pairs where both the key and the value are strings.

The `gcm` section of the transformed notification message is used when the destination URI starts with `gcm://` and the notification is sent through the Google Cloud Messaging

```
"gcm": {
  "collapseKey": "group_identifier",
  "delayWhileIdle": boolean,
  "timeToLive": integer,
  "data": {
    User-defined key-value pairs
  }
}
```

Notification message example

The following example shows the results of an online auction being sent as a push notification in both APNS and GCM format.

```
{
  "apns": {
    "aps": {
      "alert": {
        "title": "Auction Result",
        "body": "You won the auction for 'Antique oak table'"
      }
    },
    "auctionID": "abc123xyz789",
    "auctionConclusion": 123456789
  },
  "gcm": {
    "collapseKey": "abc123xyz789",
    "delayWhileIdle": false,
    "timeToLive": 60,
    "data": {
      "result": "You won the auction for 'Antique oak table'",
      "auctionID": "abc123xyz789",
      "auctionConclusion": 123456789
    }
  }
}
```

Related concepts

[Push Notification Bridge](#) on page 660

The Push Notification Bridge is a Diffusion client that subscribes to topics on behalf of other Diffusion clients and uses a push notification network to relay topic updates to the device where the client application is located.

[Configuring your Push Notification Bridge](#) on page 663

Use the `PushNotification.xml` configuration file to define the behavior of your Push Notification Bridge.

[Running the Push Notification Bridge](#) on page 670

The Diffusion installation includes scripts that you can use to start the Push Notification Bridge.

Related reference

[JSON formats used by the Push Notification Bridge](#) on page 670

Requests and responses sent between clients and the Push Notification Bridge on the bridge's service topic and the push notifications sent by the bridge to devices are all JSON format.

[Request and response JSON formats](#) on page 671

A client sends push notification requests to the topic path that the Push Notification Bridge listens on. The bridge responds through the same topic path. The default topic path is push/notifications. These requests and responses are in JSON format.

[PushNotifications.xml](#) on page 665

This file specifies the schema for the configuration required by the Push Notification Bridge.

JMS adapter

The JMS adapter for Diffusion, enables Diffusion clients to transparently send data to and receive data from destinations (topics and queues) on a JMS server.

The JMS adapter can be run within the Diffusion server or as a standalone client application.

JMS adapter

The JMS adapter comprises the following files all located in your Diffusion installation:

`adapters/jmsadapter.jar`

This JAR file contains the Diffusion Java application that links the Diffusion server and a JMS server.

`adapters/JMSAdapter.xml`

This XML file is used to configure the JMS adapter. For more information, see [JMSAdapter.xml](#) on page 695.

`xsd/JMSAdapter.xsd`

This XSD file defines the schema of the `JMSAdapter.xml` file.

`adapters/jms_adapter.sh` and `adapters/jms_adapter.bat`

These executable files can be used to start the JMS adapter when running it as a standalone client on UNIX, Linux, or Windows systems.

The JMS adapter can be run as a client on any system that has a Java 7 JRE installed on it.

Using the JMS adapter

To use the JMS adapter, first configure it by editing the `JMSAdapter.xml` to define the adapter behavior. For more information, see [Configuring the JMS adapter](#) on page 686.

The method for running the JMS adapter differs depending on whether you run it within the . For more information, see [Running the JMS adapter](#) on page 705.

DEPRECATED: JMS adapter v5.1

A legacy JMS adapter is also included in the Diffusion server installation. This adapter is provided only for customers already using it. We recommend that you use the JMS adapter to create any new JMS solutions.

Transforming JMS messages into Diffusion messages or updates

JMS messages are more complex than Diffusion content. A transformation is required between the two formats.

The following modes of transformation are provided:

Basic

Only the textual content of a message is relayed.

JSON

All JMS headers and properties are relayed, in addition to the textual content of the message. These values are expressed as JSON in the corresponding Diffusion message.

You can configure which one of these transformation modes your JMS adapter uses at the per topic level.

JMS message structure

JMS messages comprise headers, properties, and a payload. Currently, only JMS TextMessages are supported by the JMS adapter.

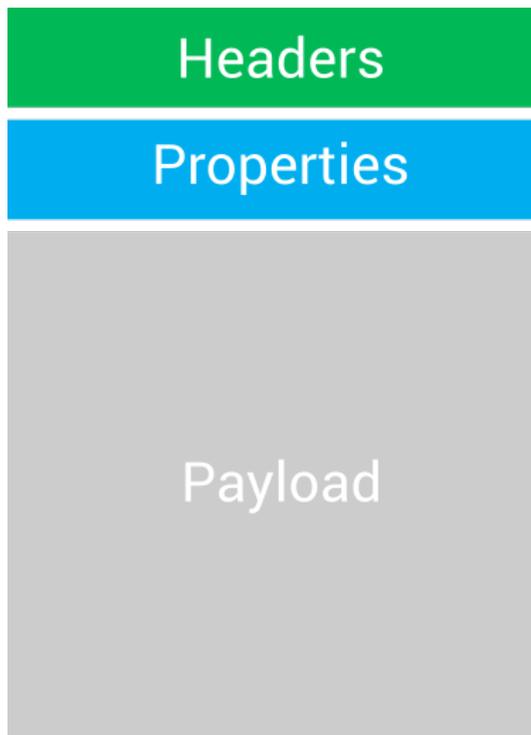


Figure 44: JMS message structure

Headers

This is a fixed set of properties whose names all begin with 'JMS'. Some, such as JMSDestination, are mandatory. Others are optional. For more information, see <https://docs.oracle.com/javaee/7/api/javax/jms/Message.html>.

Properties

A set of name-value pairs.

Payload

The contents of the message. This is a String.

Basic transformation

In a basic transformation only the textual payload or content of the message is relayed in either direction.

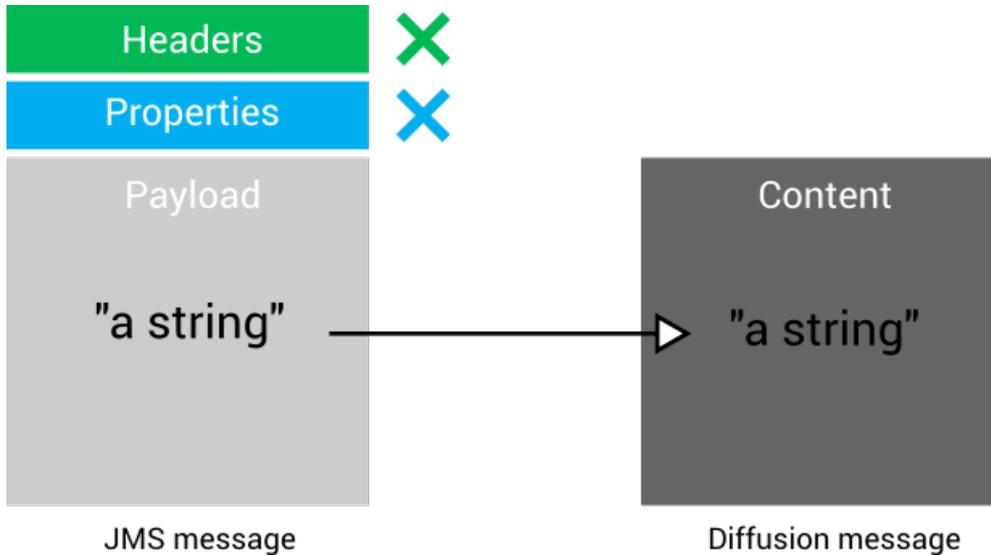


Figure 45: Basic mapping from a JMS message to a Diffusion message

When relaying a JMS message to Diffusion, the JMS adapter creates a Diffusion message whose content is the JMS message payload. The headers and properties of the JMS message are ignored.

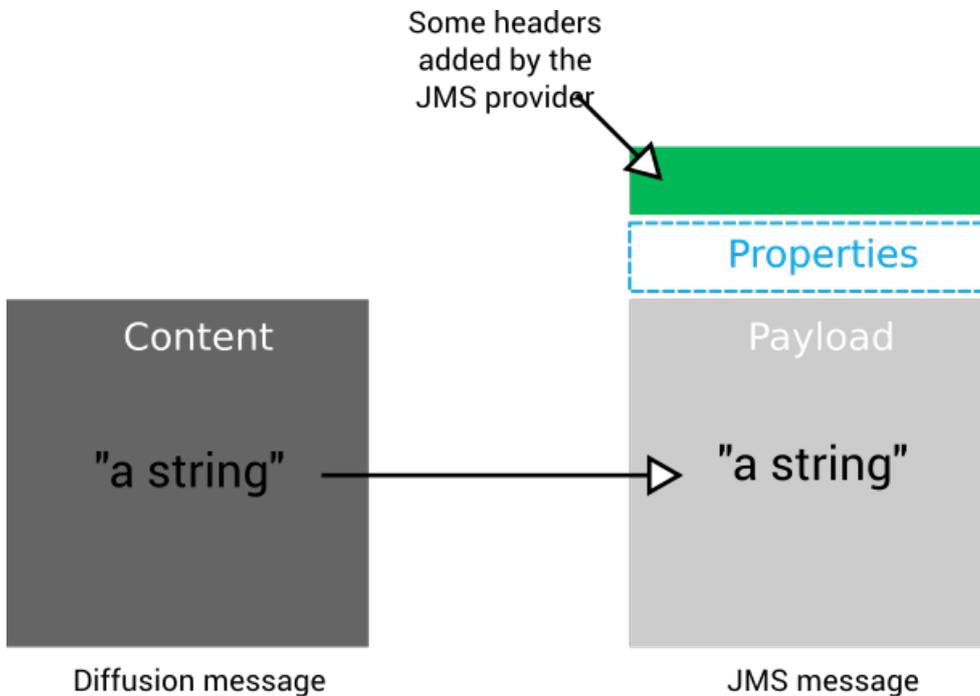


Figure 46: Basic mapping from a Diffusion message to a JMS message

When relaying a Diffusion message to JMS, the JMS adapter sets the JMS message payload to be the Diffusion content. The JMS adapter does not set any properties or headers on the JMS message. The JMS provider sets any mandatory headers that are required on the JMS message.

JSON transformation

In a JSON transformation all information is relayed both directions. The JMS message information is expressed in JSON format inside the Diffusion message content.

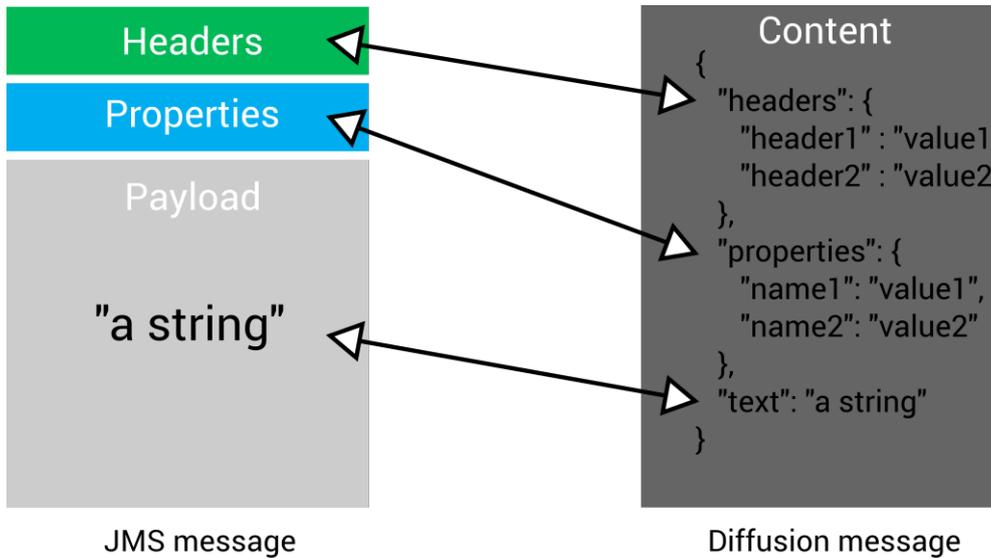


Figure 47: Mapping from a JMS message to and from JSON in a Diffusion message

When relaying a JMS message to Diffusion, the JMS adapter performs the following actions:

- Expresses the Diffusion content as a single JSON object.
- Maps the JMS message headers to a JSON object called “headers” inside of the Diffusion message content. The “headers” object contains all of the JMS message headers as name-value pairs. For example,

```
"headers": {
  "JMSType": "abc",
  "JMSPriority": 9
}
```

- Maps the JMS message properties to a JSON object called “properties” inside of the Diffusion message content. The “properties” object contains all of the JMS message properties as name-value pairs. For example,

```
"properties": {
  "AString": "def",
  "ABoolean": true
}
```

- Maps the textual payload of the JMS message to a JSON item called “text” inside of the Diffusion message content. For example,

```
"text": "Message content"
```

When relaying a Diffusion message to JMS, the JMS adapter parses the JSON content of the Diffusion message and uses the information to set the headers, properties, and payload of the JMS message accordingly.

Related concepts

[JMS](#) on page 124

Consider whether to incorporate JMS providers into your solution.

[Sending messages using the JMS adapter](#) on page 682

The JMS adapter can send messages from a Diffusion client to a JMS destination and messages from a JMS destination to a specific Diffusion client.

[Publishing using the JMS adapter](#) on page 681

The JMS adapter can publish data from a JMS destination onto topics in the Diffusion topic tree.

[Using JMS request-response services with the JMS adapter](#) on page 685

You can use the messaging capabilities of the JMS adapter to interact with a JMS service through request-response.

[Configuring the JMS adapter](#) on page 686

Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Related reference

[JMSAdapter.xml](#) on page 695

This file specifies the schema for the configuration required by the JMS adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Publishing using the JMS adapter

The JMS adapter can publish data from a JMS destination onto topics in the Diffusion topic tree.

Publishing data from a JMS destination onto a Diffusion topic

You can configure the JMS adapter to subscribe to a JMS destination and to associate that subscription with a Diffusion topic.

The Diffusion topic can be stateful or stateless, but stateful topics must be created with an initial value. For more information, see [Example: Configuring topics for use with the JMS adapter](#) on page 691.

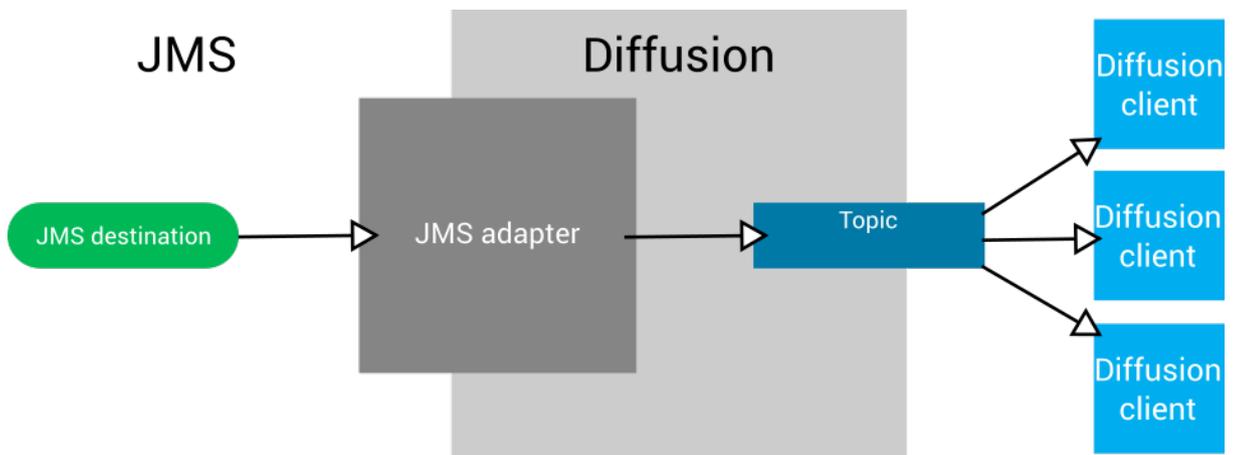


Figure 48: JMS adapter: Publishing from JMS to Diffusion

1. A message is published to the JMS destination.
2. The JMS adapter receives the JMS message.
3. The JMS adapter transforms the JMS message into a Diffusion message. For more information, see [Transforming JMS messages into Diffusion messages or updates](#) on page 678.
4. The JMS adapter publishes the transformed message to the Diffusion topic.

5. Diffusion clients that are subscribed to the Diffusion topic receive the transformed message.

Publishing data from a Diffusion topic to a JMS destination

This is not currently supported.

Related concepts

[JMS](#) on page 124

Consider whether to incorporate JMS providers into your solution.

[Transforming JMS messages into Diffusion messages or updates](#) on page 678

JMS messages are more complex than Diffusion content. A transformation is required between the two formats.

[Sending messages using the JMS adapter](#) on page 682

The JMS adapter can send messages from a Diffusion client to a JMS destination and messages from a JMS destination to a specific Diffusion client.

[Using JMS request-response services with the JMS adapter](#) on page 685

You can use the messaging capabilities of the JMS adapter to interact with a JMS service through request-response.

[Configuring the JMS adapter](#) on page 686

Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

[Example: Configuring pub-sub with the JMS adapter](#) on page 692

Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define JMS adapter subscriptions to JMS destinations and the Diffusion topics to publish updates to.

Related reference

[JMSAdapter.xml](#) on page 695

This file specifies the schema for the configuration required by the JMS adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Sending messages using the JMS adapter

The JMS adapter can send messages from a Diffusion client to a JMS destination and messages from a JMS destination to a specific Diffusion client.

Sending a message from a Diffusion client to a JMS destination

You can configure the JMS adapter to handle messages sent on a Diffusion message path and to associated messages received on that message path with a JMS destination.

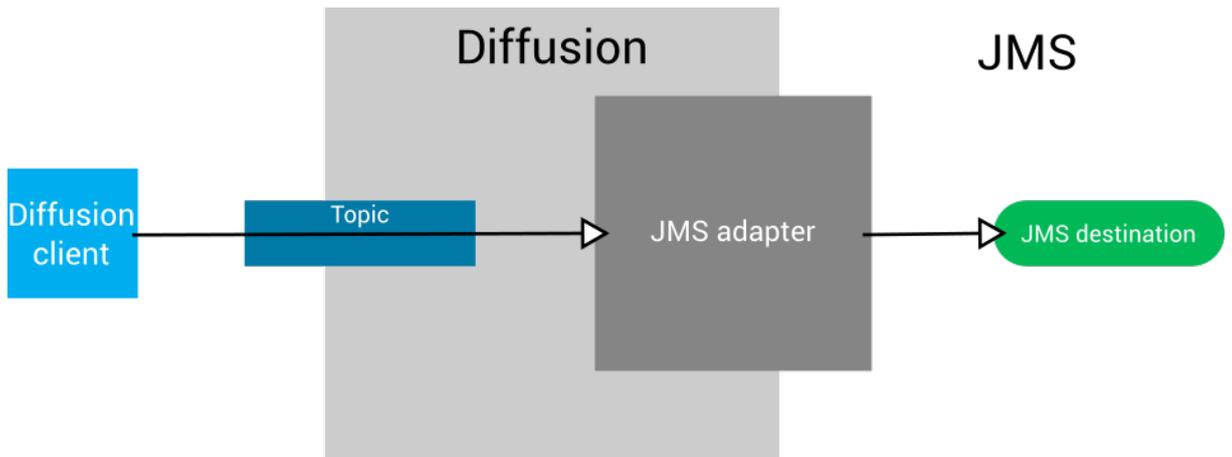


Figure 49: JMS adapter: Message flow from Diffusion to JMS

1. A Diffusion client sends a message to a topic path.
2. The JMS adapter receives the message.
3. The JMS adapter transforms the Diffusion message into a JMS message. For more information, see [Transforming JMS messages into Diffusion messages or updates](#) on page 678.
4. The JMS adapter sets a JMS header or property to include the Diffusion server name of the JMS adapter and the session ID of the Diffusion client.

This header or property is used as a return address for any response messages and is nominated using the `routingProperty` configuration element. By convention, `JMS CorrelationID` is often used. For more information, see [JMSAdapter.xml](#) on page 695.

5. The JMS adapter publishes the transformed message to the JMS destination.

Sending a message from a JMS destination to a Diffusion client

You can configure the JMS adapter to subscribe to a JMS destination and to associate that subscription with a Diffusion message path to send a message through.

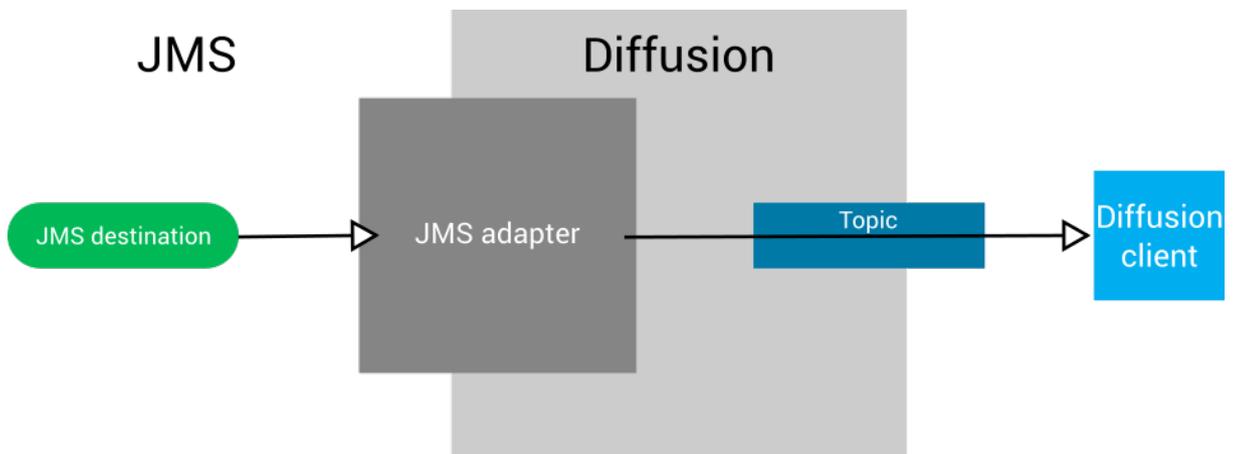


Figure 50: JMS adapter: Message flow from JMS to Diffusion

1. The JMS adapter receives a message from a JMS destination.
2. The JMS adapter transforms the JMS message into a Diffusion message. For more information, see [Transforming JMS messages into Diffusion messages or updates](#) on page 678.
3. The JMS adapter checks the nominated JMS header or property for the server name and session ID of the recipient client.

This header or property is nominated using the `routingProperty` configuration element. For more information, see [JMSAdapter.xml](#) on page 695.

4. The JMS adapter sends the transformed message through the message path to the recipient client session.

Error scenarios

- The JMS adapter consumes a message from a JMS destination that is not intended for it. That is, the routing property or header does not contain the Diffusion server name of the JMS adapter.

In this case, the JMS adapter drops the message and logs the failure to deliver.

You can avoid this scenario by using a JMS selector when subscribing to the JMS destination that specifies the JMS adapter is only interested in messages whose routing property or header include its Diffusion server name.

- The JMS adapter receives a message from a Diffusion client, but cannot send it on to JMS because the JMS provider is not connected.

In this case, the JMS adapter returns the message to the client on the same topic and logs the failure to deliver.

- The JMS adapter receives a message from a JMS destination, but cannot send it on to the Diffusion client because the Diffusion client is not connected.

In this case, the JMS adapter drops the message and logs the failure to deliver.

Related concepts

[JMS](#) on page 124

Consider whether to incorporate JMS providers into your solution.

[Transforming JMS messages into Diffusion messages or updates](#) on page 678

JMS messages are more complex than Diffusion content. A transformation is required between the two formats.

[Publishing using the JMS adapter](#) on page 681

The JMS adapter can publish data from a JMS destination onto topics in the Diffusion topic tree.

[Using JMS request-response services with the JMS adapter](#) on page 685

You can use the messaging capabilities of the JMS adapter to interact with a JMS service through request-response.

[Configuring the JMS adapter](#) on page 686

Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

[Example: Configuring messaging with the JMS adapter](#) on page 693

Use the `publications` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients send messages to JMS destinations. Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients receive messages from JMS destinations.

Related reference

[JMSAdapter.xml](#) on page 695

This file specifies the schema for the configuration required by the JMS adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Using JMS request-response services with the JMS adapter

You can use the messaging capabilities of the JMS adapter to interact with a JMS service through request-response.

Exposing a JMS service through Diffusion messaging is a typical use case for the JMS adapter.

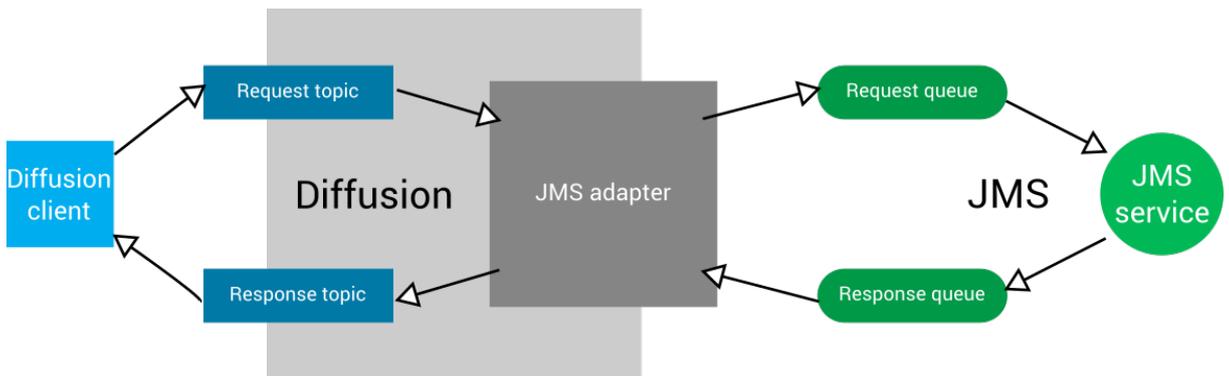


Figure 51: JMS adapter: Request-response message flow

1. A Diffusion client sends a message to a Diffusion topic path configured in the JMS adapter to receive service requests.
2. The JMS adapter receives the message on the request topic path.
3. The JMS adapter transforms the Diffusion message into a JMS message. For more information, see [Transforming JMS messages into Diffusion messages or updates](#) on page 678.
4. The JMS adapter adds a routing property or header to the JMS message identifying the Diffusion server and client to return a response to. This return information is of the form `server_name/client_session_id`.
5. The JMS adapter sends the message to the JMS service request queue.
6. The JMS service receives the request.
7. The JMS service acts on the request.
8. The JMS service places a response message on its response queue. This message must include the routing property or header that identifies the Diffusion server and client to return the response to.
9. The JMS adapter receives the response message from the JMS response queue.
10. The JMS adapter transforms the response message into a Diffusion message. For more information, see [Transforming JMS messages into Diffusion messages or updates](#) on page 678.
11. The JMS adapter uses the information in the routing property or header to discover the connected client session to relay the response to.
12. The JMS adapter sends the response message to the Diffusion client through a topic path.

Error scenarios

- The JMS adapter consumes a message from a JMS service response queue that is not intended for it. That is, the routing property or header does not contain the Diffusion server name of the JMS adapter.

In this case, the JMS adapter drops the message and logs the failure to deliver.

You can avoid this scenario by using a JMS selector when subscribing to the JMS queue that specifies the JMS adapter is only interested in messages whose routing property or header include its Diffusion server name.

- The JMS adapter receives a message from a Diffusion client, but cannot send it on to JMS because the JMS provider is not connected.

In this case, the JMS adapter returns the message to the client on the same topic and logs the failure to deliver.

- The JMS adapter receives a message from a JMS destination, but cannot send it on to the Diffusion client because the Diffusion client is not connected.

In this case, the JMS adapter drops the message and logs the failure to deliver.

Related concepts

[JMS](#) on page 124

Consider whether to incorporate JMS providers into your solution.

[Transforming JMS messages into Diffusion messages or updates](#) on page 678

JMS messages are more complex than Diffusion content. A transformation is required between the two formats.

[Sending messages using the JMS adapter](#) on page 682

The JMS adapter can send messages from a Diffusion client to a JMS destination and messages from a JMS destination to a specific Diffusion client.

[Publishing using the JMS adapter](#) on page 681

The JMS adapter can publish data from a JMS destination onto topics in the Diffusion topic tree.

[Configuring the JMS adapter](#) on page 686

Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

[Example: Configuring the JMS adapter to work with JMS services](#) on page 694

Use the `publications` and `subscriptions` elements of the `JMSAdapter.xml` configuration file to define the message flow for using Diffusion with JMS services.

Related reference

[JMSAdapter.xml](#) on page 695

This file specifies the schema for the configuration required by the JMS adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Configuring the JMS adapter

Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

The format of the `JMSAdapter.xml` configuration file is the same whether you run it within the Diffusion server or as a standalone client. However, the `server-connection` element and child elements are only used when the JMS adapter is run as a standalone client. If the `server-connection` element is included in a `JMSAdapter.xml` configuration file used when the JMS adapter runs within the Diffusion server, the JMS adapter ignores the element.

The schema of the `JMSAdapter.xml` is available in the `xsd` directory of the Diffusion installation. For more information, see [JMSAdapter.xml](#) on page 695.

In the `JMSAdapter.xml` file, you can configure the following aspects of the JMS adapter behavior:

- The JMS provider to connect to.

- For more information, see [Example: Configuring JMS providers for the JMS adapter](#) on page 689.
- The Diffusion topics to create and use.
 - For more information, see [Example: Configuring topics for use with the JMS adapter](#) on page 691.
- The JMS destinations to subscribe to and the Diffusion topics to publish data from the JMS destination to.
 - For more information, see [Example: Configuring pub-sub with the JMS adapter](#) on page 692.
- How messages are sent between Diffusion clients and JMS destinations through Diffusion topics.
 - For more information, see [Example: Configuring messaging with the JMS adapter](#) on page 693.
- A request-response message flow.
 - For more information, see [Example: Configuring the JMS adapter to work with JMS services](#) on page 694.

Configuring the JMS adapter to run within the Diffusion server

The JMS adapter running inside the Diffusion server uses the `JMSAdapter.xml` configuration file that is located in the `adapters` directory of the Diffusion installation.

When running inside the Diffusion server, the JMS adapter polls the `JMSAdapter.xml` file at five second intervals. If the timestamp changes in that interval, the JMS adapter reloads the configuration file.

When the JMS adapter reloads the configuration file, changes to the configuration are reflected in the set of Diffusion topics created and used by the JMS adapter:

- If the topic configuration is not changed, the topic is not changed on the Diffusion server.
- If a topic configuration is added, that topic is added on the Diffusion server.
- If a topic configuration is removed, that topic is deleted from the Diffusion server.
- If a topic configuration is changed – for example, if its definition is changed from stateful to stateless – that topic is deleted from the Diffusion server and a new topic is created at the same path.

Any removal of topics as part of a configuration update causes clients to become unsubscribed from the deleted topic.

When updating the `JMSAdapter.xml` configuration file on your running Diffusion server, consider using the following practices:

- Back up your original configuration file. For example, by moving it to `JMSAdapter.xml.bak`.
If a configuration file is not present, the JMS adapter continues to use its current configuration.
- Do not copy the new configuration file into place. Use a move operation instead. Move operations are atomic and remove the risk of the JMS adapter reading an incomplete file.
- In a production environment, rigorously test any new configuration file before deploying on a production server.

If the new configuration file contains an error, the configuration changes it contains are not applied. Instead the configuration rolls back to the original version and an error is logged.

Configuring the JMS adapter to run as a standalone client

When running as a standalone client, the JMS adapter uses the `JMSAdapter.xml` configuration file that is passed to the `jms_adapter.sh` or `jms_adapter.bat` file used to start the JMS adapter.

The JMS adapter standalone client loads the `JMSAdapter.xml` file only once, when the JMS adapter is started. To update the configuration used by the JMS adapter, edit the `JMSAdapter.xml` file and restart the JMS adapter.

Topics created by the JMS adapter when it runs as a standalone client remain on the Diffusion server after the JMS adapter session closes.

The `server-connection` element of the `JMSAdapter.xml` configuration file is used by the standalone client version of JMS adapter to define the connection that the JMS adapter makes to the Diffusion server. For more information, see [Example: Configuring the Diffusion connection for the JMS adapter running as a standalone client](#) on page 689.

Related concepts

[JMS](#) on page 124

Consider whether to incorporate JMS providers into your solution.

[Transforming JMS messages into Diffusion messages or updates](#) on page 678

JMS messages are more complex than Diffusion content. A transformation is required between the two formats.

[Sending messages using the JMS adapter](#) on page 682

The JMS adapter can send messages from a Diffusion client to a JMS destination and messages from a JMS destination to a specific Diffusion client.

[Publishing using the JMS adapter](#) on page 681

The JMS adapter can publish data from a JMS destination onto topics in the Diffusion topic tree.

[Using JMS request-response services with the JMS adapter](#) on page 685

You can use the messaging capabilities of the JMS adapter to interact with a JMS service through request-response.

[Example: Configuring JMS providers for the JMS adapter](#) on page 689

Use the `providers` element of the `JMSAdapter.xml` configuration file to define the JMS providers that the JMS adapter can connect to.

[Example: Configuring topics for use with the JMS adapter](#) on page 691

Use the `topics` element of the `JMSAdapter.xml` configuration file to define the Diffusion topics that the JMS adapter uses. These topics are created when the JMS adapter starts.

[Example: Configuring messaging with the JMS adapter](#) on page 693

Use the `publications` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients send messages to JMS destinations. Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients receive messages from JMS destinations.

[Example: Configuring pub-sub with the JMS adapter](#) on page 692

Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define JMS adapter subscriptions to JMS destinations and the Diffusion topics to publish updates to.

[Example: Configuring the JMS adapter to work with JMS services](#) on page 694

Use the `publications` and `subscriptions` elements of the `JMSAdapter.xml` configuration file to define the message flow for using Diffusion with JMS services.

Related reference

[JMSAdapter.xml](#) on page 695

This file specifies the schema for the configuration required by the JMS adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Example: Configuring the Diffusion connection for the JMS adapter running as a standalone client

Standalone client only: Use the `server-connection` element of the `JMSAdapter.xml` configuration file to define the connection that the JMS adapter makes to the Diffusion server.

```
<server-connection>
  <server url="transport://host:port" reconnection="timeout">
    <authentication principal="principal">
      <password>password</password>
    </authentication>
  </server>
  <properties>
    <serverName>name</serverName>
  </properties>
</server-connection>
```

- The `url` attribute of the `server` element is the URL of the Diffusion server to connect to, including the transport protocol and the port to use for the connection.
- The `authentication` element defines the principal and password to use to make the connection to the Diffusion server

Note: The JMS adapter requires a session that has the `TOPIC_CONTROL` role. Specify a principal with this role for the JMS adapter to use to make the connection.

- The `serverName` element is where you define a unique identifier to be used by the JMS adapter in correlation IDs used in messaging.

Example: Configuring JMS providers for the JMS adapter

Use the `providers` element of the `JMSAdapter.xml` configuration file to define the JMS providers that the JMS adapter can connect to.

Copy any provider JAR files that are required into the `ext` directory of your Diffusion server to ensure that they are on the server classpath.

ActiveMQ

You can connect to an ActiveMQ instance by defining a `provider` element that contains the required JNDI, credentials, and session information. See the following example:

```
<providers>
  <provider name="myActiveMQ">
    <jndiProperties>
      <property name="java.naming.factory.initial"
value="org.apache.activemq.jndi.ActiveMQInitialContextFactory"/>
      <property name="java.naming.provider.url"
value="tcp://hostname:61616"/>
    </jndiProperties>

    <jmsProperties connectionFactoryName="ConnectionFactory">
      <credentials>
        <username>user</username>
        <password>password</password>
      </credentials>
    </jmsProperties>
```

```

        <sessions>
            <anonymousSessions number="1" transacted="false"
            acknowledgeMode="AUTO_ACKNOWLEDGE" />
        </sessions>

    </provider>

</providers>

```

IBM MQ

You can connect to an IBM MQ instance by defining a `provider` element that contains the required information. See the following example:

```

<providers>
    <provider name="myIBMMQ">
        <jndiProperties>
            <property name="java.naming.factory.initial"
            value="com.sun.jndi.fscontext.RefFSContextFactory" />
            <property name="java.naming.provider.url"
            value="hostname:1414" />
        </jndiProperties>

        <jmsProperties connectionFactoryName="CF2">
            <sessions>
                <anonymousSessions number="2" transacted="false"
                acknowledgeMode="AUTO_ACKNOWLEDGE" />
            </sessions>
        </jmsProperties>

    </provider>
</providers>

```

Related concepts

[Example: Configuring topics for use with the JMS adapter](#) on page 691

Use the `topics` element of the `JMSAdapter.xml` configuration file to define the Diffusion topics that the JMS adapter uses. These topics are created when the JMS adapter starts.

[Example: Configuring messaging with the JMS adapter](#) on page 693

Use the `publications` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients send messages to JMS destinations. Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients receive messages from JMS destinations.

[Example: Configuring pub-sub with the JMS adapter](#) on page 692

Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define JMS adapter subscriptions to JMS destinations and the Diffusion topics to publish updates to.

[Example: Configuring the JMS adapter to work with JMS services](#) on page 694

Use the `publications` and `subscriptions` elements of the `JMSAdapter.xml` configuration file to define the message flow for using Diffusion with JMS services.

[Configuring the JMS adapter](#) on page 686

Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Related reference

[JMSAdapter.xml](#) on page 695

This file specifies the schema for the configuration required by the JMS adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Example: Configuring topics for use with the JMS adapter

Use the `topics` element of the `JMSAdapter.xml` configuration file to define the Diffusion topics that the JMS adapter uses. These topics are created when the JMS adapter starts.

The following example shows the definitions for a stateless and a stateful topic:

```
<topics>
  <stateless name="example/updates/stateless" />
  <stateful name="example/updates/stateful" initialState="rhubarb" />
</topics>
```

- The JMS adapter cannot create or use topics that are in a branch of the topic tree that is created by another publisher or client. For example, if `example/updates` already exists and was created by a Diffusion client, the JMS adapter cannot create and use `example/updates/stateless`.
- Similarly, other clients and publishers cannot create or use topics in a branch of the topic tree that was created by the JMS adapter.
- All stateful topics are created as single value topics.
- When defining a stateful topic, you must set the initial state of the topic.

Related concepts

[Example: Configuring JMS providers for the JMS adapter](#) on page 689

Use the `providers` element of the `JMSAdapter.xml` configuration file to define the JMS providers that the JMS adapter can connect to.

[Example: Configuring messaging with the JMS adapter](#) on page 693

Use the `publications` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients send messages to JMS destinations. Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients receive messages from JMS destinations.

[Example: Configuring pub-sub with the JMS adapter](#) on page 692

Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define JMS adapter subscriptions to JMS destinations and the Diffusion topics to publish updates to.

[Example: Configuring the JMS adapter to work with JMS services](#) on page 694

Use the `publications` and `subscriptions` elements of the `JMSAdapter.xml` configuration file to define the message flow for using Diffusion with JMS services.

[Configuring the JMS adapter](#) on page 686

Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Related reference

[JMSAdapter.xml](#) on page 695

This file specifies the schema for the configuration required by the JMS adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Example: Configuring pub-sub with the JMS adapter

Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define JMS adapter subscriptions to JMS destinations and the Diffusion topics to publish updates to.

The following example shows subscriptions to JMS destinations defined by the `destination` elements. When the JMS adapter receives an update message through the subscription, it publishes that update message to the Diffusion topic defined in the corresponding `publish` element.

```
<subscriptions>
  <subscription>
    <destination>jms:topic:EXAMPLE.UPDATE.TOPIC</destination>
    <publish topicName="example/updates/stateless" />
  </subscription>
  <subscription>
    <destination>jms:topic:EXAMPLE.UPDATE.TOPICTWO</destination>
    <publish topicName="example/updates/stateful" />
  </subscription>
</subscriptions>
```

The Diffusion topics must be defined in the `topics` section of the `JMSAdapter.xml` configuration file.

Related concepts

[Example: Configuring JMS providers for the JMS adapter](#) on page 689

Use the `providers` element of the `JMSAdapter.xml` configuration file to define the JMS providers that the JMS adapter can connect to.

[Example: Configuring topics for use with the JMS adapter](#) on page 691

Use the `topics` element of the `JMSAdapter.xml` configuration file to define the Diffusion topics that the JMS adapter uses. These topics are created when the JMS adapter starts.

[Example: Configuring messaging with the JMS adapter](#) on page 693

Use the `publications` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients send messages to JMS destinations. Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients receive messages from JMS destinations.

[Example: Configuring the JMS adapter to work with JMS services](#) on page 694

Use the `publications` and `subscriptions` elements of the `JMSAdapter.xml` configuration file to define the message flow for using Diffusion with JMS services.

[Publishing using the JMS adapter](#) on page 681

The JMS adapter can publish data from a JMS destination onto topics in the Diffusion topic tree.

[Configuring the JMS adapter](#) on page 686

Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Related reference

[JMSAdapter.xml](#) on page 695

This file specifies the schema for the configuration required by the JMS adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Example: Configuring messaging with the JMS adapter

Use the `publications` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients send messages to JMS destinations. Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients receive messages from JMS destinations.

From Diffusion clients to JMS destinations

The following example defines the Diffusion topic path through which the JMS adapter receives messages from a Diffusion client and the JMS destination to send those messages to.

```
<publications>
  <publication>
    <destination>jms:queue:EXAMPLE.REQUEST.QUEUE</destination>
    <messaging
      topicName="example/topic/requests"
      routingProperty="JMSCorrelationID">
      <transformation type="basic">
        <sessionProperties>
          <sessionProperty from="$Principal"
            to="diffusionPrincipal"/>
        </sessionProperties>
      </transformation>
    </messaging>
  </publication>
</publications>
```

- The `routingProperty` attribute describes the JMS header or property that the JMS adapter uses to set or get the client session ID.
- The `transformation` section defines how a message is transformed between a JMS message and a Diffusion message. For more information, see [Transforming JMS messages into Diffusion messages or updates](#) on page 678.
- The `sessionProperties` section defines whether the Diffusion session properties of the client that sends the message are included as JMS headers or properties in the transformed message. Currently, only `$Principal` is supported.

From JMS destinations to Diffusion clients

The following example defines the JMS destination that the JMS adapter retrieves messages on and the Diffusion topic path through which the JMS adapter relays those messages to a Diffusion client.

```
<subscriptions>
  <subscription>
    <destination>jms:queue:EXAMPLE.UPDATE.QUEUE</destination>
    <options noLocal="true"/>
    <messaging
      topicName="example/direct/messages"
      routingProperty="JMSCorrelationID"/>
  </subscription>
</subscriptions>
```

- The `routingProperty` attribute describes the JMS header or property that the JMS adapter uses to set or get the client session ID.

- The `noLocal` attribute of the `options` element defines whether the JMS adapter does not retrieve messages from a JMS queue that it is the originator of.

Related concepts

[Example: Configuring JMS providers for the JMS adapter](#) on page 689

Use the `providers` element of the `JMSAdapter.xml` configuration file to define the JMS providers that the JMS adapter can connect to.

[Example: Configuring topics for use with the JMS adapter](#) on page 691

Use the `topics` element of the `JMSAdapter.xml` configuration file to define the Diffusion topics that the JMS adapter uses. These topics are created when the JMS adapter starts.

[Example: Configuring pub-sub with the JMS adapter](#) on page 692

Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define JMS adapter subscriptions to JMS destinations and the Diffusion topics to publish updates to.

[Example: Configuring the JMS adapter to work with JMS services](#) on page 694

Use the `publications` and `subscriptions` elements of the `JMSAdapter.xml` configuration file to define the message flow for using Diffusion with JMS services.

[Sending messages using the JMS adapter](#) on page 682

The JMS adapter can send messages from a Diffusion client to a JMS destination and messages from a JMS destination to a specific Diffusion client.

[Configuring the JMS adapter](#) on page 686

Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Related reference

[JMSAdapter.xml](#) on page 695

This file specifies the schema for the configuration required by the JMS adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Example: Configuring the JMS adapter to work with JMS services

Use the `publications` and `subscriptions` elements of the `JMSAdapter.xml` configuration file to define the message flow for using Diffusion with JMS services.

In the following example, the `publications` section defines the JMS destination to put request messages on and the Diffusion topic path through which the JMS adapter receives those request messages from a Diffusion client. The `subscriptions` section defines the JMS destination that the JMS adapter retrieves response messages on and the Diffusion topic path through which the JMS adapter relays those messages to a Diffusion client.

```
<publications>
  <publication>
    <destination>jms:queue:REQUEST.QUEUE</destination>
    <messaging topicName="example/requests"
      routingProperty="JMSCorrelationID" />
  </publication>
</publications>
<subscriptions>
  <subscription>
    <destination>jms:queue:RESPONSE.QUEUE</destination>
    <options noLocal="true">
      <selector>JMSCorrelationID like '${serverName}/%'</
selector>
    </options>
    <messaging topicName="example/responses"
      routingProperty="JMSCorrelationID" />
  </subscription>
</subscriptions>
```

```
</subscription>
</subscriptions>
```

- The `routingProperty` attribute describes the JMS header or property that the JMS adapter uses to set or get the client session ID.
- The `noLocal` attribute of the `options` element defines whether the JMS adapter does not retrieve messages from a JMS queue that it is the originator of.
- When subscribing to a JMS destination, the JMS adapter can use selectors. In this example, the selector used requires that the routing property, in this case `JMSCorrelationID`, contains the name of the Diffusion server where the JMS adapter is deployed. This prevents the JMS adapter from consuming messages that are not intended for it.

The JMS adapter replaces the variable `${serverName}` with the name of its server. The server name is defined in the `serverName` element of the `JMSAdapter.xml` file when the JMS adapter runs as a standalone client. When the JMS adapter runs within the Diffusion server, the server name is defined by the `Server.xml` configuration file.

Related concepts

[Example: Configuring JMS providers for the JMS adapter](#) on page 689

Use the `providers` element of the `JMSAdapter.xml` configuration file to define the JMS providers that the JMS adapter can connect to.

[Example: Configuring topics for use with the JMS adapter](#) on page 691

Use the `topics` element of the `JMSAdapter.xml` configuration file to define the Diffusion topics that the JMS adapter uses. These topics are created when the JMS adapter starts.

[Example: Configuring messaging with the JMS adapter](#) on page 693

Use the `publications` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients send messages to JMS destinations. Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients receive messages from JMS destinations.

[Example: Configuring pub-sub with the JMS adapter](#) on page 692

Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define JMS adapter subscriptions to JMS destinations and the Diffusion topics to publish updates to.

[Using JMS request-response services with the JMS adapter](#) on page 685

You can use the messaging capabilities of the JMS adapter to interact with a JMS service through request-response.

[Configuring the JMS adapter](#) on page 686

Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Related reference

[JMSAdapter.xml](#) on page 695

This file specifies the schema for the configuration required by the JMS adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

JMSAdapter.xml

This file specifies the schema for the configuration required by the JMS adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

JMSRootConfig

The mandatory root node of the JMS adapter configuration.

The following table lists the elements that an element of type `JMSRootConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
topics	JMSTopicsConfig	The set of Diffusion topics created at startup.	0	1
providers	JMSProvidersConfig	The set of JMS providers.	1	1
server-connection	ServerConnectionConfig	Configuration specific to the JMS adapter when run as a client. When run as a publisher this is ignored.	0	1

JMSTopicsConfig

The set of Diffusion topics created at startup. Diffusion Unified client messaging does not require an existing topic, but Diffusion publishing and Diffusion Classic client messaging do.

The following table lists the elements that an element of type JMSTopicsConfig can contain:

Name	Type	Description	Min occurs	Max occurs
stateful	JMSStatefulTopicConfig	The configuration required to create a stateful Diffusion topic, including the initial state of the topic.	0	unbounded
stateless	JMSTopicConfig	The configuration required to create a Diffusion topic.	0	unbounded

JMSTopicConfig

The configuration required to create a Diffusion .

The following table lists the attributes that an element of type JMSTopicConfig can have:

Name	Type	Description	Required
name	xs:string	The full topic path of the topic to create. For example, 'foo/bar/baz'.	true
reference	xs:string	A string value to associate with this topic. For example, a description of the topic. Topic references are honored only when the JMS adapter runs as a publisher. DEPRECATED: Future versions of the product might not support this.	false

JMSStatefulTopicConfig

The following table lists the attributes that an element of type JMSStatefulTopicConfig can have:

Name	Type	Description	Required
initialState	xs:string	The initial state of a topic when the topic is created.	false

JNDIPropertiesConfig

The set of named values required to to create an InitialContext to access the JNDI configuration of the JMS server. Individual JMS providers will provide documentation on this step.

The following table lists the elements that an element of type `JNDIPropertiesConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
property	<code>JNDIProperty</code>	A named value.	0	unbounded

JNDIProperty

A named value.

The following table lists the attributes that an element of type `JNDIProperty` can have:

Name	Type	Description	Required
name	<code>xs:string</code>	The property name.	true
value	<code>xs:string</code>	The property value.	true

JMSProviderConfig

The configuration model to connect to a JMS provider (a broker), establish sessions, subscribe, and publish to destinations.

The following table lists the attributes that an element of type `JMSProviderConfig` can have:

Name	Type	Description	Required
name	<code>xs:string</code>	The name associated with this configuration model.	false

The following table lists the elements that an element of type `JMSProviderConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
jndiProperties	<code>JNDIPropertiesConfig</code>	The set of named values required to create an <code>InitialContext</code> to access the JNDI configuration of the JMS server.	1	1
jmsProperties	<code>JMSConnectionConfig</code>	The configuration related to connection to the JMS provider.	1	1
sessions	<code>JMSSessionsConfig</code>	The configuration for all JMS sessions related to this JMS provider.	1	1
reconnection	<code>JMSReconnectionConfig</code>	The configuration for reconnection behavior.	0	1
subscriptions	<code>JMSSubscriptionsConfig</code>	The set of subscriptions to JMS destinations.	0	1
publications	<code>JMSPublicationsConfig</code>	The set of publications to JMS destinations.	0	1

JMSProvidersConfig

The set of JMS providers.

The following table lists the elements that an element of type `JMSProvidersConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
provider	JMSProviderConfig	The configuration model to connect to a JMS provider.	0	unbounded

JMSSessionsConfig

The configuration for all JMS sessions related to this JMS provider.

The following table lists the elements that an element of type JMSSessionsConfig can contain:

Name	Type	Description	Min occurs	Max occurs
anonymousSessions	JMSAnonymousSessionsConfig	A number of JMS sessions shared between JMSSubscriptions.	1	1
namedSessions	JMSNamedSessionsConfig	The set of named JMS sessions, optionally used by JMSSubscription nodes in order to guarantee ordering, or use specific JMS session properties.	0	1

JMSAnonymousSessionsConfig

A number of JMS sessions shared between JMSSubscriptions.

The following table lists the attributes that an element of type JMSAnonymousSessionsConfig can have:

Name	Type	Description	Required
number	PositiveInteger	The number of shared JMS sessions	true

JMSNamedSessionsConfig

The set of named JMS sessions, optionally used by JMSSubscription nodes in order to guarantee ordering, or use specific JMS session properties.

The following table lists the elements that an element of type JMSNamedSessionsConfig can contain:

Name	Type	Description	Min occurs	Max occurs
session	JMSNamedSessionConfig	A named set of configuration relating to the placing of a JMS session.	1	unbounded

JMSReconnectionConfig

Following a disconnection event the adapter optionally attempts periodic reconnection. The first reconnection attempt occurs after minFrequency seconds, and the following after twice that number. The back-off time value doubles until it reaches the maxFrequency value in seconds. For example, where minFrequency=2 and maxFrequency=10, the reconnection will be attempted after 2s, 4s, 8s, 10s, 10s and so on.

The following table lists the attributes that an element of type JMSReconnectionConfig can have:

Name	Type	Description	Required
minFrequency	PositiveInteger	The interval between disconnection and the first reconnection attempt (in seconds). The interval is doubled for each subsequent reconnection attempt.	true
maxFrequency	PositiveInteger	The maximum interval between reconnection attempts.	true

JMSSubscriptionsConfig

The set of subscriptions to JMS destinations.

The following table lists the elements that an element of type `JMSSubscriptionsConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
subscription	JMSSubscriptionConfig	Configuration to subscribe to a JMS destination and relay to Diffusion topics or messaging or both.	0	unbounded

JMSPublicationsConfig

The set of publications to JMS destinations.

The following table lists the elements that an element of type `JMSPublicationsConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
publication	JMSPublicationConfig	Configuration to receive Diffusion topic messaging and relay to a JMS destination.	0	unbounded

JMSSubscriptionConfig

Configuration to subscribe to a JMS destination and relay to Diffusion topics

The following table lists the attributes that an element of type `JMSSubscriptionConfig` can have:

Name	Type	Description	Required
sessionName	xs:string	The name of the session to use. This session name must be defined in the <code>namedSessions</code> element. If this element is not defined, the JMS adapter does not start.	false

The following table lists the elements that an element of type `JMSSubscriptionConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
destination	JmsURI	The URI of the JMS destination.	1	1

Name	Type	Description	Min occurs	Max occurs
options	JMSSubscriptionOptions	Configuration relating to publishing in JMS.	0	1
messaging	ClientMessagingConfig	Configuration relating to the sending of a Diffusion message to a single Diffusion client.	0	1
publish	TopicPublishingConfig	Configuration relating to the publishing of a message or setting of a topic's state.	0	1

JMSPublicationConfig

Configuration to receive Diffusion topic messaging and relay to a JMS destination.

The following table lists the elements that an element of type JMSPublicationConfig can contain:

Name	Type	Description	Min occurs	Max occurs
destination	JmsURI	The URI of the JMS destination.	1	1
options	JMSPublicationOptions	Configuration relating to publishing to JMS destinations.	0	1
messaging	ClientMessagingConfig	Configuration relating to the sending of a Diffusion message to a single Diffusion client.	0	1

JMSSubscriptionOptions

Options employed when subscribing to a JMS destination.

The following table lists the attributes that an element of type JMSSubscriptionOptions can have:

Name	Type	Description	Required
noLocal	xs:boolean	Inhibits the delivery of messages published through its own connection.	false

The following table lists the elements that an element of type JMSSubscriptionOptions can contain:

Name	Type	Description	Min occurs	Max occurs
selector	xs:string	SQL 92 compliant expression used to filter messages received from a JMS destination.	0	1

JMSPublicationOptionsConfig

The following table lists the attributes that an element of type JMSPublicationOptionsConfig can have:

Name	Type	Description	Required
ttl	PositiveInteger	The Time-To-Live value for a published JMS message, in milliseconds	false
priority	JMSPriorityRange	The higher the number, the higher the priority.	false
deliveryMode	JMSDeliveryMode	Maps to javax.jms.DeliveryMode	false

ClientEndpointConfig

The following table lists the attributes that an element of type `ClientEndpointConfig` can have:

Name	Type	Description	Required
topicName	xs:string	The topic path used by this end point. Depending on the task it might not need to relate to an existing topic.	true

The following table lists the elements that an element of type `ClientEndpointConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
transformation	MessageTransformation	The transformation type to use for messages relayed to and from this topic.	0	1

MessageTransformationConfig

The following table lists the attributes that an element of type `MessageTransformationConfig` can have:

Name	Type	Description	Required
type	MessageTransformation	The transformation employed when relaying Diffusion to JMS messages, or JMS to Diffusion messages.	false

The following table lists the elements that an element of type `MessageTransformationConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
sessionProperties	SessionProperty	The set of session property mappings.	0	1

TopicPublishingConfig

Configuration relating to the publishing of a message or setting of a topic's state.

ClientMessagingConfig

Configuration relating to the sending of a Diffusion message to a single Diffusion client.

The following table lists the attributes that an element of type `ClientMessagingConfig` can have:

Name	Type	Description	Required
routingProperty	xs:string	The routingProperty attribute describes a facet of the JMS TextMessage that contains the destination Diffusion client SessionID.	false

SessionPropertyMapping

A mapping from Diffusion session properties to JMS message metadata (JMS headers or properties).

The following table lists the attributes that an element of type `SessionPropertyMapping` can have:

Name	Type	Description	Required
from	xs:string	Currently limited to \$Principal	true
to	xs:string	Values starting with "JMS" are mapped into JMS headers (for example, JMSType), others are mapped into JMS message properties.	true

SessionPropertyMappings

The set of `SessionPropertyMappings`.

The following table lists the elements that an element of type `SessionPropertyMappings` can contain:

Name	Type	Description	Min occurs	Max occurs
sessionProperty	SessionProperty	A session property name. Currently, only \$Principal is supported.	1	1

JMSCredentialsConfig

A username and password pair.

The following table lists the elements that an element of type `JMSCredentialsConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
username	xs:string	A username to use to connect to the JMS provider.	1	1
password	xs:string	The password associated with the username.	1	1

JMSConnectionConfig

The configuration related to connection to the JMS provider.

The following table lists the attributes that an element of type `JMSConnectionConfig` can have:

Name	Type	Description	Required
connectionFactoryName	xs:string	The name of the connection factory to use.	true

The following table lists the elements that an element of type `JMSConnectionConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
credentials	JMSCredentialsConfig	Optional credentials, used when connecting to the JMS provider	0	1

JMSSessionConfig

All configuration relating to the placing of a JMS session.

The following table lists the attributes that an element of type `JMSSessionConfig` can have:

Name	Type	Description	Required
transacted	xs:boolean	Currently unsupported.	false
acknowledgeMode	JMSAcknowledgeMode	Currently unsupported.	false

JMSNamedSessionConfig

A `JMSSessionConfig` that can be referred to by name.

The following table lists the attributes that an element of type `JMSNamedSessionConfig` can have:

Name	Type	Description	Required
name	xs:string	Name used to refer to the session elsewhere in the <code>JMSProviderConfig</code> .	true

ServerAuthenticationConfig

Optional session authentication details. The adapter defaults to establishing an anonymous session. Note: The JMS adapter requires a session that has the `TOPIC_CONTROL` role. Either assign the `TOPIC_CONTROL` role to the anonymous principal or specify a principal with this role for the JMS adapter to use to make the connection.

The following table lists the attributes that an element of type `ServerAuthenticationConfig` can have:

Name	Type	Description	Required
principal	xs:string	The principal used during authentication.	true

The following table lists the elements that an element of type `ServerAuthenticationConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
password	xs:string	Optional plain text password used during authentication.	1	1

ServerConfig

The following table lists the attributes that an element of type `ServerConfig` can have:

Name	Type	Description	Required
reconnection	SessionReconnect	The timeout duration in milliseconds used when attempting to reconnect, or 'none' to prevent any reconnection attempts.	false
url	xs:string	Location of the server to which the adapter connects.	false

The following table lists the elements that an element of type `ServerConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
authentication	ServerAuthentication	Optional session authentication details. The adapter defaults to establishing an anonymous session. Note: The JMS adapter requires a session that has the TOPIC_CONTROL role. Either assign the TOPIC_CONTROL role to the anonymous principal or specify a principal with this role for the JMS adapter to use to make the connection.	0	1

ClientEditionProperties

Set of properties defined for the JMS adapter running as a client.

The following table lists the elements that an element of type `ClientEditionProperties` can contain:

Name	Type	Description	Min occurs	Max occurs
serverName	xs:string	Mandatory value for placeholder '{serverName}' used in JMS request-reply scenarios.	1	1

ServerConnectionConfig

Configuration specific to the JMS adapter when run as a client. When run as a publisher this is ignored

The following table lists the elements that an element of type `ServerConnectionConfig` can contain:

Name	Type	Description	Min occurs	Max occurs
server	ServerConfig	Location and authentication details of the Diffusion server.	1	1
properties	ClientEditionProperties	Set of properties defined for the JMS adapter running as a client.	1	1

Related concepts

[Configuring the JMS adapter on page 686](#)

Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

[JMS](#) on page 124

Consider whether to incorporate JMS providers into your solution.

[Transforming JMS messages into Diffusion messages or updates](#) on page 678

JMS messages are more complex than Diffusion content. A transformation is required between the two formats.

[Sending messages using the JMS adapter](#) on page 682

The JMS adapter can send messages from a Diffusion client to a JMS destination and messages from a JMS destination to a specific Diffusion client.

[Publishing using the JMS adapter](#) on page 681

The JMS adapter can publish data from a JMS destination onto topics in the Diffusion topic tree.

[Using JMS request-response services with the JMS adapter](#) on page 685

You can use the messaging capabilities of the JMS adapter to interact with a JMS service through request-response.

[Example: Configuring JMS providers for the JMS adapter](#) on page 689

Use the `providers` element of the `JMSAdapter.xml` configuration file to define the JMS providers that the JMS adapter can connect to.

[Example: Configuring topics for use with the JMS adapter](#) on page 691

Use the `topics` element of the `JMSAdapter.xml` configuration file to define the Diffusion topics that the JMS adapter uses. These topics are created when the JMS adapter starts.

[Example: Configuring messaging with the JMS adapter](#) on page 693

Use the `publications` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients send messages to JMS destinations. Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients receive messages from JMS destinations.

[Example: Configuring pub-sub with the JMS adapter](#) on page 692

Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define JMS adapter subscriptions to JMS destinations and the Diffusion topics to publish updates to.

[Example: Configuring the JMS adapter to work with JMS services](#) on page 694

Use the `publications` and `subscriptions` elements of the `JMSAdapter.xml` configuration file to define the message flow for using Diffusion with JMS services.

Running the JMS adapter

The JMS adapter is not enabled by default.

Running the JMS adapter within the Diffusion server

The JMS adapter can run within the Diffusion server, but is not enabled by default. To enable the JMS adapter within the Diffusion server, complete the following steps:

1. Copy the `adapters/JMSAdapter.xml` configuration file into the `etc` directory.
2. Use the `JMSAdapter.xml` configuration file to define the JMS adapter behavior.

For more information, see [Configuring the JMS adapter](#) on page 686.

3. Use the `Publishers.xml` file to define and deploy the JMS adapter as a publisher on your Diffusion server:

```
<publisher name="JMSAdapter">
```

```
<class>com.pushtechnology.diffusion.adapters.jms.JMSAdapterPublisher</class>
<enabled>true</enabled>
<start>true</start>
</publisher>
```

4. Copy the `adapters/jmsadapter.jar` file into the `ext` directory of your Diffusion server to ensure that it is on the Diffusion server classpath.
5. Start or restart the Diffusion server.

Running the JMS adapter as a standalone client

The JMS adapter is a Java application. The adapter requires at least Java version 7. However, we recommend you use Java version 8 or above.

To run the JMS adapter as a client, complete the following steps:

1. To run the JMS adapter as a client on a different system to the Diffusion server, copy the following files from the Diffusion server system to the system where you want to locate the JMS adapter.
 - All files in the `adapters` directory
 - The Diffusion Java Unified API client library: `clients/java/diffusion-client.jar`
 - The SLF4J JAR file: `lib/thirdparty/slf4j-simple-1.7.21.jar`
 - A SLF4J bindings JAR files. For example Log4J2, which is provided in the Diffusion installation: `lib/thirdparty/log4j-*.jar`
2. Get the JAR file for the third-party JMS provider you use.
3. Use the `JMSAdapter.xml` configuration file to define the JMS adapter behavior.

For more information, see [Configuring the JMS adapter](#) on page 686.

4. Edit the `jms_adapter.sh` or `jms_adapter.bat` file to include the path to the Diffusion Java Unified API client library, SLF4J, and the JMS provider JAR on the classpath.
5. Use the `jms_adapter.sh` or `jms_adapter.bat` file to start the JMS adapter:

```
jms_adapter.sh relative_path/JMSAdapter.xml
```

DEPRECATED: Legacy JMS adapter

For backwards compatibility, Diffusion provides an older version of the JMS adapter, version 5.1. The version 5.1 JMS adapter is not compatible with the latest version of the JMS adapter.

The version 5.1 JMS adapter is packaged in the file `JMSAdapter51.jar` and configured using the `JMSAdapter51.xml` configuration file.

Note: We recommend that you do not develop new solutions using the version 5.1 JMS adapter. Instead use the latest version.

The JMS Adapter for Diffusion, enables Diffusion clients to transparently send and receive messages with topics and queues on a JMS server.

The behavior of a JMS adapter is configured in XML. No coding is required to use a JMS adapter.

The Diffusion server maintains a topic tree. Configure the JMS adapter to map a branch of the topic tree beneath a root topic to JMS topics and queues on your third-party JMS provider.

Considerations when using the JMS adapter

Dynamic topics

Diffusion topics are created dynamically and do not exist until a valid JMS subscription has been made and data received. Mapping to JMS queues is performed in the same way.

Message delivery

Delivery of messages to clients subscribing to a Diffusion topic that is mapped to a JMS queue is different to standard message delivery. Instead it is in keeping with the delivery characteristics of JMS queues.

When there are multiple Diffusion clients listening for data originating from the same JMS queue, each message is delivered to at most one client. Depending on the configuration of the JMS adapter the receiving client is chosen either randomly or based on the client with the fewest number of messages waiting for delivery.

Wildcards

You cannot subscribe to JMS topics using wildcards or topic selectors.

Temporary topics and queues

A common use for temporary queues is to set up a return path for request-reply operations.

A Diffusion client can request access to a JMS temporary topic or queue in the same way as subscribing to a JMS destination.

In the topic tree, temporary topics exist as sub-topics under the `.jms/tmp/topic` topic.

Acknowledgment

The only acknowledgment mode supported is `AUTO_ACKNOWLEDGE`. This implies that when any message is received from the JMS server, an acknowledgment is sent from the JMS adapter to the JMS server.

Acknowledgments sent from a Diffusion client to the JMS server (`CLIENT_ACKNOWLEDGE`) are not supported.

Message headers

The JMS adapter copies the standard JMS headers and user-supplied headers when converting between JMS and Diffusion message types.

In Diffusion, headers are logically grouped in pairs. Property n is the name, and $n+1$ is the value, where n is an even number. In the case of the `JMSReplyTo` header, the related header `DiffusionReplyTo` is created. Mapping between a `JMSReplyTo` destination and a `DiffusionReplyTo` topic is handled transparently by the adapter.

In most circumstances, a Diffusion client can ignore the `JMSReplyTo` header. The header is forwarded to the client for completeness.

Related concepts

[DEPRECATED: Configuring the legacy JMS adapter version 5.1](#) on page 708

Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Related tasks

[Configuring the JMS Adapter v5.1](#) on page 709

The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

Related reference

[DEPRECATED: Receiving data from JMS](#) on page 717

Diffusion clients can receive data from a JMS provider through the JMS adapter v5.1. A client can receive updates from a JMS topic or messages from a JMS queue.

[DEPRECATED: Sending messages to JMS](#) on page 719

Diffusion clients can send messages to a JMS provider through the JMS adapter v5.1.

[DEPRECATED: Processing a request-reply message with a Diffusion client](#) on page 720

A common pattern among JMS solutions is for a client to receive a message from JMS and a reply is expected to be sent to a specific JMS topic or queue.

[DEPRECATED: Sending a request-reply message from a Diffusion client](#) on page 722

You can send a message from a Diffusion client into a JMS server with the expectation that a JMS client processes the message and sends a response back to the same Diffusion client.

[DEPRECATED: JMS adapter data flow examples](#) on page 716

The examples in this section show how data flows between the Diffusion server and a JMS provider.

[JMSAdapter51.xml](#) on page 712

This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

DEPRECATED: Configuring the legacy JMS adapter version 5.1

Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Note: We recommend that you use the new JMS adapter. This legacy version is deprecated.

Use the `Publishers.xml` file to define and deploy the JMS adapter as a publisher on your Diffusion server:

```
<publisher name="JMSAdapter51">
  <class>com.pushtechnology.diffusion.adapters.jms51.JMSAdapter</class>
  <enabled>true</enabled>
  <start>true</start>
  <property name="config.filename">../adapters/JMSAdapter51.xml</property>
</publisher>
```

Related concepts

[DEPRECATED: Legacy JMS adapter](#) on page 706

For backwards compatibility, Diffusion provides an older version of the JMS adapter, version 5.1. The version 5.1 JMS adapter is not compatible with the latest version of the JMS adapter.

Related tasks

[Configuring the JMS Adapter v5.1](#) on page 709

The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

Related reference

[JMSAdapter51.xml](#) on page 712

This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

[DEPRECATED: Receiving data from JMS](#) on page 717

Diffusion clients can receive data from a JMS provider through the JMS adapter v5.1. A client can receive updates from a JMS topic or messages from a JMS queue.

[DEPRECATED: Sending messages to JMS](#) on page 719

Diffusion clients can send messages to a JMS provider through the JMS adapter v5.1.

DEPRECATED: [Processing a request-reply message with a Diffusion client](#) on page 720

A common pattern among JMS solutions is for a client to receive a message from JMS and a reply is expected to be sent to a specific JMS topic or queue.

DEPRECATED: [Sending a request-reply message from a Diffusion client](#) on page 722

You can send a message from a Diffusion client into a JMS server with the expectation that a JMS client processes the message and sends a response back to the same Diffusion client.

DEPRECATED: [JMS adapter data flow examples](#) on page 716

The examples in this section show how data flows between the Diffusion server and a JMS provider.

Configuring the JMS Adapter v5.1

The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

Procedure

1. Configure `Publishers.xml`

Instantiate the JMS adapter by enabling it in `etc/Publishers.xml`, for example:

```
<publisher name="JMSAdapter">
  <class>com.pushtechology.diffusion.adapters.jms51.JMSAdapter</class>
  <enabled>true</enabled>
  <start>true</start>
  <property name="config.filename">../adapters/JMSAdapter51.xml</property>
</publisher>
```

2. Configure `JMSAdapter51.xml`

A `JMSAdapter51.xml` for ActiveMQ can look like this:

```
<property name="use.global.session">false</property>
```

Similarly, a sample `JMSAdapter51.xml` TIBCO Enterprise Message Service™ looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<jms-config>
  <binding>
    <env>
      <property name="java.naming.factory.initial">
        com.tibco.tibjms.naming.TibjmsInitialContextFactory
      </property>
      <property name="java.naming.provider.url">
        tcp://localhost:7222
      </property>
      <property name="java.naming.security.principal">
        jndi_username
      </property>
      <property name="java.naming.security.credentials">
        jndi_password
      </property>
    </env>
    <connection-factory name="ConnectionFactory">
      <credentials username="jms_username" password="jms_password"/>
    >
    <reconnect>
```

```

<max-reconnections>10</max-reconnections>
<interval>5000</interval>
  </reconnect>
</connection-factory>
<root-topic>jms/tibco</root-topic>
<priority low="3" high="7" />
<queue-distribution-mode>SMALLEST_QUEUE</queue-distribution-
mode>
</binding>
<mapping>
  <artefact-names jms=". $" diffusion="/~"/>
</mapping>
</jms-config>

```

TIBCO Enterprise Message Service requires that a `ConnectionFactory` definition is provided in `factories.conf`, for example:

```

[ConnectionFactory]
  type = generic
  url = tcp://localhost:7222
  ssl_verify_host = disable

```

This has been tested with TIBCO Enterprise Message Service 7.0; see the TIBCO Enterprise Message Service documentation for further details.

For IBM MQ v7.x, the binding section can look like this:

```

<env>
  <property
    name=" java.naming.factory.initial">com.sun.jndi.fscontext.RefFSContextFactory
  </property>
  <property name=" java.naming.provider.url">file:///var/mqm/jndi</
  </property>
</env>

```

IBM MQ can include MQRFH2 headers in messages sent between JMS and non-JMS systems. For control over this behavior, set the property in `mq.target.client` in `Publishers.xml`; to disable the headers, set this value to 1. For the default behavior, do not provide the property.

```

<property name="mq.target.client">1</property>

```

Some JMS vendors (for example, IBM MQ) require a JMS `Session` for each topic or queue subscription. The default configuration for the JMS adapter does not allow for this, but you can enable it by setting the `use.global.session` property to `false` in `Publishers.xml`:

```

<property name="use.global.session">>false</property>

```

Table 64: Properties that can be specified when configuring the JMS adapter

<env>	All properties within the <env> tag of <code>JMSAdapter51.xml</code> are used when creating the <code>InitialContext</code> which is in turn used to create the connection to the JMS server.
<connection-factory>	This tag specifies the name of the connection factory to use. This varies between JMS vendors and the server configuration.

<credentials>	Optionally, specify a username and/or password which is used to create a JMS client connection. However, most JMS implementations are likely to have restricted access for anonymous clients or clients which do not require authentication so you can specify a user here who has the necessary privileges to receive and send messages to the JMS destinations that are exposed through Diffusion.
<reconnect>	If a connection cannot be made between Diffusion and the JMS server or the connection is severed, the <max-reconnections> and <interval> parameters enable you to specify how many times to retry the connection before giving up and how long to wait between each attempt. A value for <max-reconnections> of -1 indicates that the adapter keeps trying to connect forever.
<root-topic>	To provide partitioning of the topic tree between topics related to JMS and other topics, it is necessary for a root topic name is defined here. In the event of more than one JMS adapter, you can segment the topic tree further.
<priority>	JMS supports messages with up to 10 priority levels (0-9) with 0-4 considered to be different grades of normal priority and 5-9 to be different grades of high priority. Diffusion only has the concept of low, medium and high priority. Using this parameter, you can map JMS messages within a given priority range to a representative Diffusion priority, and in the other direction.
<queue-distribution-mode>	<p>Unlike topics, a message on a JMS queue is delivered to only one client. When Diffusion receives a message from a queue, it uses this parameter to determine which of its connected clients subscribed to the corresponding Diffusion topic is selected to receive that message. Valid values are:</p> <p>SMALLEST_QUEUE</p> <p>Choose a client with the fewest number of messages outstanding in its message queue from Diffusion.</p> <p>RANDOM</p> <p>Select a client randomly.</p>

<artifact-names>	Not all characters in a Diffusion topic name are valid JMS topic or queue names (and the other way around). The two attributes on this element (<code>jms</code> and <code>diffusion</code>) are lists of characters where the n^{th} character in one is replaced by the corresponding n^{th} character in the other.
------------------	--

Related concepts

[DEPRECATED: Configuring the legacy JMS adapter version 5.1](#) on page 708

Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

[DEPRECATED: Legacy JMS adapter](#) on page 706

For backwards compatibility, Diffusion provides an older version of the JMS adapter, version 5.1. The version 5.1 JMS adapter is not compatible with the latest version of the JMS adapter.

Related reference

[JMSAdapter51.xml](#) on page 712

This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

[DEPRECATED: Receiving data from JMS](#) on page 717

Diffusion clients can receive data from a JMS provider through the JMS adapter v5.1. A client can receive updates from a JMS topic or messages from a JMS queue.

[DEPRECATED: Sending messages to JMS](#) on page 719

Diffusion clients can send messages to a JMS provider through the JMS adapter v5.1.

[DEPRECATED: Processing a request-reply message with a Diffusion client](#) on page 720

A common pattern among JMS solutions is for a client to receive a message from JMS and a reply is expected to be sent to a specific JMS topic or queue.

[DEPRECATED: Sending a request-reply message from a Diffusion client](#) on page 722

You can send a message from a Diffusion client into a JMS server with the expectation that a JMS client processes the message and sends a response back to the same Diffusion client.

[DEPRECATED: JMS adapter data flow examples](#) on page 716

The examples in this section show how data flows between the Diffusion server and a JMS provider.

JMSAdapter51.xml

This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

jms-config

The following table lists the elements that an element of type `jms-config` can contain:

Name	Type	Description	Min occurs	Max occurs
binding	binding		1	1
mapping	mapping		1	1

binding

The following table lists the elements that an element of type `binding` can contain:

Name	Type	Description	Min occurs	Max occurs
env	env		1	1
connection-factory	connection-factory	A collection of properties required to create an InitialContext to access the JNDI configuration of the JMS server.	1	1
root-topic	push:string	The JMS Adapter uses the topic name specified here as the top-level Diffusion topic, with all subtopics appearing as children beneath it.	1	1
priority	priority	Specify how JMS message priorities (in the range 0-9) map to Diffusion message priorities high, normal or low.	0	1
queue-distribution-mode	push:jmsQueueDestination	Messages received from JMS queues are only sent to one Diffusion client. This element allows the distribution strategy to be chosen from RANDOM (choose any client) or SMALLEST_QUEUE (choose the client with the fewest pending messages in its queue from Diffusion).	1	1
topic-aliases	topic-aliases	This element is not currently used.	0	unbounded

mapping

The following table lists the elements that an element of type `mapping` can contain:

Name	Type	Description	Min occurs	Max occurs
artefact-names	artefact-names	Diffusion topic names and JMS destination names can both contain different reserved characters. The attributes on this element create a substitution table of characters. If a Diffusion topic name contains a character in the 'diffusion' attribute, it is replaced with the character in the same position of the 'jms' attribute when translating to a JMS topic or queue name (and vice-versa).	1	1
topic-map	jms-artefact-map	This element is not currently used.	0	unbounded
queue-map	jms-artefact-map	This element is not currently used.	0	unbounded

env

The following table lists the elements that an element of type `env` can contain:

Name	Type	Description	Min occurs	Max occurs
property	property	This element represents a property which is supplied when creating the JNDI object (InitialContext) from which JMS resources are obtained.	0	unbounded

connection-factory

Details for resolving the ConnectionFactory object from the nominated JNDI server.

The following table lists the attributes that an element of type `connection-factory` can have:

Name	Type	Description	Required
name	push:string		false

The following table lists the elements that an element of type `connection-factory` can contain:

Name	Type	Description	Min occurs	Max occurs
credentials	credentials	Username and password pair (optional), used when creating the JMS Connection from the ConnectionFactory.	0	1
reconnect	reconnect	This element is supplied to control the reconnection policy for the initial connection between Diffusion and the JMS server, and any subsequent reconnections that might be necessary.	0	1

jms-artefact-map

The following table lists the attributes that an element of type `jms-artefact-map` can have:

Name	Type	Description	Required
jms	push:string		true
descendants	push:boolean		false

artefact-names

The following table lists the attributes that an element of type `artefact-names` can have:

Name	Type	Description	Required
jms	push:string		true
diffusion	push:string		true

property

A named string property

The following table lists the attributes that an element of type `property` can have:

Name	Type	Description	Required
name	push:string	The property value	true

credentials

The following table lists the attributes that an element of type `credentials` can have:

Name	Type	Description	Required
username	push:string		false
password	push:string		false

priority

The following table lists the attributes that an element of type `priority` can have:

Name	Type	Description	Required
low	push:jmsMessagePriority		false
high	push:jmsMessagePriority		false

topic-aliases

The following table lists the elements that an element of type `topic-aliases` can contain:

Name	Type	Description	Min occurs	Max occurs
topic	topic		0	unbounded

topic

The following table lists the attributes that an element of type `topic` can have:

Name	Type	Description	Required
name	push:string		true
alias	push:string		true

reconnect

The following table lists the elements that an element of type `reconnect` can contain:

Name	Type	Description	Min occurs	Max occurs
max-reconnections	push:int	When attempting a connection between Diffusion and the JMS server, this parameter specifies how many attempts can be made before giving up. A value of -1 (the default) means to retry forever.	1	1

Name	Type	Description	Min occurs	Max occurs
interval	push:millis	This parameter specifies how long to wait between reconnection attempts between Diffusion and the JMS server.	1	1

Related concepts

DEPRECATED: [Configuring the legacy JMS adapter version 5.1](#) on page 708

Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

DEPRECATED: [Legacy JMS adapter](#) on page 706

For backwards compatibility, Diffusion provides an older version of the JMS adapter, version 5.1. The version 5.1 JMS adapter is not compatible with the latest version of the JMS adapter.

Related tasks

[Configuring the JMS Adapter v5.1](#) on page 709

The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

Related reference

DEPRECATED: [Receiving data from JMS](#) on page 717

Diffusion clients can receive data from a JMS provider through the JMS adapter v5.1. A client can receive updates from a JMS topic or messages from a JMS queue.

DEPRECATED: [Sending messages to JMS](#) on page 719

Diffusion clients can send messages to a JMS provider through the JMS adapter v5.1.

DEPRECATED: [Processing a request-reply message with a Diffusion client](#) on page 720

A common pattern among JMS solutions is for a client to receive a message from JMS and a reply is expected to be sent to a specific JMS topic or queue.

DEPRECATED: [Sending a request-reply message from a Diffusion client](#) on page 722

You can send a message from a Diffusion client into a JMS server with the expectation that a JMS client processes the message and sends a response back to the same Diffusion client.

DEPRECATED: [JMS adapter data flow examples](#) on page 716

The examples in this section show how data flows between the Diffusion server and a JMS provider.

DEPRECATED: JMS adapter data flow examples

The examples in this section show how data flows between the Diffusion server and a JMS provider.

Note: The JMS Adapter v5.1 is now deprecated. Use the new JMS adapter instead. For more information, see [JMS](#) on page 124.

The following scenarios assume that the JMS adapter v5.1 is configured with a root topic of `jms`.

Related concepts

DEPRECATED: [Legacy JMS adapter](#) on page 706

For backwards compatibility, Diffusion provides an older version of the JMS adapter, version 5.1. The version 5.1 JMS adapter is not compatible with the latest version of the JMS adapter.

DEPRECATED: [Configuring the legacy JMS adapter version 5.1](#) on page 708

Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Related tasks

[Configuring the JMS Adapter v5.1](#) on page 709

The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

Related reference

[DEPRECATED: Receiving data from JMS](#) on page 717

Diffusion clients can receive data from a JMS provider through the JMS adapter v5.1. A client can receive updates from a JMS topic or messages from a JMS queue.

[DEPRECATED: Sending messages to JMS](#) on page 719

Diffusion clients can send messages to a JMS provider through the JMS adapter v5.1.

[DEPRECATED: Processing a request-reply message with a Diffusion client](#) on page 720

A common pattern among JMS solutions is for a client to receive a message from JMS and a reply is expected to be sent to a specific JMS topic or queue.

[DEPRECATED: Sending a request-reply message from a Diffusion client](#) on page 722

You can send a message from a Diffusion client into a JMS server with the expectation that a JMS client processes the message and sends a response back to the same Diffusion client.

[JMSAdapter51.xml](#) on page 712

This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

[DEPRECATED: Receiving data from JMS](#)

Diffusion clients can receive data from a JMS provider through the JMS adapter v5.1. A client can receive updates from a JMS topic or messages from a JMS queue.

Note: The JMS Adapter v5.1 is now deprecated. Use the new JMS adapter instead. For more information, see [JMS](#) on page 124.

Receiving updates from a JMS topic

This section shows how a client receives updates from the JMS topic XYZ.

1. Diffusion client creates a subscription to the topic `jms/topic/XYZ`.
2. Once a message has been sent from the source system into the JMS server, the Diffusion client receives an initial topic load message.
3. Subsequent messages from the source system result in delta messages being delivered to the Diffusion client.

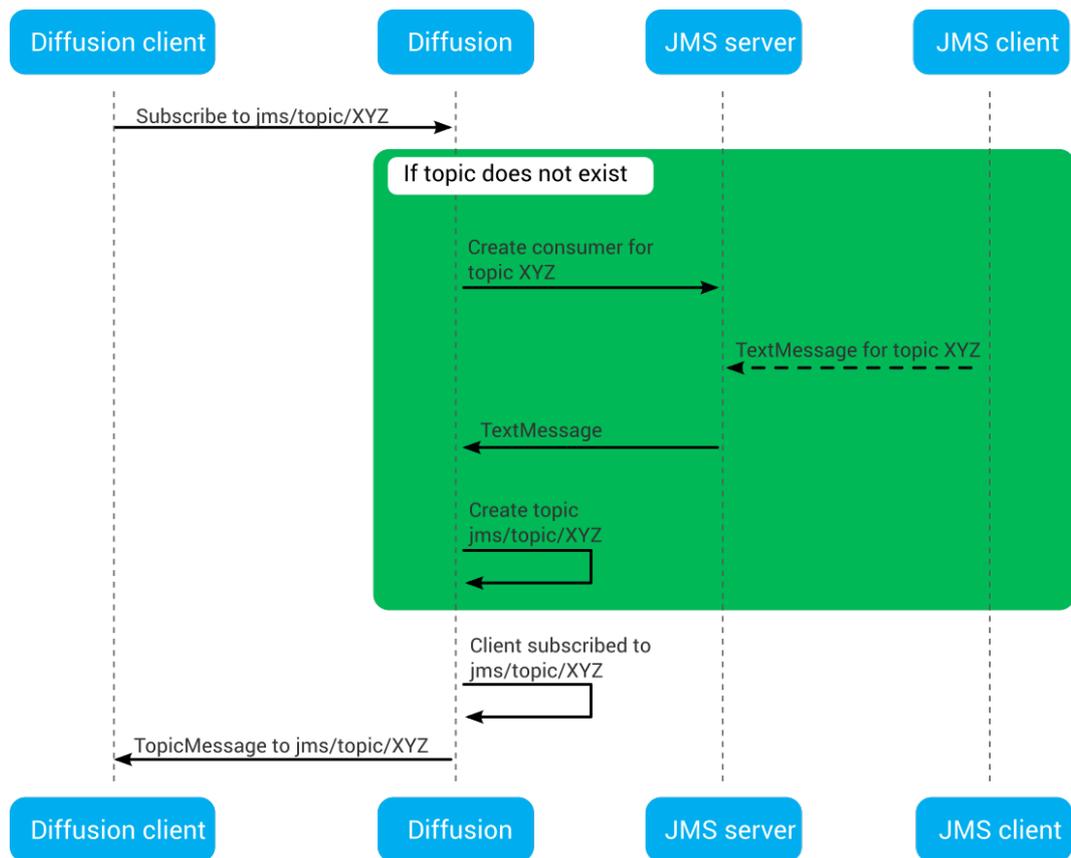


Figure 52: Subscription flow

Receiving messages from a JMS queue

The same process occurs for receiving messages from JMS queues, with the following differences:

- Other clients subscribing to the same JMS queue (either through Diffusion or directly using JMS) might receive the message instead of our client.
- All messages originating from JMS queues are initial topic load messages. Since a sequence of messages from a JMS queue are unlikely to always be delivered to the same client, the concept of delta messages does not readily apply and the full message state must be supplied every time.

Related concepts

DEPRECATED: [Legacy JMS adapter](#) on page 706

For backwards compatibility, Diffusion provides an older version of the JMS adapter, version 5.1. The version 5.1 JMS adapter is not compatible with the latest version of the JMS adapter.

DEPRECATED: [Configuring the legacy JMS adapter version 5.1](#) on page 708

Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Related tasks

[Configuring the JMS Adapter v5.1](#) on page 709

The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

Related reference

DEPRECATED: [Sending messages to JMS](#) on page 719

Diffusion clients can send messages to a JMS provider through the JMS adapter v5.1.

DEPRECATED: [Processing a request-reply message with a Diffusion client](#) on page 720

A common pattern among JMS solutions is for a client to receive a message from JMS and a reply is expected to be sent to a specific JMS topic or queue.

DEPRECATED: [Sending a request-reply message from a Diffusion client](#) on page 722

You can send a message from a Diffusion client into a JMS server with the expectation that a JMS client processes the message and sends a response back to the same Diffusion client.

DEPRECATED: [JMS adapter data flow examples](#) on page 716

The examples in this section show how data flows between the Diffusion server and a JMS provider.

[JMSAdapter51.xml](#) on page 712

This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

DEPRECATED: Sending messages to JMS

Diffusion clients can send messages to a JMS provider through the JMS adapter v5.1.

Note: The JMS Adapter v5.1 is now deprecated. Use the new JMS adapter instead. For more information, see [JMS](#) on page 124.

1. The Diffusion client sends a message to the topic path `jms/topic/XYZ`.
2. The JMS server receives an equivalent `TextMessage` on the XYZ topic.

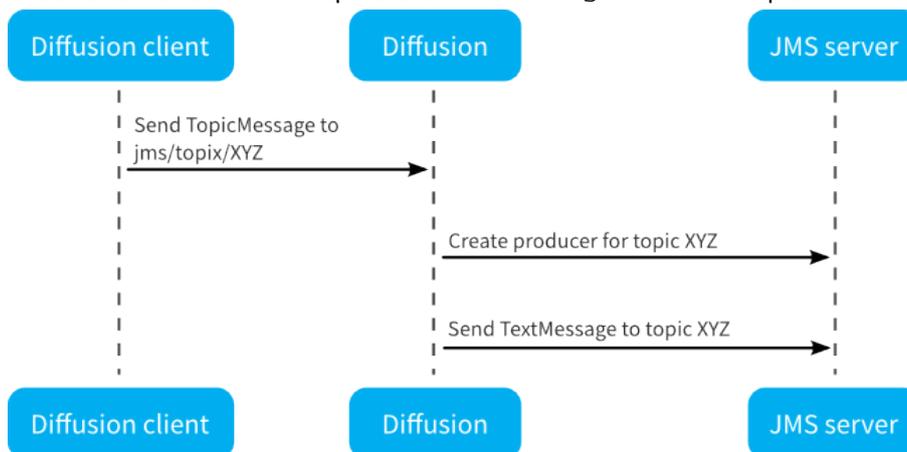


Figure 53: Sending flow from a Diffusion client to a JMS topic (or queue)

Unlike some Diffusion solutions, it is not necessary to subscribe to a Diffusion topic before sending it a message which targets a JMS destination.

Related concepts

DEPRECATED: [Legacy JMS adapter](#) on page 706

For backwards compatibility, Diffusion provides an older version of the JMS adapter, version 5.1. The version 5.1 JMS adapter is not compatible with the latest version of the JMS adapter.

DEPRECATED: [Configuring the legacy JMS adapter version 5.1](#) on page 708

Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Related tasks

[Configuring the JMS Adapter v5.1](#) on page 709

The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

Related reference

[DEPRECATED: Receiving data from JMS](#) on page 717

Diffusion clients can receive data from a JMS provider through the JMS adapter v5.1. A client can receive updates from a JMS topic or messages from a JMS queue.

[DEPRECATED: Processing a request-reply message with a Diffusion client](#) on page 720

A common pattern among JMS solutions is for a client to receive a message from JMS and a reply is expected to be sent to a specific JMS topic or queue.

[DEPRECATED: Sending a request-reply message from a Diffusion client](#) on page 722

You can send a message from a Diffusion client into a JMS server with the expectation that a JMS client processes the message and sends a response back to the same Diffusion client.

[DEPRECATED: JMS adapter data flow examples](#) on page 716

The examples in this section show how data flows between the Diffusion server and a JMS provider.

[JMSAdapter51.xml](#) on page 712

This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

[DEPRECATED: Processing a request-reply message with a Diffusion client](#)

A common pattern among JMS solutions is for a client to receive a message from JMS and a reply is expected to be sent to a specific JMS topic or queue.

Note: The JMS Adapter v5.1 is now deprecated. Use the new JMS adapter instead. For more information, see [JMS](#) on page 124.

Typically, the JMS publisher creates and sends a message with the `JMSReplyTo` header set to some other destination defined within the JMS server. This can be a topic, queue, or a temporary topic or temporary queue. The Diffusion client does not have to know the destination type as this is handled within the adapter.

1. The Diffusion client subscribes to topic `jms/queue/ABC`.
2. The JMS provider creates a temporary queue, `XYZ`, and subscribes to it.
3. The JMS provider sends a message to the queue `ABC` with `JMSReplyTo` set to the queue `XYZ`.
4. The Diffusion client receives a message on queue `jms/topic/ABC`, with `DiffusionReplyTo` set to `jms/reply/XYZ`.
5. The Diffusion client sends a response message to the queue `jms/reply/XYZ`.
6. The JMS provider receives a `TextMessage` on the temporary queue `XYZ`.

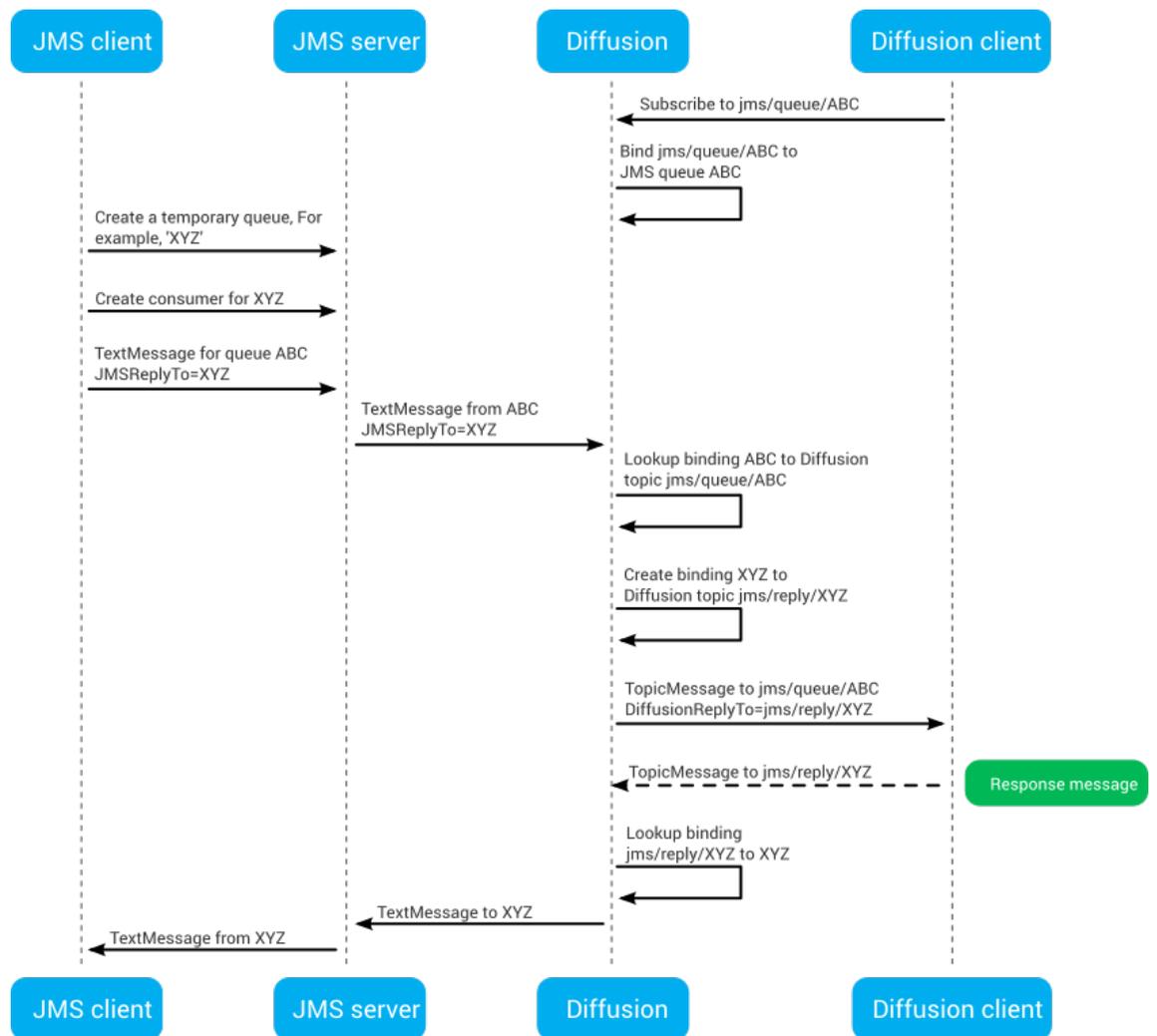


Figure 54: Request-reply initiated by a JMS client and serviced by a Diffusion client

Related concepts

[DEPRECATED: Legacy JMS adapter](#) on page 706

For backwards compatibility, Diffusion provides an older version of the JMS adapter, version 5.1. The version 5.1 JMS adapter is not compatible with the latest version of the JMS adapter.

[DEPRECATED: Configuring the legacy JMS adapter version 5.1](#) on page 708

Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Related tasks

[Configuring the JMS Adapter v5.1](#) on page 709

The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

Related reference

[DEPRECATED: Receiving data from JMS](#) on page 717

Diffusion clients can receive data from a JMS provider through the JMS adapter v5.1. A client can receive updates from a JMS topic or messages from a JMS queue.

[DEPRECATED: Sending messages to JMS](#) on page 719

Diffusion clients can send messages to a JMS provider through the JMS adapter v5.1.

[DEPRECATED: Sending a request-reply message from a Diffusion client](#) on page 722

You can send a message from a Diffusion client into a JMS server with the expectation that a JMS client processes the message and sends a response back to the same Diffusion client.

DEPRECATED: [JMS adapter data flow examples](#) on page 716

The examples in this section show how data flows between the Diffusion server and a JMS provider.

[JMSAdapter51.xml](#) on page 712

This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

DEPRECATED: [Sending a request-reply message from a Diffusion client](#)

You can send a message from a Diffusion client into a JMS server with the expectation that a JMS client processes the message and sends a response back to the same Diffusion client.

Note: The JMS Adapter v5.1 is now deprecated. Use the new JMS adapter instead. For more information, see [JMS](#) on page 124.

1. The JMS client subscribes to messages on the queue ABC.
2. The Diffusion client subscribes to `jms/tmp/queue/XYZ`. (Commonly, XYZ is a unique identifier).
3. The Diffusion client sends a request message to `jms/queue/ABC` with the `DiffusionReplyTo` header set with the value `jms/tmp/queue/XYZ`.
4. The JMS client receives the request message on queue ABC, with the `JMSReplyTo` header set to queue DEF.
5. The JMS client sends a reply to queue DEF.
6. The Diffusion client receives the reply on topic `jms/tmp/queue/XYZ`.

Note: The return Diffusion topic can be any topic, so it is not necessary that the originating Diffusion client receives the reply – it can be any client listening for messages on that topic.

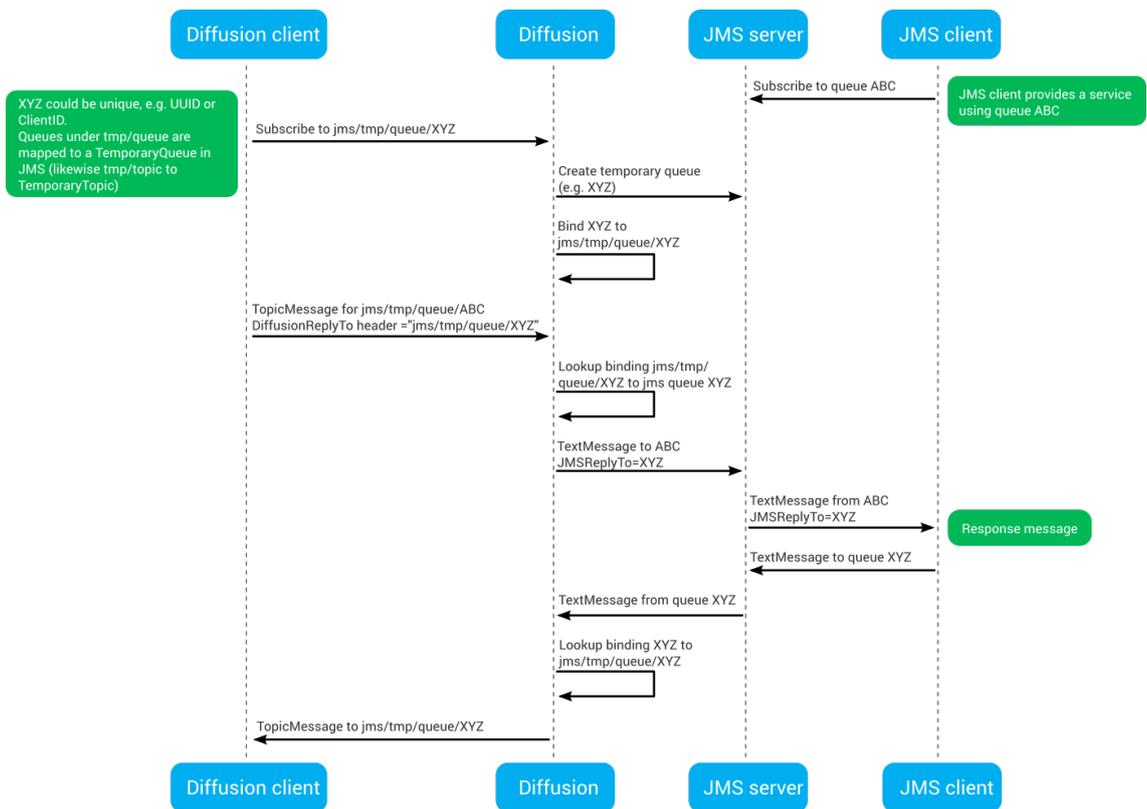


Figure 55: Request-reply initiated by a Diffusion client and serviced by a JMS client

Related concepts

DEPRECATED: [Legacy JMS adapter](#) on page 706

For backwards compatibility, Diffusion provides an older version of the JMS adapter, version 5.1. The version 5.1 JMS adapter is not compatible with the latest version of the JMS adapter.

DEPRECATED: [Configuring the legacy JMS adapter version 5.1](#) on page 708

Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Related tasks

[Configuring the JMS Adapter v5.1](#) on page 709

The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

Related reference

DEPRECATED: [Receiving data from JMS](#) on page 717

Diffusion clients can receive data from a JMS provider through the JMS adapter v5.1. A client can receive updates from a JMS topic or messages from a JMS queue.

DEPRECATED: [Sending messages to JMS](#) on page 719

Diffusion clients can send messages to a JMS provider through the JMS adapter v5.1.

DEPRECATED: [Processing a request-reply message with a Diffusion client](#) on page 720

A common pattern among JMS solutions is for a client to receive a message from JMS and a reply is expected to be sent to a specific JMS topic or queue.

DEPRECATED: [JMS adapter data flow examples](#) on page 716

The examples in this section show how data flows between the Diffusion server and a JMS provider.

[JMSAdapter51.xml](#) on page 712

This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Network security

This section describes how to deploy network security, which can be used in conjunction with data security.

Secure clients

Diffusion clients can connect to your solution using TLS or SSL. The secure connection can terminate at your load balancer or at your Diffusion server. Terminating the TLS/SSL at the load balancer reduces CPU cost on your Diffusion servers.

The following SSL and TLS versions are supported by default:

- SSLv2Hello
- TLSv1
- TLSv1.1
- TLSv1.2

You can use the system property `diffusion.tls.protocols` with the JVM that runs the Diffusion server, a Java client or an Android client to provide a different list of secure protocols to use.

The following cipher suites are supported by default:

- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA

You can use the system property `https.cipherSuites` with the JVM that runs the Diffusion server, a Java client or an Android client to provide a different list of cipher suites to use.

Session tokens

Unified API clients use session tokens to authenticate when reconnecting to their existing session. To protect the credentials supplied in the original connection request and the returned session token, ensure that the client uses a secure transport to communicate with the Diffusion server. For example, WSS.

Session tokens are generated by using `java.security.SecureRandom` with the default algorithm supplied by the Java environment used to run the Diffusion server. Each token is a 24-character string encoded in base-64, representing 18 bytes (144 bits) of random data.

A new session token is generated when a client connects to the Diffusion server, is authenticated, and creates a session. The server returns the session token to the client in the connection response. The client library keeps the session token in memory. If the client connection is lost, the client attempts to reconnect and supplies the session token. The server is configured with a reconnection timeout. If the Diffusion server detects the loss of the client connection and the client fails to reconnect to the Diffusion server before the reconnection timeout has elapsed, the Diffusion server closes the session and the session token is no longer valid. If the client reconnects before the reconnection timeout has elapsed, the Diffusion server accepts the new connection using the session token is used as proof of authentication.

Web server configuration

The web server can be configured in your test environment to allow you to deploy and undeploy DAR files by using a web service. By default this capability is not enabled.

For security, if you choose to enable this web service in your production environment, you must restrict access to the `diffusion-url/deploy` URL by other means. For example, by setting up restrictions in your firewall.

To configure the web server, use the `WebServer.xml` file. For more information, see [WebServer.xml](#) on page 617. An example of this file is provided in the `/etc` directory of the Diffusion installation. The XSD is provided in the `/xsd` directory of the Diffusion installation.

Connector configuration

If secure connections are required, Diffusion connectors must be configured to support HTTPS, WSS, DPTS, or a combination of these transports. Any connector can accept secure connections. A connector does not have to be dedicated to only secure connections. To enable secure connections a keystore entry is required in the connector configuration. This informs the connector that it is enabled for secure connections. If HTTPS is required, a keystore section and a web-server entry are also required, even for secure Diffusion clients.

To configure the connectors, use the `Connectors.xml` file. For more information, see [Connectors.xml](#) on page 583. An example of this file is provided in the `/etc` directory of the Diffusion installation. The XSD is provided in the `/xsd` directory of the Diffusion installation.

Keystores

The default Diffusion installation includes a sample keystore containing a self-signed certificate. This is suitable for development. The certificate will not be trusted by browsers and other clients without additional configuration. If you use TLS in production, you must create a new keystore, using a certificate obtained from a certificate authority.

The following steps use the Java Keytool to create a keystore. The steps can vary depending on your certificate authority. For more information, refer to your certificate authority's documentation.

1. Generate a key and place it in your keystore.

```
keytool -genkeypair -alias my_alias -keyalg RSA -  
keystore keystore_name -keysize
```

2. Generate a CSR file.

```
keytool -certreq -keyalg RSA -alias my_alias -file certreq.csr -  
keystore keystore_name
```

3. Send the CSR file to your certificate authority.
4. Receive the signed certificate from your certificate authority.
5. Install any intermediate certificates that you require.

```
keytool -import -trustcacerts -alias intermediate_alias -  
keystore keystore_name -file intermediate_certificate_file.crt
```

6. Install your own certificate. Use the same alias as when you generated the key and the signing request.

```
keytool -import -trustcacerts -alias my_alias -  
keystore keystore_name -file certificate_file.crt
```

Provided keystores

The `etc` directory of the Diffusion directory contains the following keystores:

sample.keystore

This keystore is an example keystore that contains a self-signed certificate. In production, we recommend you create your own keystore that contains a certificate signed by a certificate authority.

licence.keystore

This keystore contains the public key used for the Diffusion license file. Do not edit or delete this keystore. Diffusion requires this keystore to verify your Diffusion license.

Related information

<http://www.networking4all.com/en/support/ssl+certificates/manuals/java/java+based+webserver/keytool+commands/>

Going to production

When going to production with Diffusion review this information for recommendations on preparing for a successful production deployment.

The advice in this section is not an exhaustive list of steps to take when getting ready to take your Diffusion solution into production. You might have additional requirements based on your solution. Push Technology provides Professional Services that can work with you to advise on a pre-production testing strategy specific to your requirements. Email consulting@pushtechnology.com for more information.

Pre-production testing

The most important part of taking your solution into production is to ensure that you fully test it under as wide a range of expected conditions as possible.

Setting up your test environment

Ensure that the environment you set up to test your solution is as close as possible to the production environment you intend to deploy.

About this task

There are many benefits to creating a test environment that is the same as your production environment:

- It enables you to do regression testing when you change the version or configuration of any of the components in your solution.
- It enables you to test your solution's performance under different levels of stress and load.
- It provides a controlled environment where you can reproduce any issues that you encounter in your production system.
- It provides an environment where you can capture runtime data that cannot easily be captured in production without affecting the behavior or performance of the production system.

To create a test environment that closely reflects your production environment, consider taking the following steps:

Procedure

1. Use the same hardware as you intend to deploy your production solution on.

Consider the following aspects:

- Hardware specifications
- Operating system version and patch version

For more information, see [Requirements](#).

2. Use the same network specifications as you intend to use in your production solution.

Consider the following aspects:

- Connection reliability

Note: This is especially important when testing mobile applications. The behavior of your mobile client can be different depending on whether the client connects through WiFi or 3G, for example.

- Speed

3. Use the same JDK version as you intend to use to run your Diffusion server.

Consider the following aspects:

- The JVM version
- Any tuning parameters you intend to use

For more information, see [Requirements](#).

4. Include third-party components that will be used in your production solution.

Consider your use of the following components:

- Load balancers

For more information, see [Load balancers](#) on page 119 and [Common issues when using a load balancer](#) on page 647.

- Web servers

For more information, see [Web servers](#) on page 120 and [Web servers](#) on page 648.

- Push notification networks

For more information, see [Push notification networks](#) on page 123 and [Push Notification Bridge](#) on page 660.

- JMS servers

For more information, see [JMS](#) on page 124 and [JMS adapter](#) on page 677.

There are special considerations for using these third-party components with Diffusion. Ensure that you have fully reviewed the linked documentation for any of the components you are using.

5. Ensure your Diffusion servers use the same version and configuration as in your production solution.

Consider the following aspects:

- Diffusion server version
- Diffusion server configuration
- Diffusion server security configuration

6. Include all the components you have developed for use in your production system.

Consider your use of the following components:

- Clients

Both those within your organization and those used by your customers.

- Publishers

- Server-side components

Understanding production usage conditions

Consider the flow of data and the actions of clients in your Diffusion server. Pre-production testing that closely models the usage you expect to see in production is most useful in understanding how your solution will respond in production.

The following sections contain some of the questions to consider when deciding how to test your solution before going to production. For each of these questions consider both average use values and edge case values.

Client connections

- How many clients do you expect to attempt to connect simultaneously?
- How many clients do you expect to be connected concurrently?
- Do you have session replication enabled and, if so, in a failover situation do you expect all of your concurrently connected clients to attempt reconnect at the same time?
- How long is a client connection expected to last?
- What is the expected geographic distribution of client connections?
- How does your load balancer decide how to distribute incoming client connections?
- How are your expected client connections distributed by platform or API?
- How are your incoming client connections authenticated?

Topics

- At what frequency do you expect to create topics?
- How many topics do you expect to create at the same time?
- At what frequency do you expect to delete topics?
- How many topics do you expect to delete at the same time?
- How many topics do you expect your clients to be subscribed to?
- How many topics do you expect your clients to subscribe to in a single action?

Topic updating

- How many topics is a given client expected to update?
- How frequently do you expect topics to be updated?
- How many topics do you expect to send updates to at the same time?
- How many topics do you expect a given client to send updates to at the same time?
- How big do you expect the data in your topic updates to be?

Other client actions

- How many client authentication requests is a given client expected to handle?
- How many messages sent to a topic path is a given client expected to handle?
- How many messages sent directly to the client is a given client expected to receive?
- How many messages is a given client expected to send to a topic path?
- How many messages is a given client expected to send directly to another client?
- How often do you expect clients to manage other clients?
- How many clients do you expect a given client to manage?

How to create production usage conditions in your test environment

You can create production usage conditions in your test environment by either recording live production usage and playing it back or by simulating production usage.

Recording production conditions

Recording production conditions involves inserting components within your production system to track specific actions. For example:

- A recording tool upstream of the Diffusion server that records the data stream being fed in to Diffusion topics
- A recording tool at your load balancer to record incoming connections, where they come from, when they connect, and how long they remain connected.

After this data has been captured in production test tools use the data to replay or simulate identical conditions in your test environment.

The record and playback approach has the following advantages:

- The data and client actions have occurred in production and reflect realistic production conditions.

The record and playback approach has the following disadvantages:

- You cannot use the recorded data to test conditions that have not occurred in your production environment, but that you might expect to occur.
- You must have an existing production environment to capture data from.
- You must develop the tools to capture and store production conditions. Introducing these components to your production system might effect its behavior.
- You must develop the tools to replay production conditions in your test environment.

Simulating production conditions

Simulating production conditions involves developing tools or test harnesses that generate data or client behavior in your test environment.

The simulation approach has the following advantages:

- You can test a wider range of conditions than those that have occurred in production.

The simulation approach has the following disadvantages:

- There is a risk that the simulation might not create realistic production conditions.
- You must develop the tools to simulate the conditions you want to test.

Using live production data

You can create production conditions in your data by using the same data stream as the production environment uses to feed into your test environment.

The live production data approach has the following advantages:

- The data stream being fed in to Diffusion is real.
- You do not need to create tools to record and playback or to simulate production data.

The live production data approach has the following disadvantages:

- Depending on the type of client information in the production data and your data protection policies and legal requirements, you might not be permitted to use live production data in a test environment.
- Depending on the type of client information in the production data and your data protection policies and legal requirements, you might not be permitted to send certain diagnostics to Push Technology when requesting support.

- You must ensure that nothing in your test environment can affect either the production data or the production environment.
- You must have an existing production environment to use data from.
- If you want to test specific data conditions, you are restricted to doing so at the times when these conditions occur.

Using live production traffic

You can simulate production conditions in your client traffic by duplicating client requests coming into your live production environment in your test environment and by suppressing the responses made by the test server from reaching the production client.

The live production traffic approach has the following advantages:

- The client requests to Diffusion are real.
- You do not need to create tools to record and playback or to simulate production traffic.

The live production traffic approach has the following disadvantages:

- Depending on the type of client information in the traffic and your data protection policies and legal requirements, you might not be permitted to use live production traffic in a test environment.
- Depending on the type of client information in the production traffic and your data protection policies and legal requirements, you might not be permitted to send certain diagnostics to Push Technology when requesting support.
- You must have an existing production environment to use the traffic from.
- You must ensure that responses from your test environment do not reach the client.
- Because the responses from the test environment do not reach the production clients, who instead receive responses from the production environment, the behavior does not accurately reflect server-client interactions.
- If you want to test specific traffic conditions, you are restricted to doing so at the times when these conditions occur.

These techniques can be used separately or together to give the fullest range of test conditions.

Both involve the development of custom tooling to create the required conditions. Push Technology provides Professional Services that can work with you to create these tools. Email consulting@pushtechology.com for more information.

Types of testing

Consider performing the following types of testing before taking your solution into production.

Component testing

Before setting up your test environment, test the clients and other components that you have developed. For each component, consider doing the following testing:

- Unit testing with a high level of code coverage
- End-to-end testing
- Performance testing
- Stress testing
- Usability and accessibility testing, if the component is customer-facing.

For more information, see [Testing](#) on page 519

Smoke testing

On first setting up your test environment, smoke test your solution to ensure that all basic expected function works before proceeding to more in-depth testing.

For more information, see [Smoke Testing on Wikipedia](#)

Regression testing

If you have an existing Diffusion solution in production and are updating one or more components or their configuration, perform regression testing in your test environment to ensure that the behavior of your solution has not changed in unintended ways before updating your production environment.

For more information, see [Regression Testing on Wikipedia](#)

Load testing

Ensure that you test your Diffusion solution at peak expected load to discover how your solution handles these conditions. This load can be client connections, topics, topic updates, and combinations of load types.

For more information, see [Load Testing on Wikipedia](#)

Soak testing

Most Diffusion solutions are expected to run continuously under varying load. Ensure that you test how your solution behaves when it is left in operation for a long time, for example, 24 hours. Long test runs can uncover potential resource leaks, long garbage collections, or previously unforeseen timeouts.

For more information, see [Soak Testing on Wikipedia](#)

Failover and recovery testing

If your solution has failover or replication configured on your Diffusion servers, test that these work as you expect when one of the Diffusion becomes unavailable. For resiliency of your whole solution, other components – for example, load balancers – can be configured to failover or provide redundancy. Ensure that these measures work as you expect.

For more information, see [Configuring the Diffusion server to use replication](#) on page 611 and [Using load balancers for resilience](#) on page 647

Penetration testing

Diffusion provides mechanisms to secure which ports clients can connect to your Diffusion server on, what actions those clients can take, and what data they can view or update. You also can use load balancers and firewalls to secure your solution.

However, security flaws can occur in any system. Many companies offer a penetration testing service that can help uncover any vulnerabilities in your solution. If you do not have the resource or knowledge to perform penetration testing on your solution, we recommend that you use a third-party penetration testing company.

For more information, see [Penetration Testing on Wikipedia](#)

Testing your security

Your Diffusion solution is made up of multiple components. Ensure that you consider and test for potential security problems in all your components and in their interactions.

It is important to design your solution for security before you even begin any development or configuration. For more information about designing a secure solution, see [Design Guide](#) on page 44.

Note: Many companies offer a penetration testing service that can help uncover any vulnerabilities in your solution. If you do not have the resource or knowledge to perform penetration testing on your solution, we recommend that you use a third-party penetration testing company.

Consider these aspects of security for your solution.

URL spaces and ports exposed by your load balancer

What routes does your solution offer to connections from outside?

For more information, see [Load balancers](#) on page 642.

Connectors

What ports allow connections to the Diffusion server? What kind of connections are these ports configured to allow?

For more information, see [Configuring connectors](#) on page 581.

Users and roles on your Diffusion server

How are connections to the Diffusion server authenticated? What roles and permissions are assigned to authenticated connections? How are different parts of your topic tree secured?

For more information, see [Role-based authorization](#) on page 130.

Console

Are connections from outside your organization permitted to access the Diffusion console? Which users are assigned the permission to access the console?

For more information, see [Diffusion monitoring console](#) on page 759.

Introspector

Are connections from outside your organization permitted to access the Introspector? Which users are assigned the permission to access Diffusion using the Introspector?

For more information, see [Diffusion monitoring console](#) on page 759.

Tools you can use in your pre-production testing

There are many available tools that are useful when doing pre-production testing of your solution.

Amazon Web Services (AWS)

Use AWS to host large numbers of test clients that connect to your test environment for capacity and load testing. Using a cloud provider enables you to scale up your testing without being constrained by how much hardware resource you have in your organization.

Amazon Web Services is one of many cloud providers that you can choose between for your load and capacity testing.

For more information, see <https://aws.amazon.com/dev-test/>

Eclipse Memory Analyser Tool (MAT)

Use Eclipse MAT to analyze how your Diffusion server memory behaves under different usage conditions. You can also use this tool to analyze the memory behavior of any Java clients that you use in your solution.

For more information, see <http://www.eclipse.org/mat/>

VisualVM

VisualVM is a Java monitoring tool that you can use to monitor the behavior of the Diffusion server and other Java-based components in your solution.

For more information, see <https://visualvm.java.net/>

VisualVM also provides the ability to view the MBeans that the Diffusion server registers with the JMX service. These MBeans provide statistics and information about many of the primary features of the Diffusion server.

For more information, see [JMX](#) on page 735

Java Flight Recorder and Java Mission Control

These tools provide the capability to capture low-level JVM metrics during the test cycle. Java Flight Recorder is built into the Oracle JDK. Java Mission Control enables you to analyse the data collected by the Flight Recorder.

For more information, see [Java Mission Control documentation](#)

Diffusion monitoring console

Use the Diffusion monitoring console to validate, in real time, the metrics presented by the Diffusion server.

For more information, see [Diffusion monitoring console](#) on page 759

Diffusion JavaScript test client

Use the JavaScript test client, which is available from the Diffusion landing page at `http://localhost:8080` to perform basic feature testing and smoke testing against your test servers.

DEPRECATED: Diffusion Introspector Eclipse plugin

Use the Introspector to inspect Diffusion internals from within Eclipse.

For more information, see [DEPRECATED: Introspector](#) on page 770

Diffusion benchmarking suite

Push Technology provide a suite of benchmarks that you can use to test the behavior of the Diffusion server on your hardware and with your configuration.

For more information, see <https://github.com/pushtechnology/diffusion-benchmark-suite>

Planning for production

The key to a successful production deployment is planning and preparation.

Consider the following questions when planning for production deployment:

Hard launch or soft launch?

In a hard launch, your solution is rolled out to all of your users at the same time. In a soft launch, your solution is rolled out to only a select group of users.

The advantage of a soft launch is that it enables you to trial your new solution with a subset of your users and discover any remaining issues before rolling out to your whole user base.

Will your users experience any down-time?

How will your deployment affect existing users? Will their clients experience a disconnection? Will the deployment of your new solution force them to upgrade their client version before they can continue to use your solution?

Understand what your users will experience during your deployment and what experience you want them to have.

When are you going to roll out to production?

Select a time that fits best with your business. Consider when you have the most users, when you have certain events for which your system needs to be up, and when your team are available to support and troubleshoot the deployment.

Who do you need to notify in advance?

Do you need to notify your users of upcoming down-time? Do you need to notify your user of actions they must take? Do you have any third parties that provide data or services who need to be notified?

How are you deploying your solution to production?

Are you rolling out all of the components of your solution or just changing some of them? What order are you deploying your components in? Are you going to automate all or some of the deployment process?

What is your roll back plan if something unforeseen happens?

Even with the best testing and planning, problems can occur in a production environment. Developing a strategy in advance for handling problems ensures that you can react quickly if problems occur.

Prepare a go-live checklist

After you have considered all aspects of your deployment, we recommend that you create a go-live checklist detailing all of the tasks necessary across your organization in order to go live.

Deploying to your production environment

For the best results, consider automating deployment of your components and configuration.

Automated deployment to your test environment enables you to quickly iterate your development and roll out new changes into testing. Removing the overheads of setting up a test environment by automating the process, gives your team more time to perform testing.

Automated deployment to your production environment helps reduce the risk of human error. By automating all the steps required to deploy your solution to production, you can easily test your deployment process. Automated deployment is quicker than manual deployment and can reduce the amount of down-time that a deployment might cause. Testing your automated deployment process gives you the chance to measure this down-time duration. You can use this information to appropriately set your service-level agreements.

Managing and monitoring your running Diffusion server

This section discusses how to manage your Diffusion server and system as a whole.

We recommend that you actively monitor the health of your Diffusion server to pre-empt failures and to minimize unplanned downtime.

You can monitor your Diffusion server using the tools listed in this section.

JMX

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

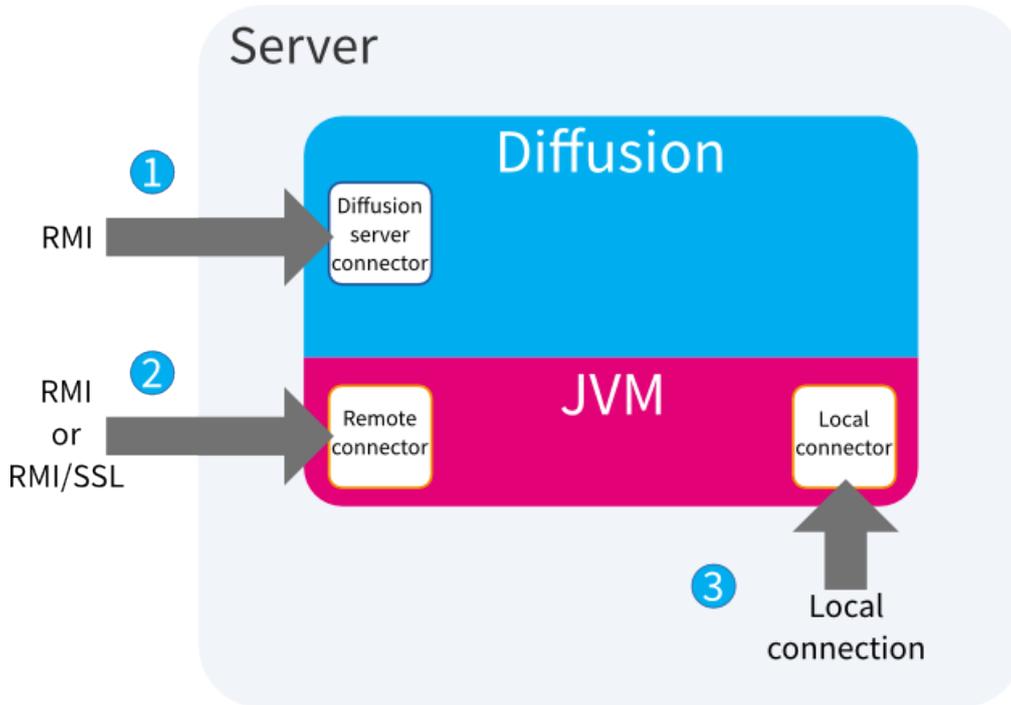


Figure 56: Connecting to Diffusion JMX

The following methods of connecting to Diffusion JMX are available:

- 1. Recommended:** Through the RMI JMX connector server provided by the Diffusion server.

This feature is integrated with Diffusion security, enabling you to use roles and permissions to control access to the MBeans. However, this connection does not use SSL.

For more information, see [Configuring the Diffusion JMX connector server](#) on page 601.

- 2.** Through the RMI JMX connector server provided by JVM that runs Diffusion.

You can use SSL to make a secure connection. However, the JVM does not use Diffusion security. You must add additional configuration to your JVM to control access to the MBeans.

For more information, see [Configuring a remote JMX server connector](#) on page 602.

- 3.** Through the local JMX connector server provided by JVM that runs Diffusion.

You make this connection from the server that Diffusion runs on. However, the JVM does not use Diffusion security. You must add additional configuration to your JVM to control access to the MBeans.

For more information, see [Configuring a local JMX connector server](#) on page 603.

Related concepts

DEPRECATED: [Introspector](#) on page 770

An introduction to the Introspector Eclipse plugin.

Related tasks

[Configuring the Diffusion JMX connector server](#) on page 601

Connect to JMX through the Diffusion connector server. This connector server is integrated with the Diffusion server and enables you to use role-based access control to define how connecting users can use the MBeans.

[Configuring a local JMX connector server](#) on page 603

Connect to JMX through a local connector to the JVM that runs the Diffusion. This connector is not integrated with the Diffusion server security and you must configure additional security in the JVM.

[Configuring a remote JMX server connector](#) on page 602

Connect to JMX through a remote connector to the JVM that runs the Diffusion. This connector is not integrated with the Diffusion server security and you must configure additional security in the JVM.

Related reference

[Statistics](#) on page 754

Diffusion provides statistics about the server, publishers, clients, and topics.

[Diffusion monitoring console](#) on page 759

A web console for monitoring the Diffusion server.

[Logging](#) on page 784

Diffusion uses the Simple Logging Facade for Java (SLF4J) API to log messages from the Diffusion server or from publishers running on the Diffusion server. SLF4J separates the logging of messages in the Diffusion server from the logging framework. This separation enables you to configure an independent back-end implementation to format and write out the log messages.

[Integration with Splunk](#) on page 1049

How to achieve basic integration between Diffusion and the Splunk™ analysis and monitoring application

[Management.xml](#) on page 604

This file specifies the schema for the management properties that enable JMX access over an RMI JMXConnectorServer.

Using Java VisualVM

You can manage Diffusion using the JMX system management console Java VisualVM.

Java VisualVM is usually installed with JDK's but can be downloaded from <https://visualvm.dev.java.net/>.

Connecting to the Diffusion connector server

1. Start Java VisualVM.
2. Right-click on the **Remote** section of the **Applications** panel and select **Add Remote Host**.
3. In the **Host name** field, type the host name or IP address of the server where Diffusion is running. Click **OK**.
4. In the **Applications** panel, right-click on the name of the server where Diffusion is running. Select **Add JMX Connection**.
5. In the **Connection** field, enter the host name and RMI registry port for the Diffusion server.
6. Select **Use security credentials** and enter the username and password of a principal that you have configured to be able to use JMX. For more information, see [Configuring the Diffusion JMX connector server](#) on page 601. Click **OK**.

Information about the Diffusion process is displayed in the main panel.

Connecting to the JVM remote connector server

Note: We recommend you use the Diffusion connector server to access the JMX service.

1. Start Java VisualVM.

2. Right-click on the **Remote** section of the **Applications** panel and select **Add Remote Host**.
3. In the **Host name** field, type the host name or IP address of the server where Diffusion is running. Click **OK**.
4. In the **Applications** panel, right-click on the name of the server where Diffusion is running. Select **Add JMX Connection**.
5. In the **Connection** field, enter the host name and RMI registry port for the Diffusion server.
6. Select **Use security credentials** and enter the username and password of a user that you have configured in the JVM. For more information, see [Configuring a remote JMX server connector](#) on page 602. Click **OK**.

Information about the Diffusion process is displayed in the main panel.

Connecting to the JVM local connector server

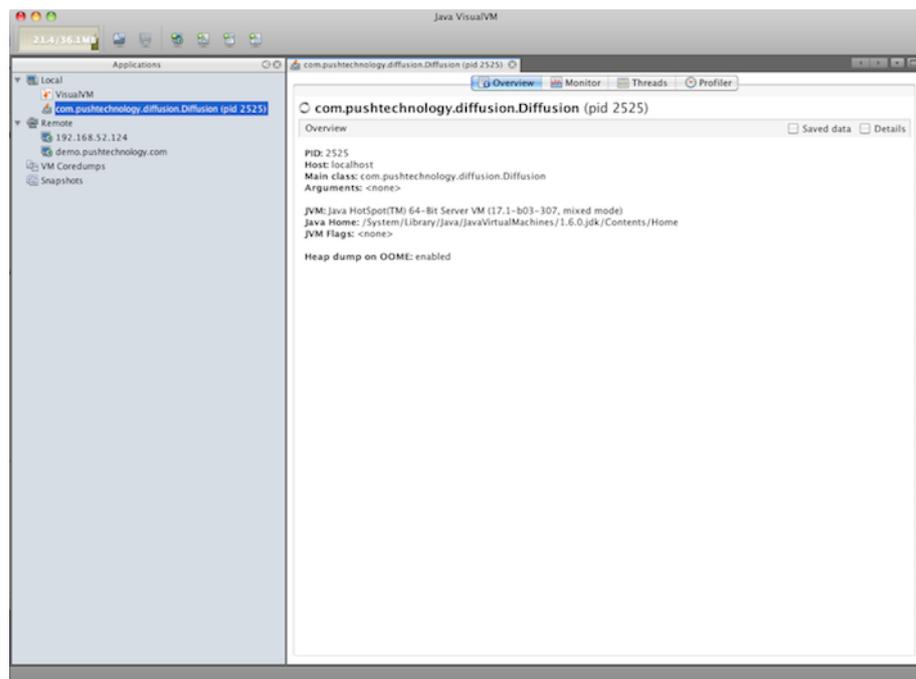


Figure 57: Java VisualVM: Overview tab

Note: We recommend you use the Diffusion connector server to access the JMX service.

1. Start Java VisualVM.
2. From the **Local** section of the **Applications** panel, select the Diffusion process, `com.pushtechnology.diffusion.Diffusion`.
3. Right-click `com.pushtechnology.diffusion.Diffusion` and select **Open**.

Information about the Diffusion process is displayed in the main panel.

Once connected to JMX, several aspects of the system are available to monitor and tune. For more information, see the Java VisualVM documentation: <http://visualvm.java.net/docindex.html>.

Related tasks

[Configuring the Diffusion JMX connector server](#) on page 601

Connect to JMX through the Diffusion connector server. This connector server is integrated with the Diffusion server and enables you to use role-based access control to define how connecting users can use the MBeans.

[Configuring a local JMX connector server](#) on page 603

Connect to JMX through a local connector to the JVM that runs the Diffusion. This connector is not integrated with the Diffusion server security and you must configure additional security in the JVM.

[Configuring a remote JMX server connector](#) on page 602

Connect to JMX through a remote connector to the JVM that runs the Diffusion. This connector is not integrated with the Diffusion server security and you must configure additional security in the JVM.

Using JConsole

You can manage Diffusion using the JMX system management console JConsole.

Connecting to the Diffusion connector server



Figure 58: JConsole New Connection dialog: Remote Process

In the **Remote Process** section of JConsole's **New Connection** dialog, enter the following information:

- The host name and RMI registry port of the Diffusion connector server. The default RMI registry port is 1099.
- The username and password of a principal that you have configured to be able to use MBeans. For more information, see [Configuring the Diffusion JMX connector server](#) on page 601



Figure 59: JConsole New Connection dialog: Remote Process

Note: We recommend you use the Diffusion connector server to access the JMX service.

In the **Remote Process** section of JConsole's **New Connection** dialog, enter the following information:

- The host name and RMI port of the Diffusion connector server. The default port is 1099.
- A username and password that you have configured in the JVM to be able to connect to the JMX service. For more information, see [Configuring a remote JMX server connector](#) on page 602

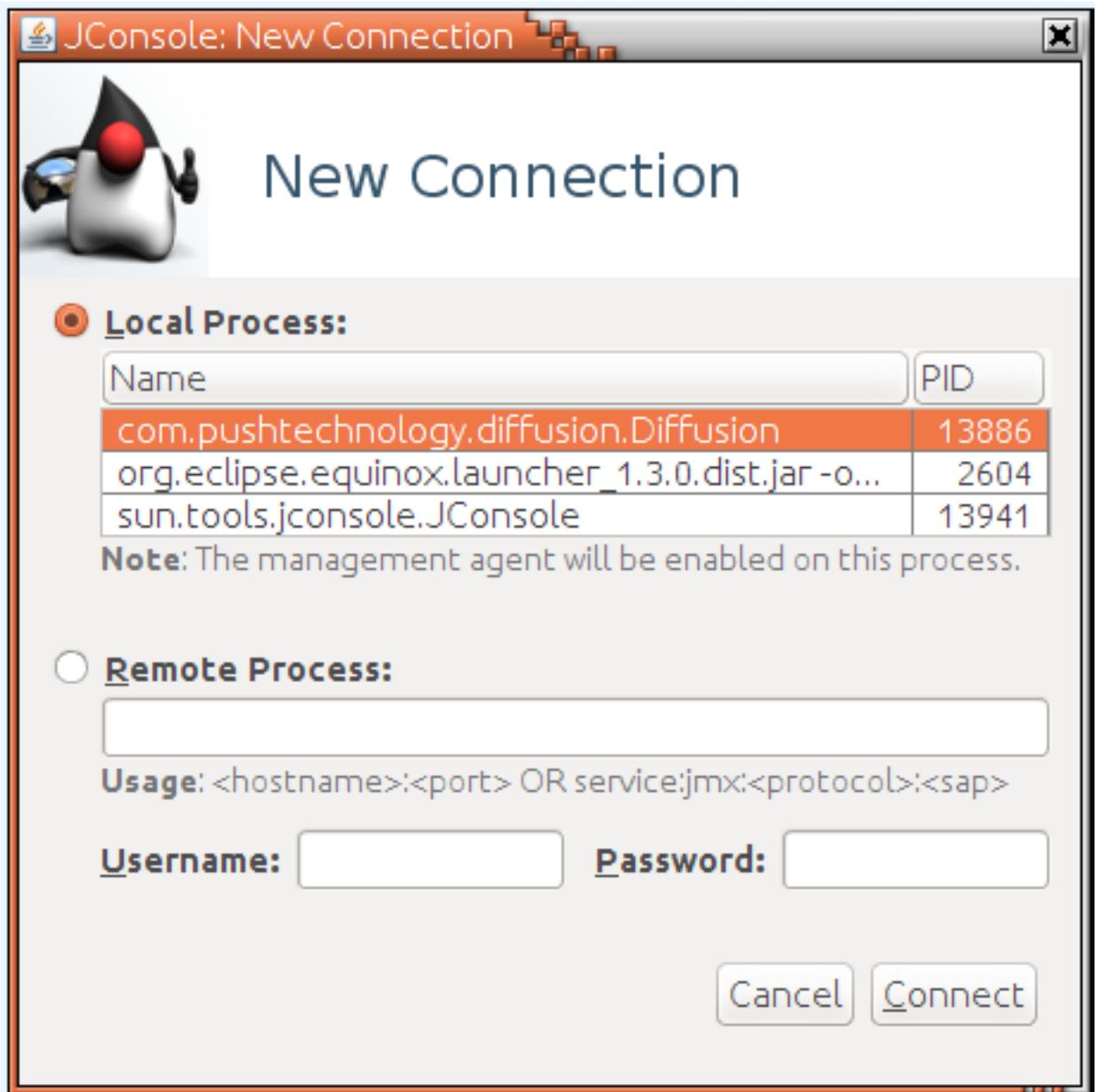


Figure 60: JConsole New Connection dialog: Local Process

Note: We recommend you use the Diffusion connector server to access the JMX service.

In the **Local Process** section of JConsole's **New Connection** dialog, select the Diffusion process, `com.pushtechnology.diffusion.Diffusion`.

Once connected to JMX, several aspects of the system are available to monitor and tune. For more information, see the JConsole documentation: <https://docs.oracle.com/javase/8/technotes/guides/management/jconsole.html>.

Related tasks

[Configuring the Diffusion JMX connector server](#) on page 601

Connect to JMX through the Diffusion connector server. This connector server is integrated with the Diffusion server and enables you to use role-based access control to define how connecting users can use the MBeans.

[Configuring a local JMX connector server](#) on page 603

Connect to JMX through a local connector to the JVM that runs the Diffusion. This connector is not integrated with the Diffusion server security and you must configure additional security in the JVM.

[Configuring a remote JMX server connector](#) on page 602

Connect to JMX through a remote connector to the JVM that runs the Diffusion. This connector is not integrated with the Diffusion server security and you must configure additional security in the JVM.

Detecting deadlocks with JConsole

To check if your publisher is experiencing a deadlock, you can use JConsole to inspect the threads.

Procedure

1. Connect JConsole to the JMX service on the Diffusion server.
For more information, see [Using JConsole](#) on page 738.
2. Go to the **Threads** tab.
This tab provides information about thread use: number of threads, a list of active threads, and details for a selected thread.
3. On the **Threads** tab, click the **Detect Deadlock** button.
If deadlocks are present, new tabs open next to the **Threads** tab. You can use the information in these tabs to diagnose any deadlocks.

What to do next

For more information, see <http://docs.oracle.com/javase/7/docs/technotes/guides/management/jconsole.html>.

MBeans

Diffusion registers MBeans with the JMX service for many of its principal features.

Annotations on each of the MBeans employed are used to produce the following pages in this manual as well as feeding JMX clients with descriptive information. MBeans, attributes and operations have descriptions; operation arguments have names; operations also have JMX impact information.

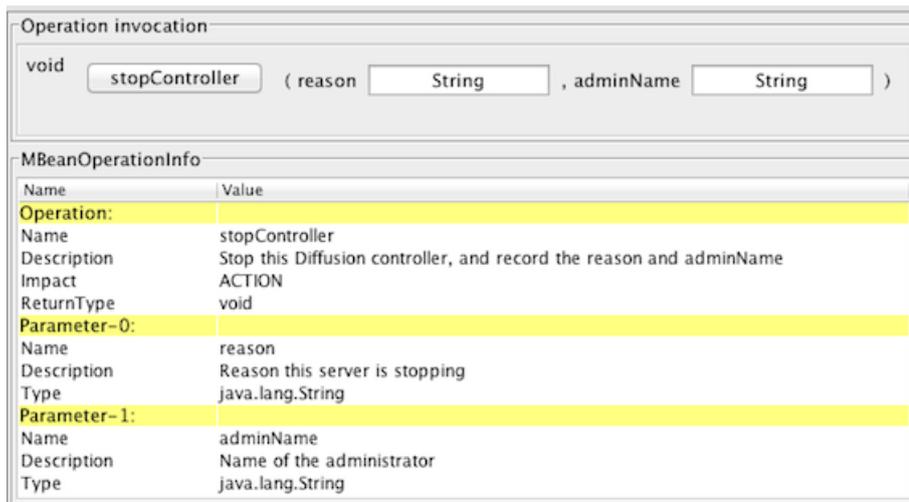


Figure 61: The server MBean stopController operation showing in JConsole

AggregateStatistics

Interface for a specific StatisticsCollector

Object name format

The objectName for MBeans of this type is of the following form:

`com.pushtechnology.diffusion:type=AggregateStatistics,name="name",server="server_`

Attributes

Name	Type	Read/Write	Description
instanceStatisticsEnabled	boolean	read-write	Whether statistics collection is enabled or not.
managementName	String	read	A name describing the set of statistics being aggregated.

AuthorisationManager

Management interface to the optional AuthorisationManager

Object name format

The objectName for MBeans of this type is of the following form:

`com.pushtechnology.diffusion:type=AuthorisationManager,server="server_name"`

Attributes

Name	Type	Read/Write	Description
connections	int	read	Number of connections authorized
connectionsDenied	int	read	Number of connections denied authorization
fetches	int	read	Number of fetches authorized
fetchesDenied	int	read	Number of fetches denied authorization

Name	Type	Read/Write	Description
handlerClassName	String	read	Class name of any configured AuthorisationHandler
hasHandler	boolean	read	True if this server has an AuthorisationHandler configured
subscriptions	int	read	Number of subscriptions authorized
subscriptionsDenied	int	read	Number of subscriptions denied authorization
writes	int	read	Number of writes authorized
writesDenied	int	read	Number of writes denied authorization

Notifications

Class Name	Description
javax.management.Notification	News that a client interaction with a topic was not allowed (deprecated)

ClientStatistics

Monitoring interface to the client session statistics MBean

Object name format

The objectName for MBeans of this type is of the following form:

`com.pushtechnology.diffusion:type=ClientStatistics,server="server_name"`

Attributes

Name	Type	Read/Write	Description
clientOutputFrequency	long	read-write	Statistics output frequency in milliseconds
clientResetFrequency	long	read-write	The frequency at which the counters are reset
concurrentClientCount	int	read	The current client session count
connectionCounts	Map	read	The current client session count, broken down by client type
maximumConcurrentClientCount	int	read	The maximum number of concurrent client sessions
maximumDailyClientCount	int	read	The count of client sessions started in a day

Connector

Management interface to a connector

Object name format

The objectName for MBeans of this type is of the following form:

```
com.pushtechnology.diffusion:type=Connector,name="name",server="server_name"
```

Attributes

Name	Type	Read/Write	Description
keepAliveQueueMaximumDepth	int	read-write	The maximum queue depth used for clients in the keep-alive state
keepAliveTime	long	read-write	The time in milliseconds that a unexpectedly disconnected client is kept alive before closing
numberOfAcceptors	int	read	The number of acceptors
queueDefinition	String	read-write	The queue definition
totalNumberOfConnections	long	read	The number of connections accepted since the connector was started
uptime	String	read	The time this connector has been running as a formatted string, or 0 if the connector is not running
uptimeMillis	long	read	The time this connector has been running in milliseconds, or 0 if the connector is not running

Operations

Name	Return Type	Arguments	Impact	Description
remove	void	0	ACTION	Remove the connector. It will not be possible to restart the connector again (until system restart).

Name	Return Type	Arguments	Impact	Description
start	void	0	ACTION	Start the connector

Name	Return Type	Arguments	Impact	Description
stop	void	0	ACTION	Stop the connector. Allows it to be restarted.

JMXAdapter

Management interface to the adapter that reflects MBean attributes and notifications as Diffusion topics

Object name format

The objectName for MBeans of this type is of the following form:

```
com.pushtechnology.diffusion:type=JMXAdapter,server="server_name"
```

Attributes

Name	Type	Read/Write	Description
MBeanFilter	String	read	the filter applied to the set of all MBeans, as a flattened AST
adapterState	String	read	Values: STOPPED, STARTED
updateFrequency	long	read	MBean attribute poll frequency, in milliseconds

Operations

Name	Return Type	Arguments	Impact	Description
start	void	0	UNKNOWN	Build the topic tree and periodically refresh it

Name	Return Type	Arguments	Impact	Description
stop	void	0	UNKNOWN	Cease refresh and remove topics

Log

Management interface for Log Definition

Object name format

The objectName for MBeans of this type is of the following form:

```
com.pushtechnology.diffusion:type=Log,name="name",server="server_name"
```

Attributes

Name	Type	Read/Write	Description
description	LogDescription	read	The LogDescription for this log
filename	String	read	The fully qualified filename of this log
logLevel	String	read-write	The current log level as a string

Multiplexer

Management interface to a multiplexer

Object name format

The objectName for MBeans of this type is of the following form:

`com.pushtechnology.diffusion:type=Multiplexer,name="name",server="server_name"`

Attributes

Name	Type	Read/Write	Description
latencyWarningTime	long	read	The latency threshold of this multiplexer, after which notifications will be sent
pendingEvents	int	read	The number of operations pending on the multiplexer queue
name	String	read	The Multiplexer name
numberOfClients	int	read	The current number of clients assigned to multiplexer

Operations

Name	Return Type	Arguments	Impact	Description
diagnosticReport	String	0	UNKNOWN	Generate a diagnostic report describing the state of this multiplexer

Notifications

Class Name	Description
javax.management.Notification	Published in case of multiplexer latency (deprecated)

MultiplexerManager

Management interface to the multiplexer manager

Object name format

The objectName for MBeans of this type is of the following form:

`com.pushtechnology.diffusion:type=MultiplexerManager,name="name",server="server_name"`

Attributes

Name	Type	Read/Write	Description
numberOfMultiplexers	int	read	The number of multiplexers

Operations

Name	Return Type	Arguments	Impact	Description
diagnosticReport	String	0	UNKNOWN	Generate a diagnostic report for each multiplexer

Publisher

Management interface for a publisher

Object name format

The objectName for MBeans of this type is of the following form:

`com.pushtechnology.diffusion:type=Publisher,name="name",server="server_name"`

Attributes

Name	Type	Read/Write	Description
inboundClientAverageMessageSize	long	read	The average size of a message received from clients
inboundClientNumberOfMessages	long	read	The count of messages received from clients
logLevel	String	read-write	The log level for this publisher
numberOfTopics	int	read	The count of topics associated with this publisher
outboundAverageMessageSize	long	read	The average size of a message sent to clients
outboundNumberOfMessages	long	read	The count of messages sent to clients
started	boolean	read	True if the publisher is started

Operations

Name	Return Type	Arguments	Impact	Description
removePublisher	void	0	ACTION	Permanently remove the publisher

Name	Return Type	Arguments	Impact	Description
startPublisher	void	0	ACTION	Start this publisher

Name	Return Type	Arguments	Impact	Description
stopPublisher	void	0	ACTION	Stop this publisher

Name	Return Type	Arguments	Impact	Description
undeploy	void	0	ACTION	Undeploy this publisher

Server

Diffusion Controller

Object name format

The objectName for MBeans of this type is of the following form:

```
com.pushtechnology.diffusion:type=Server,server="server_name"
```

Attributes

Name	Type	Read/Write	Description
buildDate	String	read	The build date and time
freeMemory	long	read	The amount of free memory in the Java virtual machine
licenseExpiryDate	Date	read	The license expiry date
maxMemory	long	read	The total amount of memory in the Java virtual machine
numberOfTopics	long	read	The number of topics on this server
release	String	read	The Diffusion release string, for example, 5.1.2_01
startDate	Date	read	The date and time at which this Diffusion server was started
startDateMillis	long	read	The time at which this Diffusion server was started, as milliseconds since the epoch
timeZone	String	read	The name of the time zone to which this Diffusion server belongs
totalMemory	long	read	The total amount of memory in the Java virtual machine
uptime	String	read	The time that this controller has been running, as a formatted string. For example, "3 hours 4 minutes 23 seconds"
uptimeMillis	long	read	The time this controller has been running, in milliseconds
usedPhysicalMemorySize	long	read	Free physical memory, in bytes
usedSwapSpaceSize	long	read	Used swap space, in bytes
userDirectory	String	read	The directory in which the Diffusion server was started
userName	String	read	The name of the user to whom the Diffusion server belongs

Operations

Name	Return Type	Arguments	Impact	Description
checkLicense	void	0	ACTION	Recheck the license being used

Name	Return Type	Arguments	Impact	Description
stopController	void	0	ACTION	Stop this Diffusion controller

Name	Return Type	Arguments	Impact	Description
stopController	void	2	ACTION	Stop this Diffusion controller, and record the reason and adminName

Argument name	Type	Description
reason	String	The reason this server is stopping
adminName	String	The name of the administrator

StatisticsService

Interface for StatisticsService controller

Object name format

The objectName for MBeans of this type is of the following form:

```
com.pushtechnology.diffusion:type=StatisticsService,server="server_name"
```

Attributes

Name	Type	Read/Write	Description
clientStatisticsEnabled	boolean	read-write	True if aggregate statistics for clients are enabled
enabled	boolean	read-write	True if the statistics service is enabled
publisherStatisticsEnabled	boolean	read-write	True if aggregate statistics for publishers are enabled
topicStatisticsEnabled	boolean	read-write	True if aggregate statistics for topics are enabled

ThreadPool

Management interface to the basic thread pool

Object name format

The objectName for MBeans of this type is of the following form:

```
com.pushtechnology.diffusion:type=ThreadPool,name="name",server="server_name"
```

Attributes

Name	Type	Read/Write	Description
activeCount	int	read	The number of active threads
coreSize	int	read-write	The number of threads to maintain
keepAlive	long	read-write	Keep-alive time in milliseconds
largestSize	int	read	The largest number of threads that have simultaneously been in the pool
maximumSize	int	read-write	Maximum pool size
queueLowerThreshold	int	read	The lower queue size at which the notification handler will be notified
queueMaximumSize	int	read	The maximum queue size that the task queue can reach
queueSize	int	read	The size of the embedded task queue
queueUpperThreshold	int	read	The upper queue size at which the notification handler will be notified
size	int	read	The current number of threads in the pool
taskCount	long	read	The approximate total number of tasks that have ever been scheduled for execution

VirtualHost

HTTP virtual host management interface

Object name format

The objectName for MBeans of this type is of the following form:

```
com.pushtechnology.diffusion:type=VirtualHost,name="name",server="server_name",we
```

Attributes

Name	Type	Read/Write	Description
cacheSizeBytes	int	read	The cache size in bytes
cacheSizeEntries	int	read	The number of entries in the cache
aliasFile	String	read	the alias file name
aliasProcessor	HTTPAliasProcessor	read	the alias processor object
cache	HTTPCache	read	the HTTP cache
compressionThreshold	int	read	the compression threshold
debug	boolean	read	true if debug is set
documentRoot	String	read	the document root directory
errorPage	String	read	the error-page file name
fileServiceName	String	read	the file service name

Name	Type	Read/Write	Description
homePage	String	read	the home-page file name
host	String	read	the host name
minify	boolean	read	true if the minify property is set
name	String	read	the virtual host name
numberOfRequests	int	read	number of requests actioned since service was started
static	boolean	read	true if static
webServerName	String	read	the web server name

Operations

Name	Return Type	Arguments	Impact	Description
startService	void	0	ACTION	Restart a previously stopped virtual host

Name	Return Type	Arguments	Impact	Description
stopService	void	0	ACTION	Stops the virtual host from processing requests

Name	Return Type	Arguments	Impact	Description
clearCache	void	0	ACTION	Clear the cache of all entries

The JMX adapter

The JMX adapter reflects JMX MBeans and their properties and notifications as topics.

The JMX adapter is packaged in the Diffusion publisher. The Diffusion publisher must be running for the JMX adapter be enabled.

You can configure the adapter to reflect the state of JMX MBeans and MXBeans as topics. These MBeans can be built-in, Diffusion, or third-party in origin.

The following aspects of the JMX adapter can be configured:

- Whether it is enabled or disabled.
By default, the adapter is enabled.
- Which MBeans are reflected as topics.
By default, all Diffusion MBeans, `java.nio:*`, `java.lang:*`, and `java.util.logging:*` are reflected as topics.
- How often the data on those topics is refreshed.
By default, the topics are refreshed every 3 seconds.

For more information about configuring the JMX adapter, see [Configuring the JMX adapter](#) on page 605

Many statistics are available as MBean properties, for example, CPU load, OS version, number of file-descriptors, and threads. Making these statistics available as topics to Diffusion clients makes possible the implementation of system monitoring solutions to the web, and all other Diffusion platforms.

Note: Publishing MBean data to topics can constitute a security risk. Ensure that crucial information about your Diffusion server is protected by permissions.

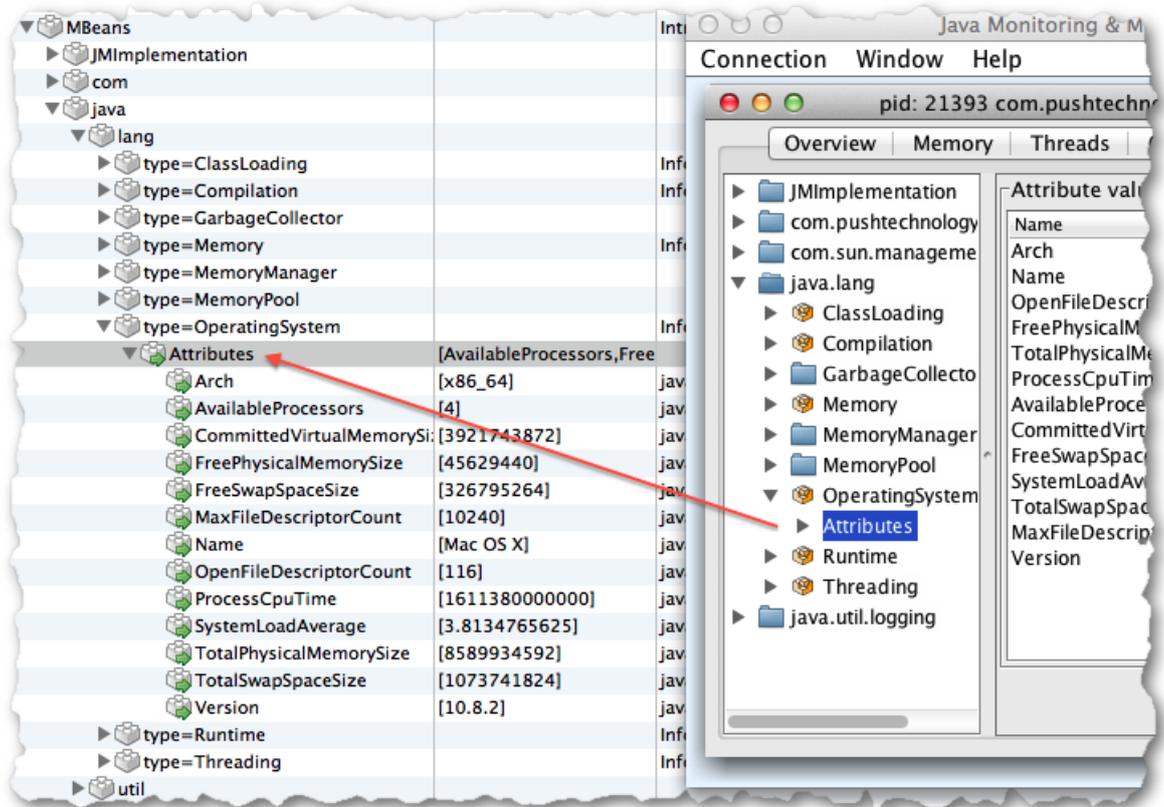


Figure 62: Reflecting MBeans as topics

MBean notifications are also available as topics. Whenever a notification is thrown and the matching topic is subscribed and a message holding a number of key attributes is published to it.

Table 65: Notifications as topics

Record starting ...	Holding
message	<code>javax.management.Notification.getMessage()</code>
sequenceNumber	<code>javax.management.Notification.getSequenceNumber()</code>
timeStamp	<code>javax.management.Notification.getTimeStamp()</code>
userData	<code>javax.management.Notification.getUserData()</code> if present
source	<code>javax.management.Notification.getSource()</code>

The JMX Adapter is itself an MBean with object-name `com.pushtechology.diffusion:name=JMXAdapter`, which exposes the polling frequency in milliseconds as attribute 'UpdateFrequency'. A value less than or equal to zero prevents polling.

MXBeans versus Simple MBeans

The JMX adapter caters for both MXBeans and simpler MBeans. All MBean attributes are serialized as strings when converted to topics, this might be impractical if a solution returns an object or an array of objects. MXBean attributes with ArrayType and CompositeType types are treated differently.

- CompositeType Fields within the composite attribute are mapped to discrete topics.

MBean Path	Value	Type
MBeans		Introspect JMX Mbeans as Diffusion topics
JMIImplementation		
com		
java		
lang		
type=ClassLoading		Information on the management interface of the MB
type=Compilation		Information on the management interface of the MB
type=GarbageCollector		
name=ConcurrentMarkSweep		Information on the management interface of the MB
name=ParNew		Information on the management interface of the MB
Attributes	[LastGcInfo,MemoryPoolN	
CollectionCount	[5]	java.lang.Long
CollectionTime	[44]	java.lang.Long
LastGcInfo	[endTime,memoryUsageB	javax.management.openmbean.CompositeData
duration	[5]	java.lang.Long
endTime	[3551376]	java.lang.Long
id	[5]	java.lang.Long
memoryUsageAfterGc	[CMS Old Gen,javaax.mana	table of java.util.Map<java.lang.String, java.lang.ma
memoryUsageBeforeGc	[CMS Old Gen,javaax.mana	table of java.util.Map<java.lang.String, java.lang.ma
startTime	[3551371]	java.lang.Long
MemoryPoolNames	[Par Eden Space,Par Survi	array of java.lang.String with 1 dimension
Name	[ParNew]	java.lang.String
Valid	[true]	java.lang.Boolean

Figure 63: Showing a composite attribute as a topic nest

- ArrayType One dimensional arrays are presented as a single record with many values. Two dimensional arrays are not supported. ArrayType attributes holding attributes that are not SimpleType are not supported (for example, an ArrayType attribute holding Composite or ArrayType values)

Attribute Name	Value	Class
Attributes	[VmVersion,SpecVendor,ClassPath,SystemProp...	
BootClassPath	[/Library/Java/JavaVirtualMachines/1.6.0_31-b...	java.lang.String
BootClassPathSupport	[true]	java.lang.Boolean
ClassPath	[/Users/martincowie/Development/Diffusion/M...	java.lang.String
InputArguments	[-Xmx1g,-Dcom.sun.management.jmxremote,	array of java.lang.String with 1 dimension
LibraryPath	[./Library/Java/Extensions:/System/Library/Ja...	java.lang.String
ManagementSpecVer	[1.2]	java.lang.String
Name	[17400@roobarb.local]	java.lang.String
SpecName	[Java Virtual Machine Specification]	java.lang.String
SpecVendor	[Sun Microsystems Inc.]	java.lang.String
SpecVersion	[1.0]	java.lang.String
StartTime	[1351772970790]	java.lang.Long
SystemProperties	[awt.nativeDoubleBuffering,true][awt.toolkit,ap	table of java.util.Map<java.lang.String, java
Uptime	[106824079]	java.lang.Long
VmName	[Java HotSpot(TM) 64-Bit Server VM]	java.lang.String
VmVendor	[Apple Inc.]	java.lang.String
VmVersion	[20.6-b01-415]	java.lang.String

Figure 64: Topics reflecting an ArrayType MBean attributes

Related tasks

[Configuring the JMX adapter](#) on page 605

The JMX adapter can reflect JMX MBeans their properties and notifications as topics. Configure the JMX adapter using the `Publishers.xml` configuration file.

Statistics

Diffusion provides statistics about the server, publishers, clients, and topics.

Statistics exposed by MBeans

Diffusion provides a set of statistics as MBeans. For more information about these statistics, see [MBeans](#) on page 741.

These statistics can be accessed in the following ways:

- Using a JMX tool, such as VisualVM or JConsole. For more information, see [Using Java VisualVM](#) on page 736 or [Using JConsole](#) on page 738.
- Using the Diffusion monitoring console. For more information, see [Diffusion monitoring console](#) on page 759.
- Through topics under the topic `Diffusion/Metrics`.

You can configure whether these statistics are collected and how they are reported. For more information, see [Configuring statistics](#) on page 757.

You can configure Diffusion to report the top-level statistics through JMX. These statistics appear under `com.pushtechology.diffusion.metrics`. Reporting statistics through JMX can have a significant performance impact. Ensure that you test how using MBeans for statistics affects your solution.

The following is a list of all the top level statistics and their attributes.

- `clients.concurrent`
 - `Count`
- `clients.concurrent_max`

- Value
- client.connections
 - RateUnit
 - Count
 - FifteenMinuteRate
 - FiveMinuteRate
 - MeanRate
 - OneMinuteRate
- clients.disconnections
 - RateUnit
 - Count
 - FifteenMinuteRate
 - FiveMinuteRate
 - MeanRate
 - OneMinuteRate
- eventpublishers.concurrent
 - Count
- eventpublishers.connections
 - RateUnit
 - Count
 - FifteenMinuteRate
 - FiveMinuteRate
 - MeanRate
 - OneMinuteRate
- eventpublishers.disconnections
 - RateUnit
 - Count
 - FifteenMinuteRate
 - FiveMinuteRate
 - MeanRate
 - OneMinuteRate
- publishers.loaded
 - Count
- publishers.started
 - Count
- topics.additions
 - RateUnit
 - Count
 - FifteenMinuteRate
 - FiveMinuteRate
 - MeanRate
 - OneMinuteRate
- topics.count
 - Count
- topics.deletions
 - RateUnit

- Count
- FifteenMinuteRate
- FiveMinuteRate
- MeanRate
- OneMinuteRate

In all top-level statistics that attributes have the following values:

Count

The total number of items or events of this type over the lifetime of the metric.

RateUnit

The unit for the various rates. Generally, this is events per second.

MeanRate

The count divided by the lifetime of the metric.

OneMinuteRate, FiveMinuteRate, FifteenMinuteRate

A moving average across the given time period that is exponentially weighted towards new data.

Statistics in the Publisher API

You can get some statistics from the Publisher API. For more information, see the [Java Unified API documentation](#).

The following statistics are provided through the Publisher API:

- PublisherStatistics
 - InboundClientMessageStatistics
 - AverageMessageSize
 - BytesOnWire
 - NumberOfMessages
 - OutboundMessageStatistics
 - AverageMessageSize
 - BytesOnWire
 - NumberOfMessages
- ClientStatistics
 - InboundMessageStatistics
 - AverageMessageSize
 - BytesOnWire
 - NumberOfMessages
 - OutboundMessageStatistics
 - AverageMessageSize
 - BytesOnWire
 - NumberOfMessages
- TopicStatistics
 - InboundMessageStatistics
 - AverageMessageSize
 - BytesOnWire
 - NumberOfMessages
 - OutboundMessageStatistics

- `AverageMessageSize`
- `BytesOnWire`
- `NumberOfMessages`
- `TotalNumberOfSubscribers`

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

DEPRECATED: [Introspector](#) on page 770

An introduction to the Introspector Eclipse plugin.

Related reference

[Diffusion monitoring console](#) on page 759

A web console for monitoring the Diffusion server.

[Logging](#) on page 784

Diffusion uses the Simple Logging Facade for Java (SLF4J) API to log messages from the Diffusion server or from publishers running on the Diffusion server. SLF4J separates the logging of messages in the Diffusion server from the logging framework. This separation enables you to configure an independent back-end implementation to format and write out the log messages.

[Integration with Splunk](#) on page 1049

How to achieve basic integration between Diffusion and the Splunk™ analysis and monitoring application

Configuring statistics

You can configure statistics using the `etc/Statistics.xml` configuration file or programmatically using the Classic API.

By default statistic collection is turned off for performance reasons, to enable statistic collection, set the statistics root attribute `enabled` in `Log.xml` to true.

Statistics and performance

Collecting and communicating statistics introduces a performance overhead on the Diffusion server.

Enabling client instance statistics in a solution with significant numbers of clients can cause up to 20% reduction in the maximum throughput achieved. This performance impact can inhibit the system from supporting further connections.

If your system has more than 20,000 concurrent client connections per Diffusion instance, we recommend that client instance statistics be turned off.

Enabling topic instance statistics causes the creation of 5 statistics topics for each of your own topics.

If your system supports very large number of topics, we recommend that topic instance statistics are turned off.

The `Statistics.xml` configuration file

Diffusion servers provide statistics which are available through the JMX MBeans or through topics under the topic `Diffusion/Metrics`. If statistics is enabled via `etc/Statistics.xml`, users can take measurements including the average message size, number of messages per topic per second, etc.

The statistics configuration has several distinct elements, that allow granular control over what is enabled on server start.

- `<statistics>`

The top-level element. Setting the `enabled` property to `false` will disable all statistics for the server.

- `<topic-statistics>`, `<client-statistics>`, `<publisher-statistics>`, and `<server-statistics>`

Enabling these elements provides aggregate metrics for each given class.

The collection of metrics is configured separately from the distribution thereof. Within the `Statistics.xml` configuration file, there is also a `<reporters>` element which contains definitions of available reporters, which expose metrics over different mediums.

Some reporters allow certain properties to be passed to them. For instance, the topics reporter allows the use of `<property name="interval">` to provide the desired update frequency in seconds. Details of valid properties is documented within the `etc/Statistics.xml` file itself.

Configuring a reporter to distribute statistics through JMX

Inside the `reporters` element of the `Statistics.xml` configuration file, add the following XML:

```
<reporter name="JMX" enabled="true">
  <type>JMX</type>
</reporter>
```

Reporting statistics through JMX can have a significant performance impact. Ensure that you test how using MBeans for statistics affects your solution.

Configuring a reporter to distribute statistics through topics

Inside the `reporters` element of the `Statistics.xml` configuration file, add the following XML:

```
<reporter name="Topics" enabled="true">
  <type>TOPIC</type>
</reporter>
```

Programmatic configuration

You can programmatically query and configure the recording and calculation of statistics for the classes:

- `com.pushtechology.diffusion.api.publisher.Client`
- `com.pushtechology.diffusion.api.topic.Topic`
- `com.pushtechology.diffusion.api.publisher.Publisher`

Developers are able to query the state of each class, to discover whether it is recording statistics using method `isStatisticsEnabled()`, stop recording with `stopStatistics()` and start recording using `startStatistics()`. The relevant API documentation holds more detail on the subject.

Diffusion monitoring console

A web console for monitoring the Diffusion server.

About

The Diffusion Monitoring Console is an optional publisher, provided as `console.dar`. It is deployed by default, and can be undeployed in the same manner as any DAR file. It exists to give operational staff using a web browser accessible visibility over the operations of a Diffusion solution

To manage a Diffusion server and make changes to it, use JMX tools such as JConsole. Unless you have to stop the Diffusion server, and stop and restart a publisher.

Dependencies

The console depends on the Diffusion publisher to mirror JMX MBeans as topics. The console also makes use of the statistics controlled by `etc/Statistics.xml`

The live graphing feature mandates a web browser that supports Scalar Vector Graphics (SVG). Most modern web browsers implement the features required by the Console however Internet Explorer v9 is the recommended minimum for Microsoft users.

There are also two configuration settings within the Diffusion publisher configuration within `etc/Publishers.xml`. These are:

- `console.control.server` – Enable the ability to stop the Diffusion server through the console.
- `console.control.publishers` – Enable the ability to stop a particular publisher through the console.

Both of these options are disabled by default.

Logging in

The console is secured by a username and password. The username you use to login must have permissions to view and act on information on the Diffusion server, for example by having the ADMINISTRATOR role.

The default configuration of the Diffusion server, provides a user 'admin' with the password 'password'. This user has the appropriate permissions to use all of the console capabilities. For more information, see [Pre-defined users](#) on page 146.

Note: We recommend that you change the default security configuration before putting your solution into production. For more information, see [Configuring user security](#) on page 588

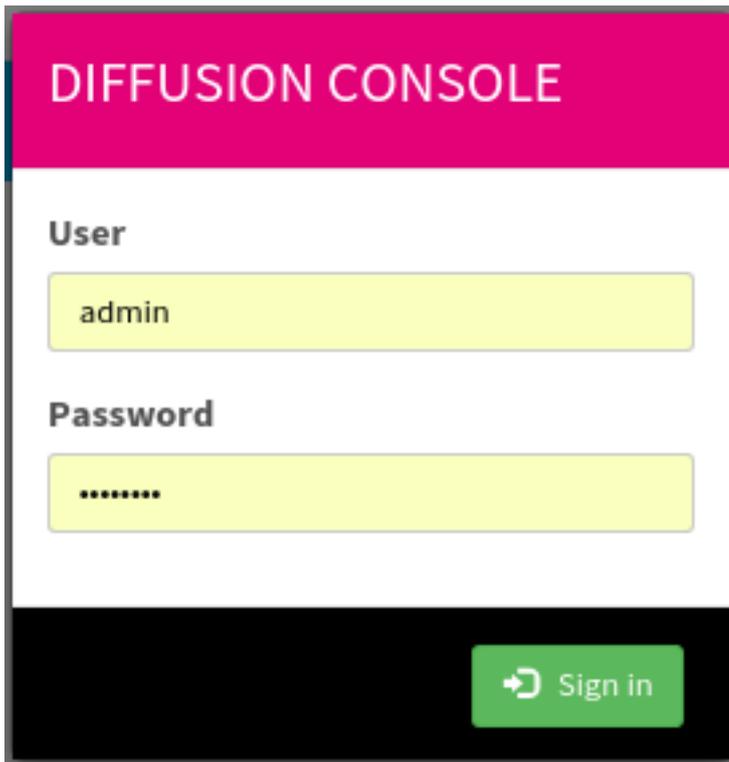


Figure 65: Logging in the monitoring console

Features: Overview tab

By default the console is deployed as part of Diffusion. It is available in a fresh installation at <http://localhost:8080/console>.

Default layout

By default, the console consists of six panels, each focusing on a key feature of the server.

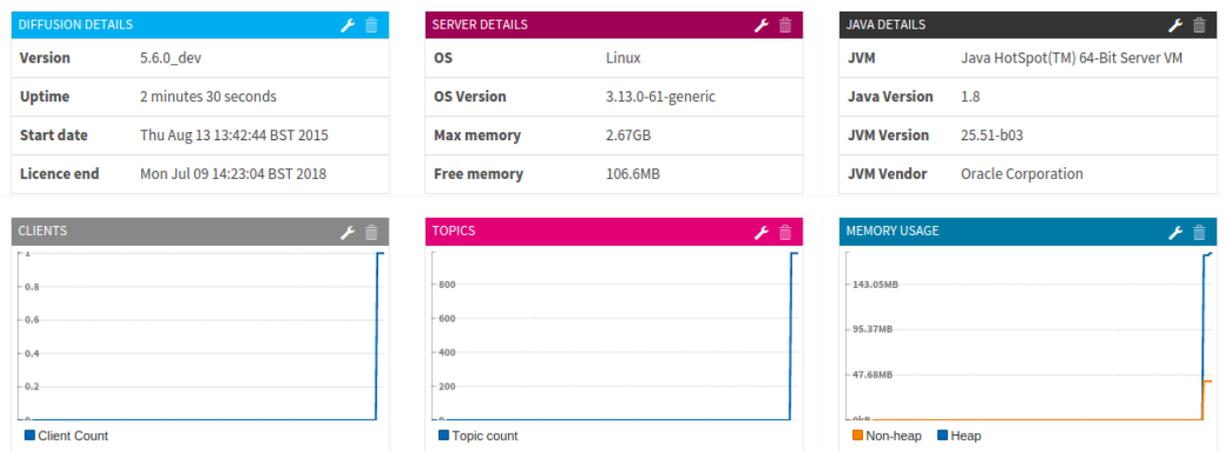


Figure 66: The default console layout

- **Diffusion Details:** the server version; the server up time, the server start date and time and the time and which the current license expires.
- **Server details:** the name and version of the underlying operating system; the total memory available (physical and virtual) and the amount of free memory.

- **Java Details:** the name, vendor and version of the JVM.

Instead of tabular data the second row show live line graphs.

- **Memory pool usage:** the values over time of the memory used by the JVM process.
- **Clients:** the value over time of the number of clients connected.
- **Topics:** the value over time of the number of topics on your Diffusion server.

Publishers table

At the bottom of the Overview is the publishers table. At a glance this shows the installed publishers and their vital statistics: the number of topics created, client connected, messages sent, bytes sent and finally publisher status

PUBLISHERS					
Publisher name	Topics	Clients	Messages sent	Bytes sent	Status
AssetPublisher	227				Started Details
ConsolePublisher	0				Started Details
DemoController	0				Started Details
Diffusion	741				Started Details
DogfightPublisher	7				Started Details
DrawingBoard	2				Started Details
Echo	1				Started Details
Tech Magnets	3				Started Details
V5 Diffusion Publisher	0				Started Details

Figure 67: The table of publishers

Using the pull-down menu on the **Details** button publishers can be stopped and restarted. The **Details** button itself reveals the publisher statistics: clients, topics, average messages per second and average bytes per second.

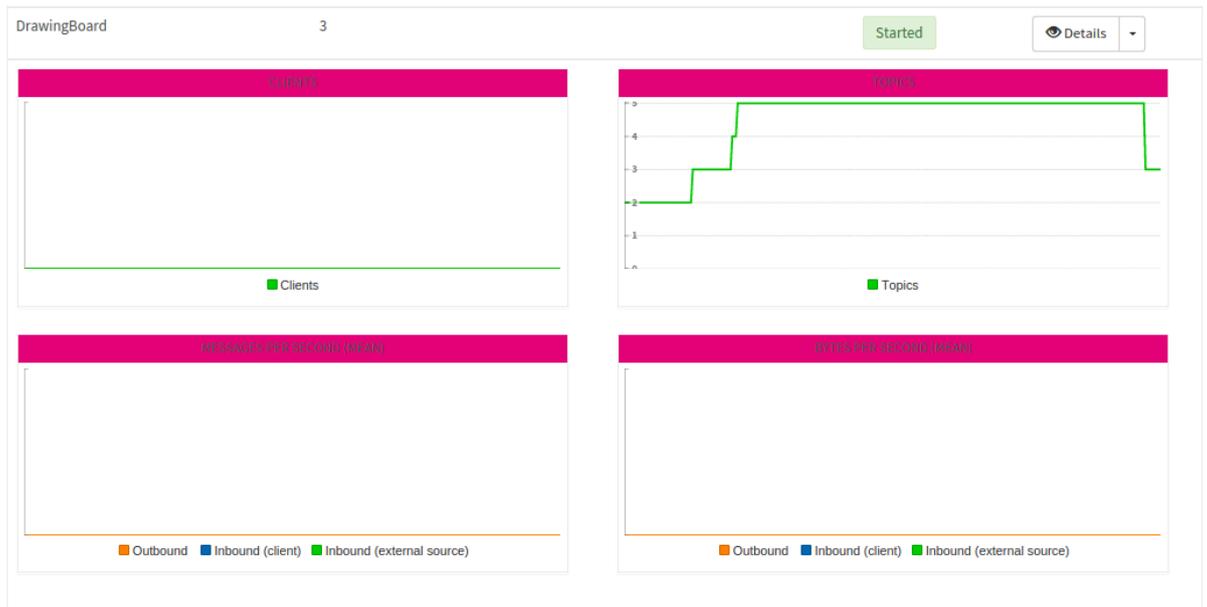


Figure 68: Publisher statistics graphs

Features: Topics tab

The Topics tab brings to the web browser the ability to browse and interact with the Diffusion topic tree.

TOPICS						
<input type="button" value="Subscribe"/> <input type="button" value="Unsubscribe"/> <input type="button" value="Add to dashboard"/>						
<input type="checkbox"/>	Name	Value	Clients	Messages sent	Bytes sent	
<input type="checkbox"/>	Accounts		0	0	0	Details
<input type="checkbox"/>	▶ Assets		0	0	0	Details
<input type="checkbox"/>	Commands		0	0	0	Details
<input type="checkbox"/>	Data		0	0	0	Details
<input type="checkbox"/>	▶ Diffusion		0	0	0	Details
<input type="checkbox"/>	▶ dogfight		0	0	0	Details
<input type="checkbox"/>	▶ drawingboard		0	0	0	Details
<input type="checkbox"/>	Echo		0	0	0	Details
<input type="checkbox"/>	News		0	0	0	Details
<input type="checkbox"/>	▶ Techmagnets		0	0	0	Details
<input type="checkbox"/>	Trades		0	0	0	Details

Figure 69: The table of topics

Users can intuitively browse the live topic tree, fetch and subscribe to topics. If the server is so configured the table also shows the number of subscribed clients, messages sent and bytes sent. Enable individual topic statistics through `etc/Statistics.xml`, for example,

```
<!-- Enable global topic statistics -->
<topic-statistics enabled="true">
```

```

<!-- Enable individual topic instance statistics -->
<monitor-instances>true</monitor-instances>

</topic-statistics>

```

Once a set of topics is selected using its checkbox the Subscribe and Unsubscribe work intuitively, and each button has an recursive alternative available through the drop-down menu-button.

The details button shows more detail on the topic in question, as well as offering to fetch the topic value

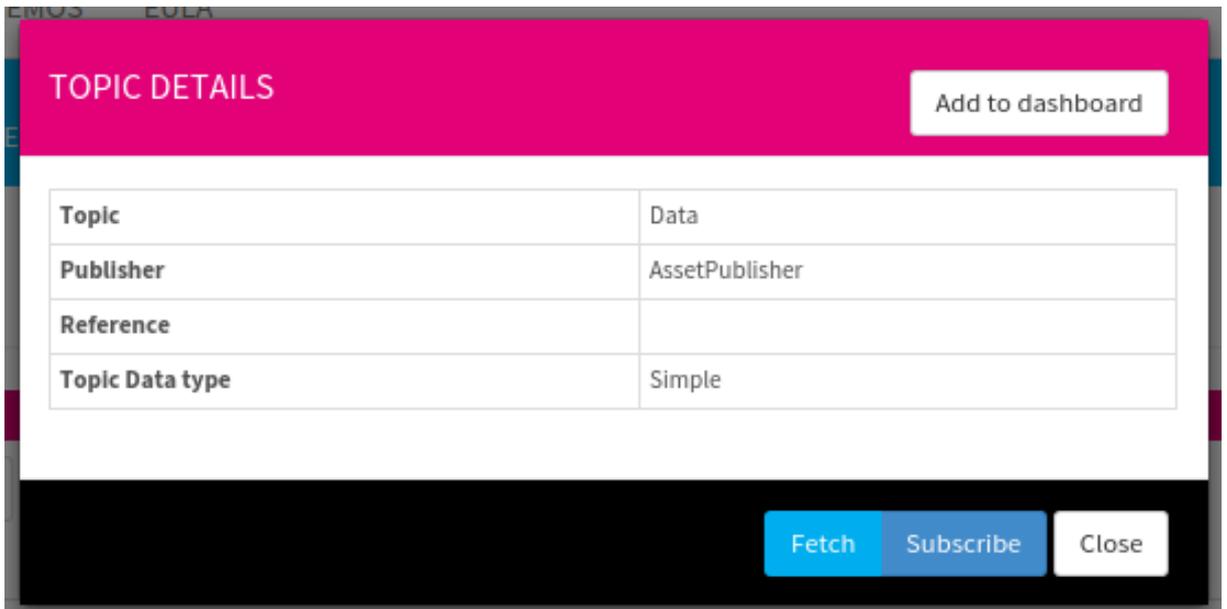


Figure 70: Details of the topic publishing the CPU load of the host server

Features: Clients tab

The Client tab shows a live list of the clients connected to the Diffusion server. Additionally it shows the number of messages to and from the server, the client IP address, connection type and connection time.

Session ID	Messages to server	Messages from server	IP Address	Connection Type	Time connected
883B9D732B751E71-0000000100000000			127.0.0.1	WEBSOCKET_BROWSER_CLIENT	0 hours, 1 minutes, 10 seconds

Figure 71: The table of clients

Configure the Diffusion server to provide live client statistics through `etc/Statistics.xml`

```

<!-- Enable global client statistics -->
<client-statistics enabled="true">
  <!-- Definition of the log in Logs.xml -->
  <log-name>stats</log-name>
  <!-- Specifies the output frequency of the log, this is one entry
  per frequency -->

  <output-frequency>lh</output-frequency>
  <!-- Enable individual client instance statistics -->
  <monitor-instances>true</monitor-instances>

```

```
</client-statistics>
```

Features: Logs tab

The Logs tab shows a live color-coded display of log entries emitted by the server at the levels of **INFO**, **WARN**, and **ERROR**.

Level	Timestamp	Message
INFO	2015-08-13 13:44:35.743	Connector Client Connector: Requested input buffer size could not be allocated, requested: '1M' allocated: '208K'.
INFO	2015-08-13 13:42:46.144	Loaded XML Allases properties from '/home/kshann/Diffusion5.6.0_dev/deploy/dogfight/etc/Allases.xml'.
INFO	2015-08-13 13:42:46.144	Diffusion finished deploying 6 components.
INFO	2015-08-13 13:42:46.142	Started publisher 'DogfightPublisher'.
INFO	2015-08-13 13:42:46.117	Starting publisher 'DogfightPublisher'.
INFO	2015-08-13 13:42:46.111	Loaded XML Publishers properties from '/home/kshann/Diffusion5.6.0_dev/deploy/dogfight/etc/Publishers.xml'.
INFO	2015-08-13 13:42:46.069	Deploying publishers compiled for Diffusion version '5.6.0'.
INFO	2015-08-13 13:42:46.067	Deploying publishers in DAR file '/home/kshann/Diffusion5.6.0_dev/deploy/dogfight.dar'.
INFO	2015-08-13 13:42:46.066	Started publisher 'Echo'.
INFO	2015-08-13 13:42:46.062	Starting publisher 'Echo'.
INFO	2015-08-13 13:42:46.048	Loaded XML Allases properties from '/home/kshann/Diffusion5.6.0_dev/deploy/echo/etc/Publishers.xml'.
INFO	2015-08-13 13:42:46.031	Deploying publishers compiled for Diffusion version '5.6.0'.
INFO	2015-08-13 13:42:46.029	Deploying publishers in DAR file '/home/kshann/Diffusion5.6.0_dev/deploy/echo.dar'.
INFO	2015-08-13 13:42:46.029	Loaded XML Allases properties from '/home/kshann/Diffusion5.6.0_dev/deploy/asset-trader/etc/Allases.xml'.
INFO	2015-08-13 13:42:46.026	Started publisher 'AssetPublisher'.
INFO	2015-08-13 13:42:45.500	Starting publisher 'AssetPublisher'.

Figure 72: The table of log entries

Users can also perform client-side simple filtering on log entries. Unlike other monitoring metrics the Diffusion server retains up to 250 log entries in memory.

Features: Security tab

The Security tab shows a live list of security principals and roles that are configured on the Diffusion server.

For more information about security, see [Security](#) on page 129.

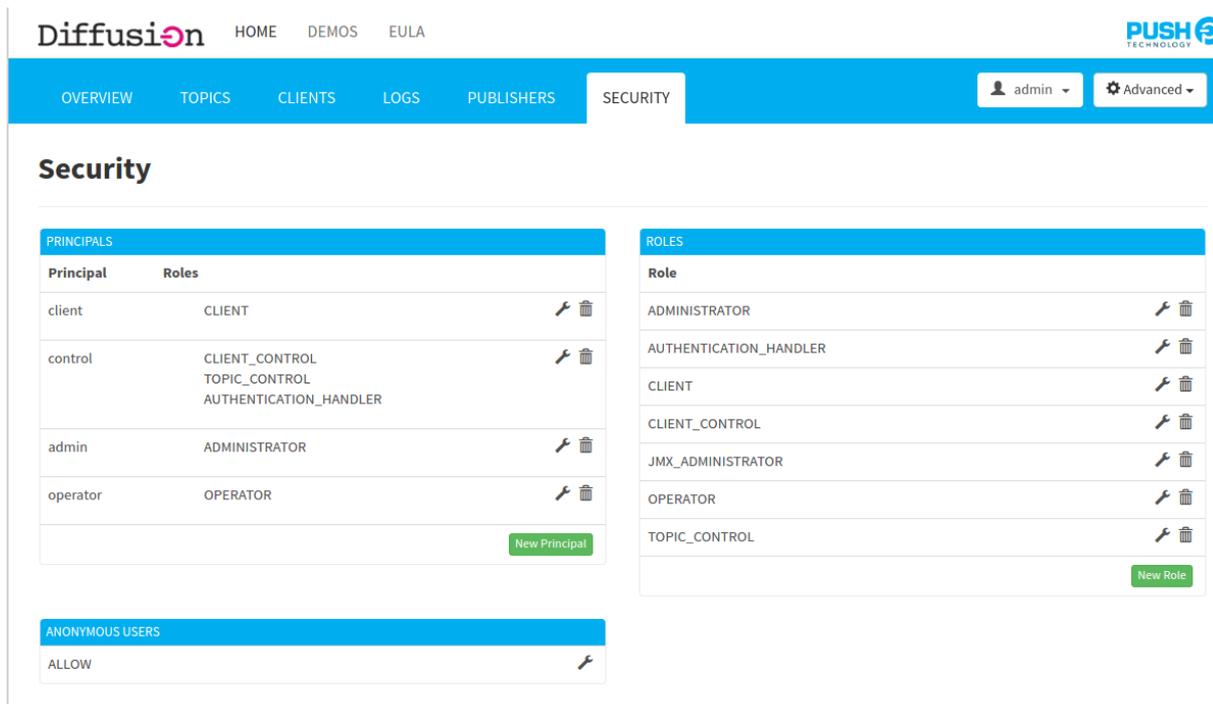


Figure 73: Security tables

Create, edit, or delete principals: The **Principals** table shows a list of the principals that the system authentication handler is configured to allow to connect to the Diffusion server. The table also shows the roles that are assigned to any client session that authenticates with the principal.

Click the **New Principal** button to add a new principal and define its associated password and roles.

Click the spanner icon next to an existing principal to edit that its password or roles.

Click the trashcan icon next to an existing principal to delete that principal.

Edit authentication policy and roles for anonymous users: The **Anonymous Users** table shows the authentication decision for client session that connect anonymously to the Diffusion server. You can choose to ALLOW or DENY anonymous connections or to ABSTAIN from the authentication decision, which then passes to the next configured authentication handler.

Click the spanner icon to edit the authentication decision for anonymous connections and, if that decision is ALLOW, edit any roles that are assigned to anonymous sessions.

Create, edit, or delete roles: The **Roles** table shows a list of roles that have been configured in the security store of the Diffusion server. These are the roles that you can choose to assign to any principals that connect to the Diffusion server.

Click the **New Role** button to add a new role and define its permissions and any roles it inherits from.

Click the spanner icon next to an existing role to edit its permissions and any roles it inherits from.

Click the trashcan icon next to an existing role to delete that role.

Advanced

Saving the console layout

Users can save changes made to their console layout with the “Save Overview layout” button. This persists a file on the server side, making it shared amongst all Console users.

White & Blacklist editing

The Console optionally maintains a blacklist or whitelist of IP addresses that are allowed to make use of it. Users can specify discrete IP addresses or use syntax supported by [etc/SubscriptionValidationPolicy.xml](#) to cover subnets. In order to make these changes active, after editing the whitelist or blacklist and clicking the "Save settings" button, you must restart the server.

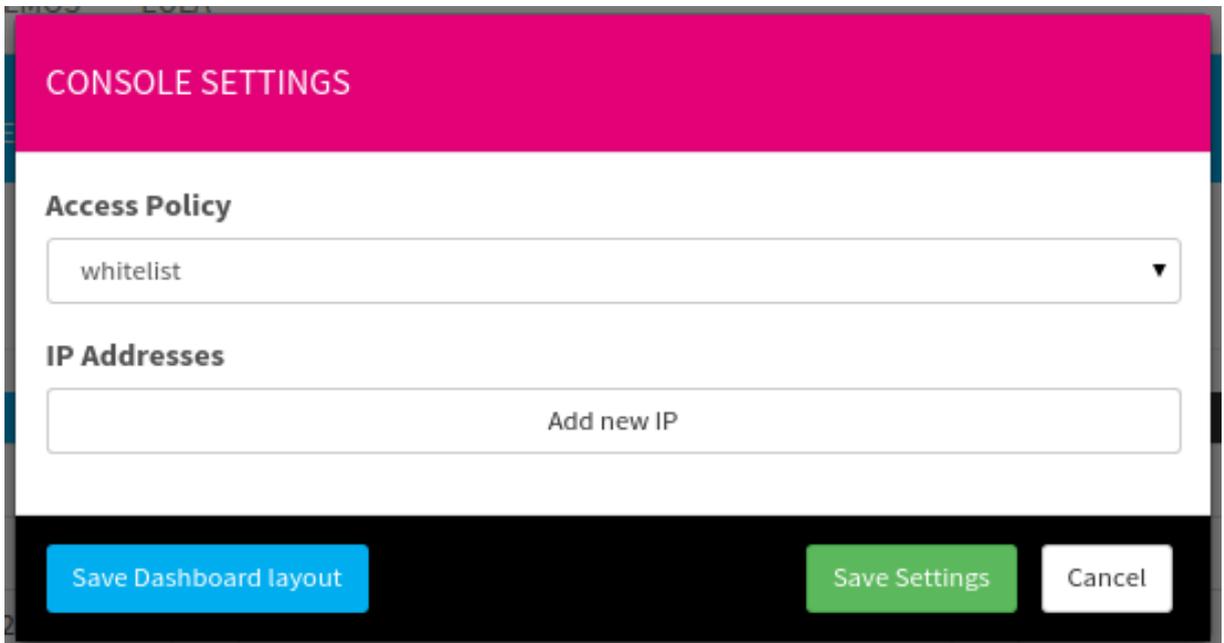


Figure 74: Editing the Access Policy

Stop Diffusion

Required permissions: control_server

The **Stop Diffusion** menu item stops the server when clicked upon.

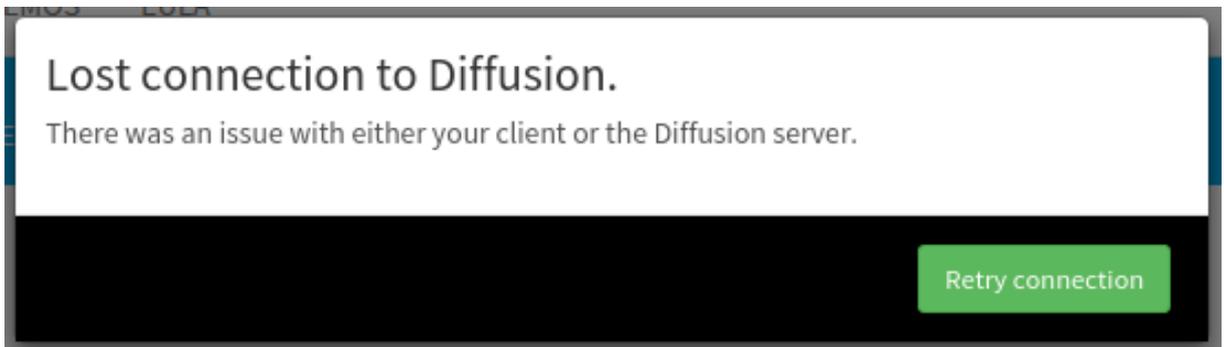


Figure 75: Notification that the Diffusion server has stopped

Going further

Changing the console layout

The console is designed to be extensible and flexible. Users can reorder, edit, create and remove panels. Grab the panel header and drag it to a new location as desired. Click the trash-can icon to remove the icon – with verification

DIFFUSION DETAILS  	
Version	5.6.0_dev
Uptime	3 minutes 42 seconds
Start date	Thu Aug 13 14:27:48 BST 2015
Licence end	Mon Jul 09 14:23:04 BST 2018

Figure 76: The default Diffusion Details panel

Click on the spanner or wrench icon to configure the panel.

The image shows a mobile application dialog box titled "EDIT PANEL" with a close button (X) in the top right corner. The dialog contains three configuration sections: "Panel name" with a text input field containing "Diffusion Details"; "View type" with a dropdown menu currently set to "Table"; and "Header colour" with a dropdown menu currently set to "Blue". At the bottom of the dialog, there are two buttons: a grey "Edit fields" button and a green "Save changes" button.

Figure 77: Editing the properties of the Diffusion Details panel

Panel name' and 'Header color' are self explanatory. 'View type' is a choice of data renderings.

- **Table:** As seen already, this option shows one or more monitoring metrics in a table of textual values.
- **Line:** Renders one or more numeric metrics as a line graph.
- **Area:** Renders one or more numeric metrics as an overlapping area graph.
- **Single:** Used to visualize a single metric in large text, for metrics that are worth the screen real-estate.

Line and area graphs have an extra two configuration fields: 'Refresh rate (ms)' and 'Max data points'. The latter configures how much data is retained for rendering the graph. The former governs the frequency with which the graph is updated and does not influence the frequency of updates from the server. Historic data is only stored in the browser and refreshing the page loses the stored set of data points.

Hovering the mouse over a graph panel shows the detail of the underlying data point

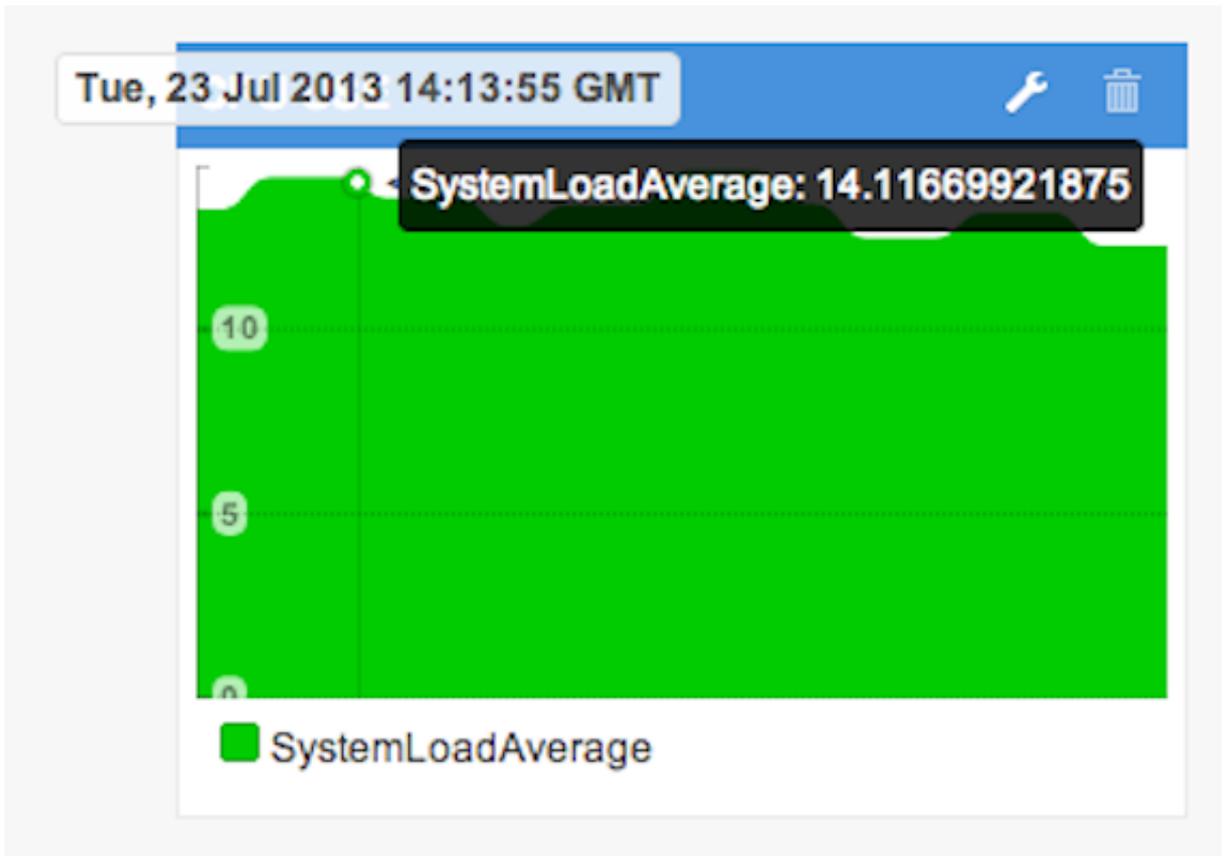


Figure 78: Visualizing the CPU load on a server at a specific time.

Sourcing monitoring metrics

Clicking the 'Edit fields' button presents the user with a Topic Data Fields dialog, where the user nominates one or more topics from where metrics are drawn.

Topics data fields

Name	Topic	Format		
Topic count	Diffusion/Metrics/server/topics/c	Default	↕	✎ 🗑

Figure 79: Editing and adding to the set of topics for this panel

Users of the **Topics** tab have already seen the **Add to Overview** button in the **Topic details** dialog that can shortcut this process.

The default Console layout draws metrics from topics in the Diffusion/MBeans topic tree, however this is not mandatory and solution implementers are free to draw on any suitable topic to reflect their own monitoring needs – including 3rd party topics implemented as part of the solution.

The Diffusion/MBeans topic tree is populated by the JMX adapter which reflects all JMX MBeans as topics. Solution implementers that build custom MBeans to manage their solution can re-use the same MBeans for monitoring purposes.

The Console can draw on features that are themselves optional (Topic and client statistics, for example). If they are disabled, the Console points this out, and request they be enabled in `etc/Statistics.xml`

Production deployment notes

Securing the Diffusion/ topics

The topics in the Diffusion/ tree convey a great deal of power and it is highly probable that bringing a Diffusion based solution to production requires limiting their access to suitable users: for example, users with an IP address in a specific range. Solution implementers can achieve this by implementing an authentication handler.

The default configuration for the console allows users to stop and restart publishers as well as stop the Diffusion server itself. This feature is configured using the properties `console.control.server` and `console.control.publishers` on the Diffusion publisher in `etc/Publishers.xml`.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

DEPRECATED: [Introspector](#) on page 770

An introduction to the Introspector Eclipse plugin.

Related reference

[Statistics](#) on page 754

Diffusion provides statistics about the server, publishers, clients, and topics.

[Logging](#) on page 784

Diffusion uses the Simple Logging Facade for Java (SLF4J) API to log messages from the Diffusion server or from publishers running on the Diffusion server. SLF4J separates the logging of messages in the Diffusion server from the logging framework. This separation enables you to configure an independent back-end implementation to format and write out the log messages.

[Integration with Splunk](#) on page 1049

How to achieve basic integration between Diffusion and the Splunk™ analysis and monitoring application

DEPRECATED: Introspector

An introduction to the Introspector Eclipse plugin.

Note: The Introspector plugin is deprecated and will be removed in a future release. Use the Diffusion console instead. For more information, see [Diffusion monitoring console](#) on page 759.

This section introduces and explains the features of the Diffusion Introspector GUI. This is currently a beta-product, and as such might contain bugs.

The Introspector GUI is a set of Eclipse plugins. This product depends on a minimum of Eclipse version Kepler or later for Java EE Developers which can be downloaded from <http://eclipse.org/downloads/>.

This GUI uses features from the existing system publisher as well as the new Diffusion publisher for which a license must be obtained.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Statistics](#) on page 754

Diffusion provides statistics about the server, publishers, clients, and topics.

[Diffusion monitoring console](#) on page 759

A web console for monitoring the Diffusion server.

[Logging](#) on page 784

Diffusion uses the Simple Logging Facade for Java (SLF4J) API to log messages from the Diffusion server or from publishers running on the Diffusion server. SLF4J separates the logging of messages in the Diffusion server from the logging framework. This separation enables you to configure an independent back-end implementation to format and write out the log messages.

[Integration with Splunk](#) on page 1049

How to achieve basic integration between Diffusion and the Splunk™ analysis and monitoring application

Supported platforms.

The stack of technologies upon which this software is expected to function.

Introspector is tested on version Kepler or later for Java EE Developers and Java HotSpot™ Development Kit 8 (latest update).

Installing from update site

All Diffusion Eclipse plugins are distributed primarily through the Eclipse Update site.

The Eclipse Update site is located at the following URL: <http://download.pushtechology.com/eclipse/5.9>.

To install the software, ensure you have version Kepler or later for Java EE Developers installed. Earlier versions might work, but are not supported. Pick the **Help** menu, click the **Install new software** menu item. From the dialog, choose the **Add ...** button. Complete the dialog as shown below.

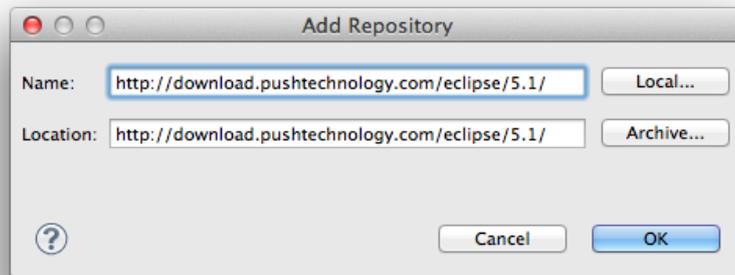


Figure 80: Adding a repository

Click on the **OK** button to go to the Available Software dialog.

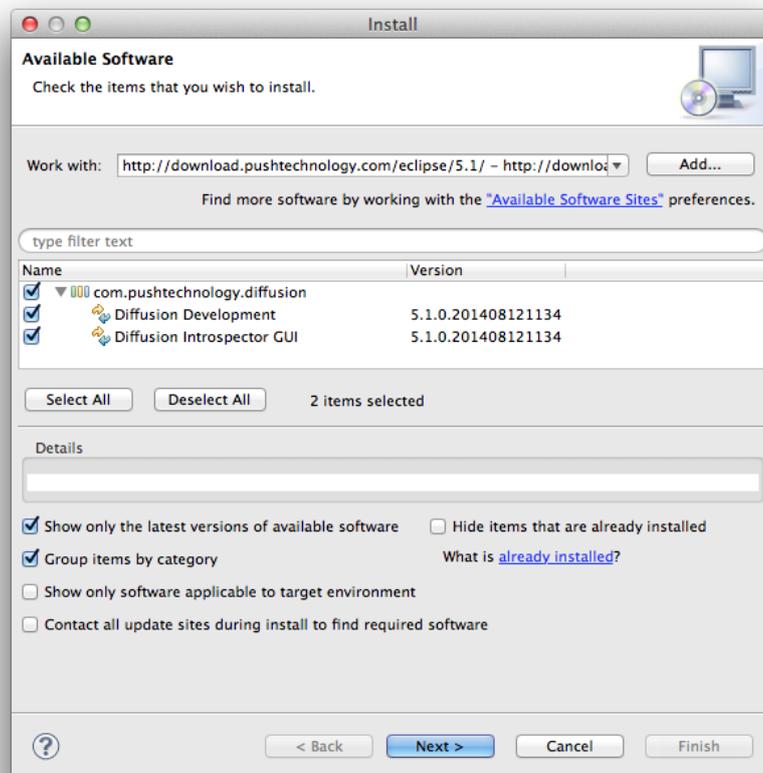


Figure 81: Install dialog

Click the **Next** button. Follow the sequential stream of dialogs. When you are given the dialog shown below, select **I accept the terms of the license agreement** if you agree to the license terms.

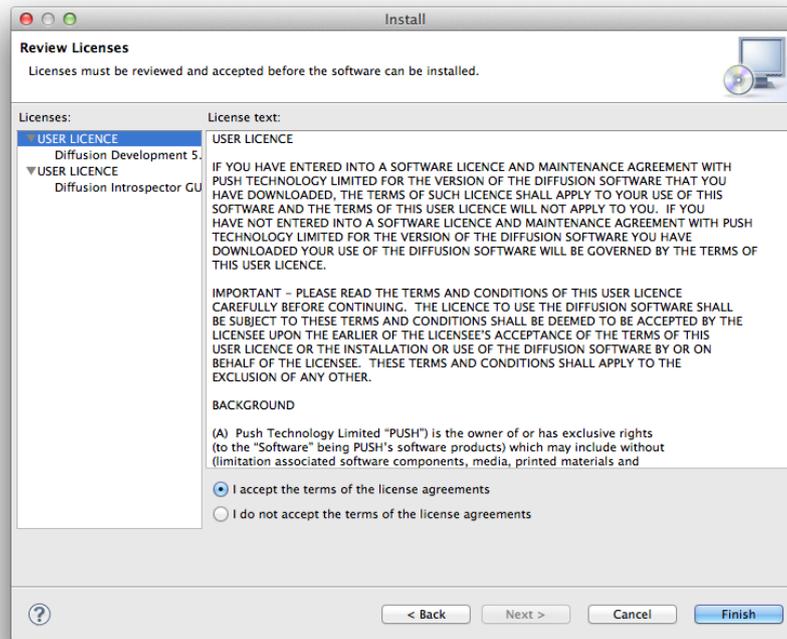


Figure 82: Accept the license agreement

If asked to select the certificate to trust, select **Push Technology Ltd; Development; Push Technology Ltd** and click **OK**.

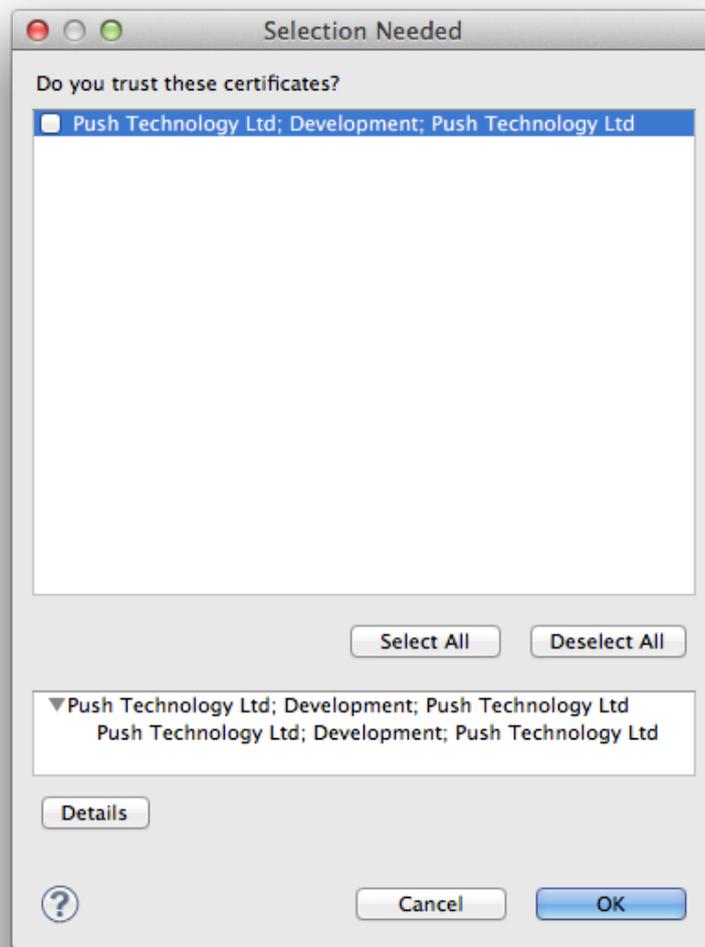


Figure 83: Click OK

When the installation process is done, click the **Restart Now** button.



Figure 84: Restarting

Installing subsequent plugin updates.

Once this process is complete users can install the latest version of the plugins by clicking on the **Help** menu and picking the **Check for updates** menu item.

Uninstalling

Eclipse maintains a registry of what plugins have been installed, and gives users a means of removing plugins from it.

Firstly, open the **About Eclipse** dialog. On macOS this can be found in the **Eclipse** menu, on every other platform – the 'Help' menu.

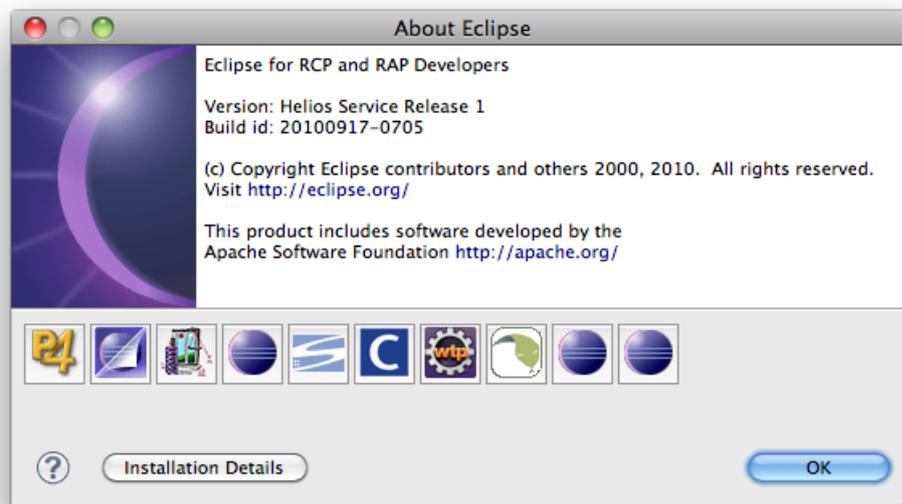


Figure 85: About Eclipse dialog

Click on **Installation details**. Once the **Eclipse Installation Details** dialog appears, click on the **Installed Software** tab to review which plugins are installed.

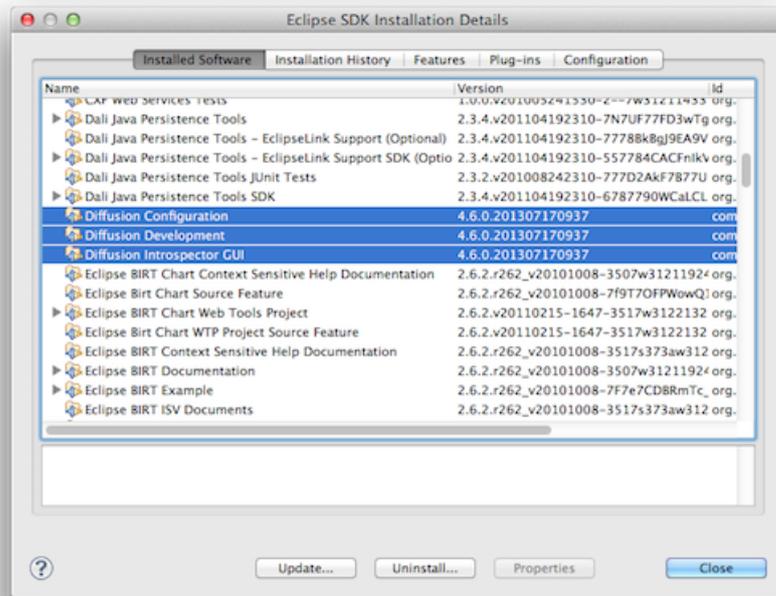


Figure 86: Installed plugins

Once a the software is found, you can uninstall it using the **Uninstall ...** button. Users might be advised to restart Eclipse once the process is complete.

Opening the Diffusion perspective

The Diffusion perspective contains a set of views that relate to Diffusion functional.

Select **Window > Open Perspective > Other ...**. Pick the **Diffusion** item from the list and click on **OK**.

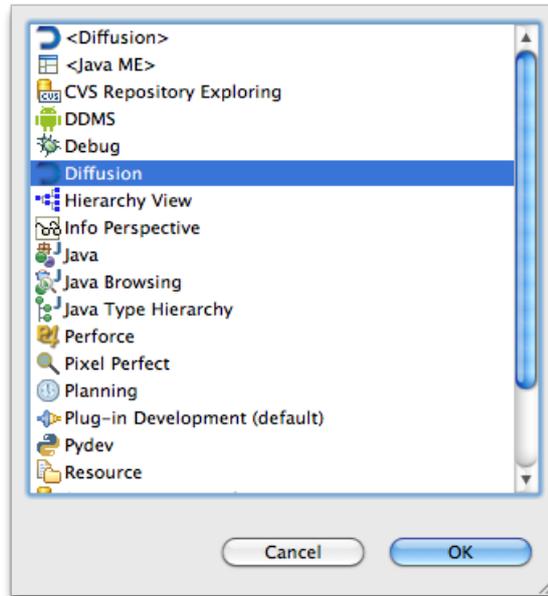


Figure 87: Perspective

Opening the Diffusion perspective presents to the user a set of views related to Diffusion functionality. The same views can be added to other perspectives (such as the Java perspective):

- Select **Window > Show View > Other**
- Find the Diffusion section in the list and pick the views required.

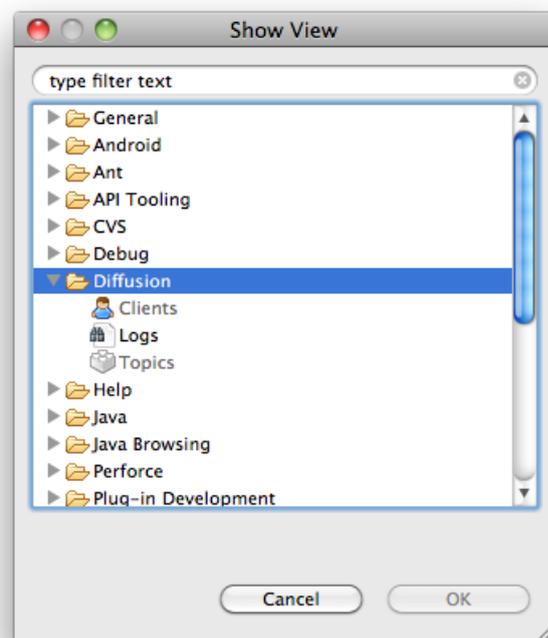


Figure 88: Views

The Diffusion perspective shows the topics and clients views on the left hand side.

Adding servers

To add server details to the plugin click the green plus button in the head of the topic view or client view.



You can click on the **Test** button to test the connection before clicking **OK**.

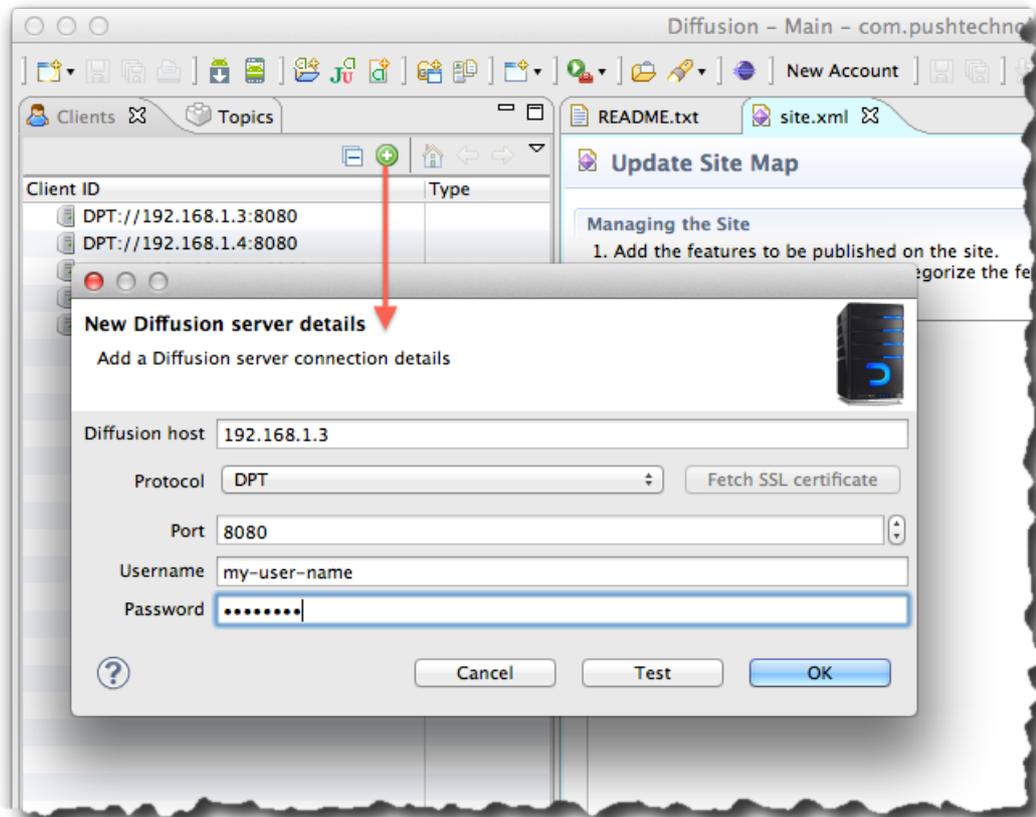


Figure 89: Add a server

If the you want to amend server details, the context menu of closed server (that is, those servers that are not currently connected) holds an **Edit** item that presents the same dialog box.

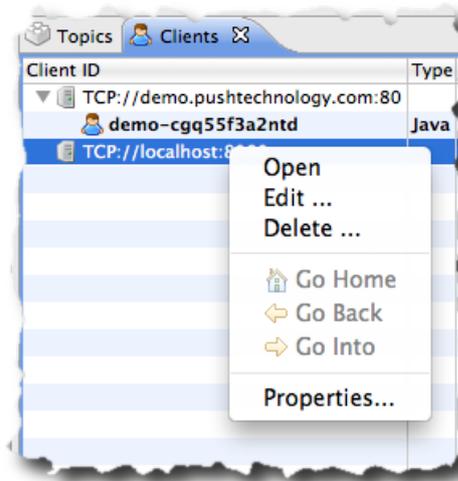


Figure 90: Edit server details

Opening servers

You can view server information through Eclipse.

In either the topics view or the clients view, right-click on a server icon and pick **Open** from the context menu, or double-click on the icon.

The set of columns shown by default show the Diffusion server name (server.name in `etc/Server.xml`) in the value column and the time that the server was started in the **Created** column.

Exploring the topics

You can navigate the tree of topics the same way as any other tree control.

The GUI populates the tree on-demand and asynchronously, so a branch of the tree might be populated some short time after it has been opened. Each topic shows its name, the publisher, the type of any topic-data associated, the date-stamp the topic was created, the date-stamp the topic was last updated and the value of that topic. The last two are populated only if the topic is fetched or subscribed.

The topic tree is live and as new topics are created and old topics deleted so the tree is updated.

Getting topic values

Users can retrieve the current state of the topic by picking **Fetch** from the context menu of a topic. Picking **Fetch recursively** also fetches the values of all topics below this one.

Users can also subscribe to a topic by picking **Subscribe** from the context-menu of a topic. **Subscribe recursively** also subscribes to values of all topics below this one. When the value of a topic changes the **Value** and **Updated** columns flash blue for a second.

To drill into a part of the topic tree they can pick **Go Into** from the topic context menu. The view shows the name of the top-level topic in the view header.

Name	Value	Publisher	Topic Da	Created	Updated
B	103.48	Multi Asset Single Dealer	Single	14:48:08	15:31:15
BY	10.05	Multi Asset Single Dealer	Single	14:48:08	15:31:13
C	-100	Multi Asset Single Dealer	Single	14:48:08	15:30:08
CK	EUR	Multi Asset Single Dealer	Single	14:48:08	15:25:49
CR	10.00	Multi Asset Single Dealer	Single	14:48:08	15:25:49
H	104.30	Multi Asset Single Dealer	Single	14:48:08	15:25:49
L	102.56	Multi Asset Single Dealer	Single	14:48:08	15:25:49
M	103.00	Multi Asset Single Dealer	Single	14:48:08	15:30:08
MD	01/02/2013	Multi Asset Single Dealer	Single	14:48:08	15:25:49
N	BTPS 4 1/4	Multi Asset Single Dealer	Single	14:48:08	15:25:49
O	103.48	Multi Asset Single Dealer	Single	14:48:08	15:31:16
OY	10.05	Multi Asset Single Dealer	Single	14:48:08	15:31:10
PV	104.00	Multi Asset Single Dealer	Single	14:48:08	15:25:49
S	2	Multi Asset Single Dealer	Single	14:48:08	15:31:11
T	FLAT	Multi Asset Single Dealer	Single	14:48:08	15:30:09

Figure 91: View topic values

Click the back button to return to the previous position in the topic tree.

Configuring columns

Columns can be reordered with drag and drop. Column positions are stored between uses. The context menu of the headers lets users toggle individual columns on and off.



Figure 92: Re-order columns

Ping servers

To establish the response time of a remote server pick **Ping** from the context menu of a connected server. The results in milliseconds is shown next to the server name in the **Name** column.

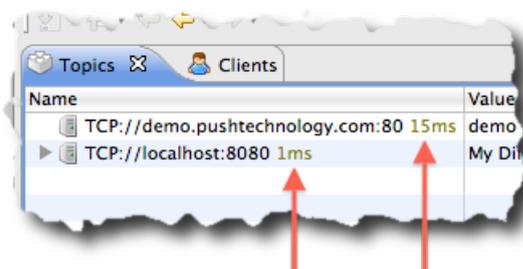


Figure 93: Ping a server

Count topics

Pick **Count topics** from the context-menu of a topic for the server to count the number of topics underneath this one.

The result is shown to the right of the topic name

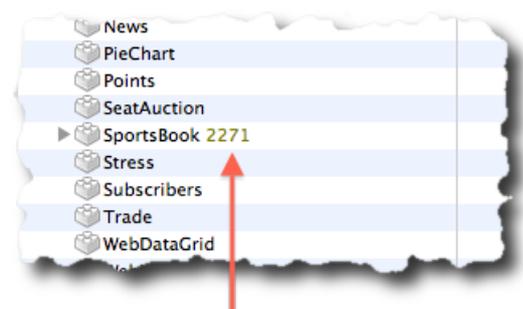


Figure 94: Topic count

Using the clients view

The clients view is a live view of the clients connected to the set of Diffusion servers.

As the Diffusion GUI is also a Diffusion client, it shows up in this list and is shown in bold. A newly connected client is bright blue for 1s, while a disconnecting client is drawn pale gray for 1s before it vanishes.

The properties of a client listed in the following table.

Table 66: Client properties in the Eclipse client view

Type	Flash, Silverlight, WebSocket, iOS, or Android, and so on.
Server	the name of the server this client is connected to
Start time	the time-stamp the client connected to its server
Resolved name	the result of a WHOIS query on the IP address of the remote client
Description	more details from the WHOIS query
Locale	The nation or territory from which this client is connecting
Ping	The time taken to ping the remote client
Alarm	Not currently in use
IP address	The IP address of the remote client
Client reference	The reference/description granted to a client

Ping

Using this feature the user can ping individual clients, the results of which show up in the **Ping** column. All Diffusion clients respond to ping requests from the server.

Further details of a client can be gained from the properties view. Pick **Properties** from the context menu of a client.

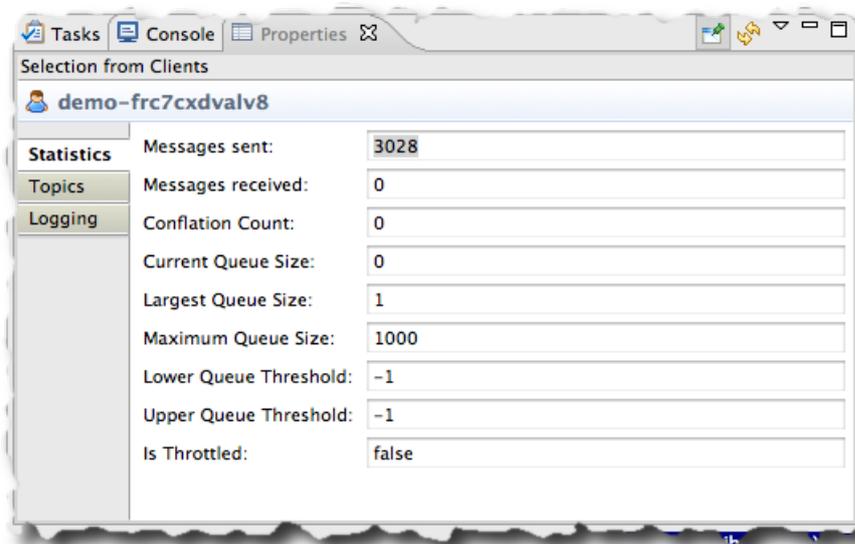


Figure 95: Ping clients

The **Properties** view is part of the Eclipse framework and used in many other non-Diffusion circumstances. Consequently, if the user clicks a new object in another view the **Properties** view might show the properties of that object instead.

Statistics

This shows the number of messages sent, received, the number of messages conflated, the current queue size, the highest queue size, the maximum queue size allowable and throttling thresholds.

Unlike previous views this data is not live, and the refresh button in the view header must be clicked to refresh this view.

Topics

This section shows the topics to which this client is currently subscribed along with any related topic reference. This data is also not live, and must be refreshed manually.

Logging

Access to remote client logs is currently an experimental and optional feature. Those clients that do support it subscribe to the Diffusion/ClientLogs topic.

Using this section of the view, you can retrieve log entries from remote JavaScript clients that are normally sent to the JavaScript console alone.

The JavaScript environment inside a modern browser is memory constrained. This facet of the GUI lets the user set the level at which a log entry is stored and the size of the buffer in which entries are stored before being removed.

Server logs

Diffusion server logs are published through the Diffusion/Logs topic as well as written to log files. The Logs view subscribes to Diffusion to display the log entries as they happen.

Right-click on any open Diffusion server item in the topics or clients view and pick **Logs ...** from the context menu.

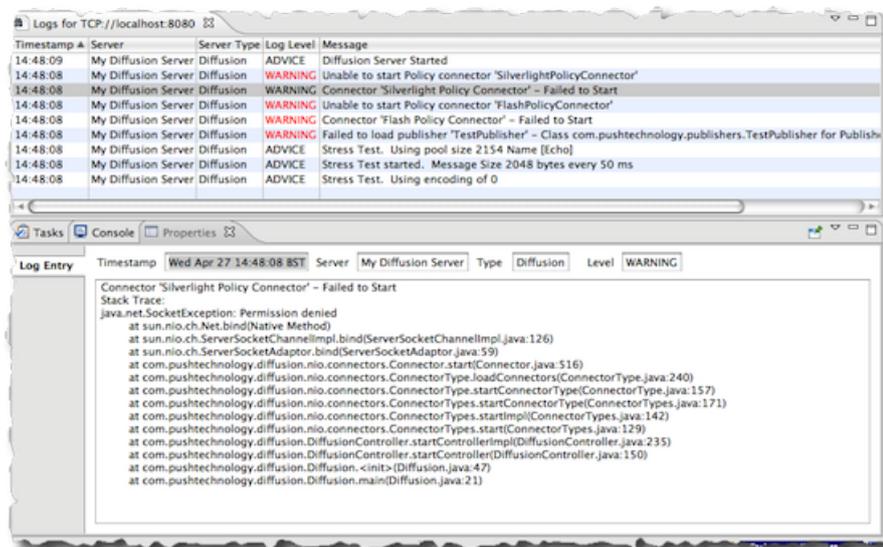


Figure 96: Server log entries

By clicking on the **Timestamp** table header, users can order the log entries by time-stamp. The pull down menu on the right hand side of the view header lets the user filter log entries by the entry log-level.

Note: This does not affect the logging from the server as this is a client-side filter only.

You can get the properties of server log entry from the context menu of an entry. Shown above are the properties of a multi-line log-entry.

Property obfuscator

This dialog is part of the Diffusion perspective and can be used to hide sensitive Diffusion configuration file entries, such as passwords and JMS login credentials.

Select or enter into the dialog the text that you want to obfuscate. All obfuscated entries start with `OB:`. The **Copy** button copies the obfuscated text to the clipboard, ready for pasting back into the configuration file.

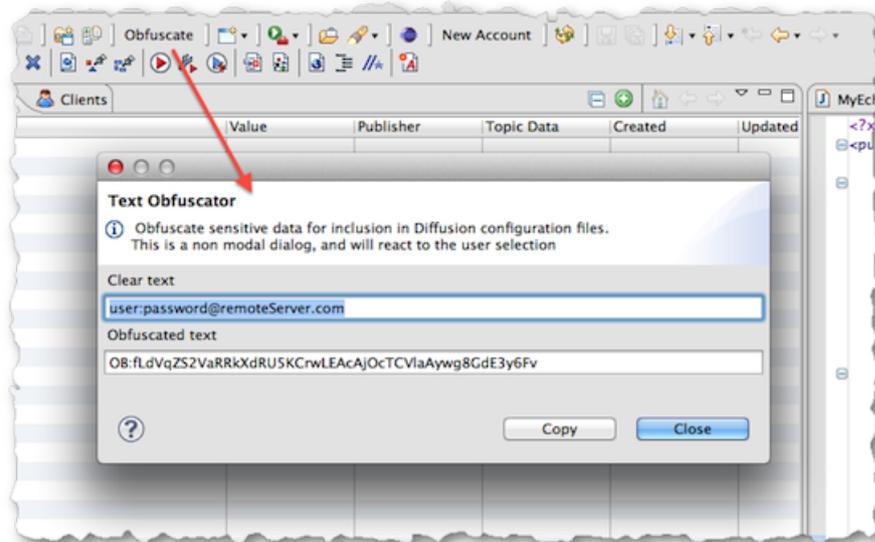


Figure 97: Property Obfuscator dialog

The dialog is modeless, and reacts to the users selection changes you do not need to dismiss the dialog if you have more than one value to obfuscate.

Logging

Diffusion uses the Simple Logging Facade for Java (SLF4J) API to log messages from the Diffusion server or from publishers running on the Diffusion server. SLF4J separates the logging of messages in the Diffusion server from the logging framework. This separation enables you to configure an independent back-end implementation to format and write out the log messages.

Note: The information in this section applies to logging that occurs at the Diffusion server or publishers running on the Diffusion server. Some clients provide logging capabilities. For information about using logging with your Diffusion clients, see the Developer Guide section for the client API you are using.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

DEPRECATED: [Introspector](#) on page 770

An introduction to the Introspector Eclipse plugin.

[Configuring logging on the Diffusion server](#) on page 594

Your Diffusion installation provides a default logging framework and the log4j2 logging framework. Configure the Diffusion server to use your preferred framework.

[Configuring default logging](#) on page 595

To use the default logging, ensure that the Diffusion logging JAR is at `lib/slf4j-binding.jar`. The default logging implementation is already located here when you first install the Diffusion server. Use the `Logs.xml` configuration file to configure the behavior of the Diffusion default logging.

[Configuring log4j2](#) on page 598

To use log4j2, replace the default logging JAR file with the log4j2 JAR file. Use the `log4j2.xml` configuration file to configure the behavior of log4j2.

Related reference

[Statistics](#) on page 754

Diffusion provides statistics about the server, publishers, clients, and topics.

[Diffusion monitoring console](#) on page 759

A web console for monitoring the Diffusion server.

[Integration with Splunk](#) on page 1049

How to achieve basic integration between Diffusion and the Splunk™ analysis and monitoring application

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Log4j2.xml](#) on page 599

Use the `Log4j2.xml` configuration file to configure the behavior of the log4j2 logging framework.

[Logging using another SLF4J implementation](#) on page 600

You can use other implementations of SLF4J for your logging. However, this is not supported for production use.

Logging back-end

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

Default logging framework

The default logging framework provided by Diffusion is configured to log messages out to the console and write them to a file. You can configure the behavior of the default logging framework using the `Logs.xml` configuration file.

For more information, see [Configuring default logging](#) on page 595.

Log4j2 logging framework

Diffusion supports log4j2 as an alternative logging implementation. Log4j2 is a third-party SLF4J implementation provided by the Apache Software Foundation. For more information, see <http://logging.apache.org/log4j/2.x/>.

You can replace the Diffusion default logging with the log4j2 implementation of SLF4J. The log4j2 implementation of SLF4J supports a wide range of appenders and allows fine-grained tuning of logged events.

By default, log4j2 is configured to behave in the same way as the default logging. Change this configuration by editing the provided `log4j2.xml` configuration file.

For more information, see [Configuring log4j2](#) on page 598.

Note: Messages logged using the deprecated LogWriter publisher API are passed directly to the default logging framework, not to log4j2. To use log4j2, you must update your publisher to use SLF4J.

Other logging frameworks

Your Diffusion server can be configured to use any logging framework that implements SLF4J. However, only the default and log4j2 frameworks are supported for production use.

For more information, see [Logging using another SLF4J implementation](#) on page 600.

Note: Messages logged using the deprecated LogWriter publisher API are passed directly to the default logging framework, not to log4j2. To use log4j2, you must update your publisher to use SLF4J.

Related concepts

[Configuring logging on the Diffusion server](#) on page 594

Your Diffusion installation provides a default logging framework and the log4j2 logging framework. Configure the Diffusion server to use your preferred framework.

[Configuring default logging](#) on page 595

To use the default logging, ensure that the Diffusion logging JAR is at `lib/slf4j-binding.jar`. The default logging implementation is already located here when you first install the Diffusion server. Use the `Logs.xml` configuration file to configure the behavior of the Diffusion default logging.

[Configuring log4j2](#) on page 598

To use log4j2, replace the default logging JAR file with the log4j2 JAR file. Use the `log4j2.xml` configuration file to configure the behavior of log4j2.

Related reference

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Log4j2.xml](#) on page 599

Use the `Log4j2.xml` configuration file to configure the behavior of the log4j2 logging framework.

[Logging using another SLF4J implementation](#) on page 600

You can use other implementations of SLF4J for your logging. However, this is not supported for production use.

Logging reference

Messages logged by the Diffusion server are logged at different levels depending on their severity.

Log levels

Diffusion events are logged at different levels of severity. The log levels, ordered from most severe to least severe, are as follows:

Table 67: Log levels

Level	Description
ERROR	Events that indicate a failure.
WARN	Events that indicate a problem with operation.
INFO	Significant events.
DEBUG	Verbose logging. Not usually enabled for production.
TRACE	High-volume logging of interest only to Push Technology Support. Push Technology Support may occasionally ask you to enable this log level to diagnose issues.

Warning: Logging can use considerable CPU resources. In a production environment, enable only significant log messages (INFO and above). Performance degrades significantly when running at finer logging levels as more messages are produced, each requiring processing.

Log format

Log messages output by the Diffusion default logging back-end are output in the following format.

Each log line is made up of a number of fields. All of the fields except for the Exception are formatted on a single line, delimited by pipe (|) characters.

```
yyyy-MM-dd HH:mm:ss.SSS | Level | Thread | Code | Message | LoggerName
Exception
```

If you use log4j2 as your logging back-end it also produces output in this format if you use the provided `Log4j2.xml` configuration file. However, you can edit the configuration file to change the log format. For more information, see [Configuring log4j2](#) on page 598.

Note: Sometimes log messages that are output to the same location as Diffusion messages can be from other products. You can see which messages are Diffusion messages by looking for the message code of the format `PUSH-XXXXXX`. All messages that Diffusion outputs at INFO level or above include this code.

The meaning of each field is described in the following table.

Table 68: Fields included in the logs

Field	Optional or Mandatory	Format/values stable between releases	Description
Time stamp	Mandatory	Yes	The time and date the log event occurred. Asynchronous logging is enabled by default. The server might log a message in a different thread to the one that produced the log event, and at a slightly later time. Consequently, log lines might not be logged in exact time stamp order.

Field	Optional or Mandatory	Format/values stable between releases	Description
			The time stamp is displayed using the timezone configured for the JVM running the server. The date format can be changed in the Server.xml configuration file.
Level	Mandatory	Yes	The log severity, using the SLF4J levels: ERROR, WARN, INFO, DEBUG, TRACE.
Thread	Mandatory	No	The name of the Java thread that logged the event.
Code	Optional	Yes	Diffusion log messages have a unique code. For example, PUSH-000123. For more information, see Log messages on page 790. All messages at that are logged at INFO or above are documented.
Message	Mandatory	No	A natural language description of the event.
Logger name	Mandatory	No	The logger name. Usually the fully qualified name of the Java class that produced the event.
Exception	Optional	No	If the log event has an associated Java Throwable, the exception message and stack trace directly follows the message line.

Optional fields are empty if the log event does not have the information.

The third column indicates whether fields are stable between releases. Where possible, Push Technology will not change the format or values of these fields so they can be relied on for automated log monitoring. The fields not marked as stable are more likely to change between releases, including patch releases.

Log message examples

The following examples show the log format output by the Diffusion default logging back-end. Log4j2 also produces output in this format if you use the provided `Log4j2.xml` configuration file.

Most log messages are formatted on a single line.

```
2016-02-19 14:01:31.199|INFO|main|PUSH-000159|
The maximum message size is 32768 bytes.|
com.pushtechnology.diffusion.DiffusionController
```

If a log event has an exception, the exception message and stack trace directly follows the message line. The exception can span multiple lines.

```
2016-02-19 14:14:54.095|ERROR|main|PUSH-000164|Diffusion Server not
started. |com.pushtechnology.diffusion.api.server.DiffusionServer
com.pushtechnology.diffusion.server.security.persistence.store.StoreException:
Error parsing SystemAuthentication.store
at
com.pushtechnology.diffusion.server.security.persistence.store.systemauthenticac
at
com.pushtechnology.diffusion.server.security.persistence.store.AbstractFileProv
at
com.pushtechnology.diffusion.server.security.persistence.store.AbstractStoreImp
at
com.pushtechnology.diffusion.server.security.authentication.systemhandler.Syste
at
com.pushtechnology.diffusion.server.security.persistence.store.systemauthenticac
at
com.pushtechnology.diffusion.server.security.authentication.AuthenticationManag
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:498)
at java.lang.reflect.Method.invoke(Method.java:498)
```

Log headers

Currently, log files output by the Diffusion default logging back-end start with a special header.

```
2016-02-19 14:14:53.376 : Starting log for Diffusion 5.7.0_01
(Server) 29689@tangerine (2016-02-08 12:22:07)
```

Do not depend on this header. In a future release, this header will be replaced with a standard log message.

This does not apply to log files output by log4j2 or other third-party SLF4J implementations.

Log stopped

When the Diffusion default logging back-end rotates the log files, it outputs the message `Log stopped` at the end of the log file before creating a new log file and continuing to log messages in that new file.

This does not apply to log files output by log4j2 or other third-party SLF4J implementations.

Related concepts

[Configuring logging on the Diffusion server](#) on page 594

Your Diffusion installation provides a default logging framework and the log4j2 logging framework. Configure the Diffusion server to use your preferred framework.

[Configuring default logging](#) on page 595

To use the default logging, ensure that the Diffusion logging JAR is at `lib/slf4j-binding.jar`. The default logging implementation is already located here when you first install the Diffusion server. Use the `Logs.xml` configuration file to configure the behavior of the Diffusion default logging.

[Configuring log4j2](#) on page 598

To use log4j2, replace the default logging JAR file with the log4j2 JAR file. Use the `log4j2.xml` configuration file to configure the behavior of log4j2.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Log4j2.xml](#) on page 599

Use the `Log4j2.xml` configuration file to configure the behavior of the log4j2 logging framework.

[Logging using another SLF4J implementation](#) on page 600

You can use other implementations of SLF4J for your logging. However, this is not supported for production use.

Log messages

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

Related concepts

[Configuring logging on the Diffusion server](#) on page 594

Your Diffusion installation provides a default logging framework and the log4j2 logging framework. Configure the Diffusion server to use your preferred framework.

[Configuring default logging](#) on page 595

To use the default logging, ensure that the Diffusion logging JAR is at `lib/slf4j-binding.jar`. The default logging implementation is already located here when you first install the Diffusion server. Use the `Logs.xml` configuration file to configure the behavior of the Diffusion default logging.

[Configuring log4j2](#) on page 598

To use log4j2, replace the default logging JAR file with the log4j2 JAR file. Use the `log4j2.xml` configuration file to configure the behavior of log4j2.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Logging](#) on page 784

Diffusion uses the Simple Logging Facade for Java (SLF4J) API to log messages from the Diffusion server or from publishers running on the Diffusion server. SLF4J separates the logging of messages in the Diffusion server from the logging framework. This separation enables you to configure an independent back-end implementation to format and write out the log messages.

[Log4j2.xml](#) on page 599

Use the `Log4j2.xml` configuration file to configure the behavior of the log4j2 logging framework.

[Logging using another SLF4J implementation](#) on page 600

You can use other implementations of SLF4J for your logging. However, this is not supported for production use.

PUSH-000001

Attempt #{} to reconnect to JMS Server.

Description

The JMS adapter is currently disconnected from a JMS server and has just failed another attempt to reconnect.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000002

Cannot stop the JMSAdapter.

Description

Something went wrong during initialization of the JMS adapter, and there was an error when trying to clean up and stop the adapter.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging

framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000003

Couldn't find JMSReplyTo destination: '{}'.

Description

An attempt has been made to send a message with a JMSReplyTo destination, but the JMS adapter cannot create the ReplyTo destination on the JMS server.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000004

Discarding non-TextMessage from JMS: '{}'.

Description

The JMS adapter only supports TextMessage types but it received another message type.

Related concepts

[JMS adapter](#) on page 677

The JMS adapter for Diffusion, enables Diffusion clients to transparently send data to and receive data from destinations (topics and queues) on a JMS server.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging

framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000005

Exception from JMS provider '{}'.

Description

The JMS adapter received notification of an exception from the JMS server. Typically, this happens when the connection between the adapter and the server has been terminated.

Related concepts

[JMS adapter](#) on page 677

The JMS adapter for Diffusion, enables Diffusion clients to transparently send data to and receive data from destinations (topics and queues) on a JMS server.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000006

Extra JMS header found, ignoring: '{}'.

Description

JMS messages can have user-defined headers as key/value pairs. Diffusion messages are effectively "value" only. To map user headers from Diffusion to JMS there must be an even number of headers. These headers are artificially grouped as key/value pairs in the JMS message.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging

framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000007

Failed to find topic binding for '{}'.
`{}`

Description

An attempt has been made to send a reply from the JMS adapter back to the JMS server using the DiffusionReplyTo header, but the adapter was unable to determine the corresponding JMSReplyTo destination.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000008

Failed to generate delta message '{}'.
`{}`

Description

A message has been received from the JMS server and a delta message needs to be sent, but the update or publish of the topic data associated with the Diffusion topic failed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000009

Failed to generate ITL message '{}'.
`{}`

Description

An error occurred while generating an ITL message for a client after a message has been received from a JMS queue.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000010

Failed to get message content of message to JMS, topic is '{}'.
`{}`

Description

Failed to read/parse the content of a Diffusion message while translating it to a JMS message for sending.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000011](#)

Failed to send message to JMS destination '{}'.

Description

An error occurred while sending a message from the JMS adapter to the JMS server.

Related concepts

[JMS adapter](#) on page 677

The JMS adapter for Diffusion, enables Diffusion clients to transparently send data to and receive data from destinations (topics and queues) on a JMS server.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000012](#)

Failed to start JMS Adapter.

Description

The JMS adapter was unable to start.

Related concepts

[JMS adapter](#) on page 677

The JMS adapter for Diffusion, enables Diffusion clients to transparently send data to and receive data from destinations (topics and queues) on a JMS server.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000013

Maximum number '{}' of reconnection attempts reached, JMSAdapter disconnected.

Description

The adapter has exhausted the maximum number of configured connection attempts with the JMS server and has given up.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000014

Received TextMessage for unknown destination: '{}'.
'{}' is a placeholder for a value that is not known at the time the message is logged.

Description

A message was received by the JMS adapter from the JMS server, but the adapter was unable to determine which Diffusion topic to map the message to.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000015](#)

JMS Server disconnect detected, attempting reconnection.

Description

The JMS server has been disconnected from the JMS adapter and the adapter is attempting to reconnect.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000016](#)

JMS Server reconnected after '{}' attempts.

Description

The JMS adapter has managed to (re)connect to the JMS server.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000017](#)

Unable to create a subscription to '{}', exception is {}.

Description

The JMS adapter failed to create a subscription to the JMS destination associated with the given topic name.

Related concepts

[JMS adapter](#) on page 677

The JMS adapter for Diffusion, enables Diffusion clients to transparently send data to and receive data from destinations (topics and queues) on a JMS server.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000018](#)

Unable to create new topic '{}'.

Description

The JMS adapter has received a message from the JMS server and needs to create a corresponding Diffusion topic, but was unable to.

Related concepts

[JMS adapter](#) on page 677

The JMS adapter for Diffusion, enables Diffusion clients to transparently send data to and receive data from destinations (topics and queues) on a JMS server.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000019

Unable to extract headers from JMS message: '{}'.

Description

Something went wrong while attempting to convert JMS message headers to Diffusion message headers.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000020

Unable to get payload from JMS message '{}'.

Description

The JMS adapter has received a message from the JMS server, but is unable to read the content.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000021](#)

Unable to restart JMS Server connection.

Description

After an exception from the JMS server, the JMS adapter attempted to reconnect but was unable to.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000022](#)

Exception has been thrown by the `ServerAckListener.messageNotAcknowledged` method of {}.

Description

The specified listener methods has thrown an exception which has been logged but otherwise ignored.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000023](#)

Client Auto Failover failed.

Description

A connection to a server has failed, and the auto failover process has encountered an error.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000024](#)

Mime extension '{}' was already mapped to '{}': overwriting map with '{}'.
'

Description

A configured Mime extension was already mapped to another value but has now been remapped to a new value. This indicates duplicate mime extension values in the mimes configuration.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Mime.xml](#) on page 626

This file specifies the schema for the mime properties.

[PUSH-000025](#)

Connection attempt failed with {} - trying next server.

Description

An attempt to connect to a server has failed but more than one possible server has been specified and so the next is being tried.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000026](#)

Requested receive buffer size for HTTP Proxy connection not allocated, requested: {}, allocated {}.

Description

Diffusion tried to set the size of the TCP receive buffer for a socket connection to an HTTP proxy, but the actual number of bytes allocated was different.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000027](#)

Requested send buffer size for HTTP Proxy connection not allocated, requested: {}, allocated {}.

Description

Diffusion tried to set the size of the TCP send buffer for a socket connection to an HTTP proxy, but the actual number of bytes allocated was different.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000028](#)

An exception has been thrown by the {} method in {}.

Description

An exception has been thrown by an implementation of the specified listener interface. The exception is logged but has no further impact.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000037](#)

Subscription Handler '{}' specified for Routing Topic '{}' - ignored.

Description

A subscription handler has been specified for a routing topic which is not allowed therefore the handler has been ignored.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000038](#)

Exception caught from TopicDeletionListener '{}' topicDeleted method.

Description

An exception has been thrown from a call to a TopicDeletionListener.topicDeleted method. The exception has been logged but has no other effect.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000039](#)

Exception caught from TopicTreeListener '{} {}' method.

Description

An exception has been thrown from a call to a TopicTreeListener method. The exception has been logged but has no other effect.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000040](#)

Whols connection failure limit exceeded - not resolving.

Description

Five attempts to connect to the Whols have failed. There will be no Whols service.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Server.xml](#) on page 563

This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

[PUSH-000041](#)

Failure to connect to Whols provider at {}:{} - will retry 5 times.

Description

Unable to connect to the Whols provider indicated. The connection will be attempted up to five times before giving up.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Server.xml](#) on page 563

This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

[PUSH-000042](#)

Failed to enumerate files in usr.lib directory '{}'.
'{}'.

Description

An error has occurred while traversing the files within the given directory.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000043](#)

Attempted to load duplicate jar file '{}'.

Description

An attempt was made to load the given file more than once.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000044](#)

Entry '{}' in server configuration is not a directory.

Description

The given file system name must identify a directory.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000045](#)

Attempt to queue message '{}' for client {} when ACK timeout has already expired.

Description

An acknowledgement would be queued in response to a message requiring an acknowledgement. However the ACK timeout has already expired. Consider increasing the ACK timeout.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000046](#)

Client {} closing - {}. {}.

Description

A client session was closed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000047](#)

Client {} closing - {}. {}.

Description

The given client is closing due to an exception.

Related concepts

[Common issues when using a load balancer](#) on page 647

There are some configuration options on your load balancer that can cause problems or inefficient behavior in your Diffusion solution.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000049](#)

Error subscribing client {} to topic '{}', publisher '{}'. {}.

Description

An error has occurred while trying to subscribe the specified client to the specified topic.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000052](#)

Exception in ACK listener '{}': messageNotAcknowledged method.

Description

An error occurred while informing a listener that a dispatched message has not been acknowledged.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000054](#)

Client {} tried to fetch invalid topic selector '{}'.

Description

The topic selector expression supplied by a client for a fetch request was invalid.

Related concepts

[Topic selectors in the Unified API](#) on page 61

A topic selector identifies one or more topics. You can create a topic selector object from a pattern expression.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000057](#)

Server is not licensed to accept distributed connections.

Description

This Diffusion server is not licensed to accept connection from other Diffusion servers.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000058](#)

Server is not licensed to accept mobile connections.

Description

The licence does not allow for Diffusion Mobile-to-Server connections.

Related concepts

[License restrictions](#) on page 544

The Diffusion license can include restrictions on how the Diffusion server is used.

Related tasks

[Updating your license file](#) on page 545

You can update your Diffusion license file without having to restart the Diffusion server. Copy the new file over the old and ensure that the timestamp is updated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000061](#)

Failed to subscribe client {} to topic(s) '{}' - Invalid topic name or selector.

Description

The client sent an invalid topic selector pattern for subscription.

Related concepts

[Topic selectors in the Unified API](#) on page 61

A topic selector identifies one or more topics. You can create a topic selector object from a pattern expression.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000062](#)

Failed to schedule future close of client session {}.

Description

It was not possible to schedule the future close of a client session that is awaiting reconnection.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000064](#)

Failed to unsubscribe Client {} from '{}' - Invalid Topic name or selector.

Description

The client sent an invalid topic selector pattern for unsubscription.

Related concepts

[Topic selectors in the Unified API](#) on page 61

A topic selector identifies one or more topics. You can create a topic selector object from a pattern expression.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000065](#)

Failure processing message from server.

Description

An error has occurred at the client end of a connection to a server while processing a message from the server.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000070](#)

Invalid connection from '{}'.

Description

An invalid connection has been attempted from the given address.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Connectors.xml](#) on page 583

This file specifies the schema for the connectors properties.

[PUSH-000072](#)

Connector '{}' created for connections of type {}, listening on {}.

Description

The given connector was created to support the given types of connection.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000074](#)

Connector '{}' received an HTTP connection but does not support HTTP.

Description

Connector has received an HTTP connection attempt but does not support HTTP connections. The connector does not define a valid web server.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Connectors.xml](#) on page 583

This file specifies the schema for the connectors properties.

[PUSH-000075](#)

Unable to start Connector '{}'.

Description

An error occurred while trying to start a connector.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

This file specifies the schema for the connectors properties.

[PUSH-000078](#)

HTTP Processing error, request from '{}'.

Description

An error occurred in the processing of an HTTP request from the given client.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000079](#)

Connector '{}' received a Policy File request which is not supported.

Description

Connector has received a policy file request but does not support such requests. The connector does not have a policy file configured.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000080](#)

Non-SSL connection attempted to connector '{}'
which only supports SSL-type connections.

Description

An attempt has been made to make a non-secure connection to a connector that is configured to only accept secure connections.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Connectors.xml](#) on page 583

This file specifies the schema for the connectors properties.

[PUSH-000081](#)

Connector '{}': no connection - SSL handshake error.

Description

A secure (SSL) connection was attempted but the SSL handshake failed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Connectors.xml](#) on page 583

This file specifies the schema for the connectors properties.

[PUSH-000082](#)

SSL connection attempted but connector '{}' does not support SSL.

Description

An attempt has been made to make a secure (SSL) connection via a connector that does not support SSL. The connector does not define a keystore.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000083](#)

Connector '{}' received a client connection but does not support clients.

Description

The given connector is not configured to receive and handle client connections. For example, the connector might be dedicated to event publisher connections.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000085

{}: Requested input buffer size could not be allocated, requested: '{}' allocated: '{}'.

Description

The receive buffer of the socket was assigned a different amount of memory than requested. This is configured by the input buffer size. The configured input buffer size is a hint to the operating system. Refer to your OS documentation for any socket buffer limits. This can have performance implications.

Additional information

When you change the `input-buffer-size` in the `Connectors.xml` configuration file, this configures the following buffers:

- A buffer in the client multiplexer, which will be of the configured size
- A socket buffer managed by the operating system

Depending on the operating system configuration, the operating system might not provide you with a socket buffer of the specified size and you might be allocated a smaller one.

To ensure that the socket buffer is set to the same size as the input buffer in the client multiplexer, change your OS socket configuration.

On Linux

To see the current maximum size of the input socket buffer, run the following command:

```
sysctl -a | grep rmem_max
```

To set the maximum size of the input socket buffer, run the following command:

```
sudo sysctl -w net.core.rmem_max=number_of_bytes
```

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Connectors.xml](#) on page 583

This file specifies the schema for the connectors properties.

PUSH-000086

{}: Requested output buffer size could not be allocated, requested: {} allocated: {}.

Description

The send buffer of the socket was assigned a different amount of memory than requested. This is configured by the output buffer size. The configured output buffer size is a hint to the operating system. Refer to your OS documentation for any socket buffer limits. This can have performance implications.

Additional information

When you change the `output-buffer-size` in the `Connectors.xml` configuration file, this configures the following buffers:

- A buffer in the client multiplexer, which will be of the configured size
- A socket buffer managed by the operating system

Depending on the operating system configuration, the operating system might not provide you with a socket buffer of the specified size and you might be allocated a smaller one.

To ensure that the socket buffer is set to the same size as the output buffer in the client multiplexer, change your OS socket configuration.

On Linux

To see the current maximum size of the output socket buffer, run the following command:

```
sysctl -a | grep wmem_max
```

To set the maximum size of the output socket buffer, run the following command:

```
sudo sysctl -w net.core.wmem_max=number_of_bytes
```

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Connectors.xml](#) on page 583

This file specifies the schema for the connectors properties.

[PUSH-000087](#)

Connector '{}' received an unidentified connection request [{}] from {}.

Description

An unidentified connection attempt has been made via a connector. The hexadecimal representation of the initial bytes of the connection request are logged in order to aid in identifying the origin.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Connectors.xml](#) on page 583

This file specifies the schema for the connectors properties.

[PUSH-000089](#)

Failed to update Child List Topic Data for Topic '{}'.

Description

An unforeseen error occurred while updating this ChildListTopicData.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000090](#)

An exception has been thrown from CustomTopicDataHandler.{} method.

Description

The custom TopicData implementation threw an exception in the given method.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000091](#)

Client {} sent invalid topic selector pattern '{}'

 to Topic Notify Topic.

Description

A client has sent an invalid topic name or selector pattern to a topic notify topic selection.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000092](#)

An exception has been thrown from `TopicNotifyTopicListener.{} method in '{}'`.

Description

The topic notify topic listener threw an exception in the given method.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000093](#)

Topic Notify Select Request Mode '{}' from Client {} is invalid.

Description

Client has sent an invalid select request mode to a topic notify topic.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000094](#)

`{}` was unable to notify Client `{}` of Topic `'{}`' notification event - Client closing.

Description

A failure has occurred notifying a client of a topic addition, removal or update - the client will be closed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000095](#)

Unrecognised notification type `'{}`'.

Description

The notification request holds unexpected content.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000097](#)

Paged topic '{}' Unable to process command '{}' from client {}.

Description

A paged topic was unable to process a command from a client.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000098](#)

Failure sending a paged topic status message to client {}.

Description

A failure occurred while sending a paged topic status message to the specified client.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000099](#)

Failure sending a paged topic update message to client {}.

Description

A failure occurred while sending a paged topic update message to the specified client.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000100](#)

Command topic data '{}': received invalid command message '{}'.
'

Description

A client has sent an invalid command message to command topic.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000142](#)

Error removing routing data {} from {}.

Description

A failure occurred while trying to remove a topic from a routing topic.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000144](#)

Routing topic failed to subscribe client {} to routing topic '{}' with target topic '{}'.

Description

A routing topic failed to subscribe a client to a topic.

Related concepts

[Routing topics](#) on page 79

A special type of topic, which can map to a different real topic for every client that subscribes to it. In this way, different clients can see different values for what is effectively the same topic from the client point of view.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000145](#)

An exception has been thrown from `ServiceTopicListener.{} method in '{}'`.

Description

The service topic listener threw an exception in the given method.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000146](#)

Service Request error for client `{}` on service `'{}'`, request id `'{}'`, error `'{}'`.

Description

A request from a client to a service topic has failed and it was not possible to report the failure back to the client, possibly because it is closed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000147](#)

Failure in TopicDataListener '{}' topicDataChanged method.

Description

An exception has been thrown by an implementation of the TopicDataListener.topicDataChanged method. The exception is simply logged and has no other effect.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000150](#)

Message '{}' received from '{}' by '{}' has not been handled.

Description

A command topic that does not handle messages from clients has received a message from a client and ignored it.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000151](#)

Unable to get JMX MBean attribute.

Description

It was not possible to retrieve an attribute from a JMX MBean.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000152](#)

Unable to load shutdown hook class '{}': {}.

Description

An error occurred loading the given third party class.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000153](#)

Unable to load startup hook class '{}': {}.

Description

An error occurred loading the given third party class.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000154](#)

Caught JVM shutdown.

Description

The JVM shut down unexpectedly.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000155](#)

No connectors have been configured. Creating default connector listening on port {}.

Description

No connectors have been configured therefore a default connector listening on port 8080 has been created with all default values.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Connectors.xml](#) on page 583

This file specifies the schema for the connectors properties.

[PUSH-000156](#)

No multiplexer configuration. Creating default configuration.

Description

No multiplexer configuration has been found therefore a default multiplexer configuration will be assumed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Server.xml](#) on page 563

This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

[PUSH-000157](#)

No queue definitions configured. Creating DefaultQueue.

Description

No queues have been configured therefore a default queue definition has been assumed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Server.xml](#) on page 563

This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

[PUSH-000158](#)

No thread pools configured. Created new pool definition called '{}'.

Description

No thread pools were configured therefore a default one has been added.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Server.xml](#) on page 563

This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

[PUSH-000159](#)

The maximum message size is {} bytes.

Description

The maximum message size has been established.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Server.xml](#) on page 563

This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

[PUSH-000160](#)

No inbound pool has been configured - using '{}'.

Description

No inbound thread pool has been configured therefore the first configured thread pool has been assumed to define the inbound pool.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Server.xml](#) on page 563

This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

[PUSH-000161](#)

Diffusion - removing publishers.

Description

The server is exiting, and all publishers within are being unloaded.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000162](#)

Running shutdown hook '{}'.
'{}'.

Description

The given third party class is being executed at server shutdown.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000163](#)

Running startup hook '{}'.

Description

The given third party class is being executed at server startup.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000164](#)

Diffusion server not started.

Description

Diffusion server failed to start.

Related concepts

[Starting the Diffusion server](#) on page 636

After you have installed and configured your Diffusion server, you can start it using one of a number of methods.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000165](#)

Diffusion server started.

Description

Diffusion started successfully.

Related concepts

[Starting the Diffusion server](#) on page 636

After you have installed and configured your Diffusion server, you can start it using one of a number of methods.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000166](#)

Diffusion server '{}' starting.

Description

The Diffusion server is starting.

Related concepts

[Starting the Diffusion server](#) on page 636

After you have installed and configured your Diffusion server, you can start it using one of a number of methods.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000167

Diffusion stopped.

Description

The Diffusion server has been stopped.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000168

Diffusion stopping, reason='{}' by administrator='{}'.

Description

Diffusion is processing a shutdown request.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000169](#)

Diffusion - stopping connectors.

Description

Connectors are being stopped during a shutdown of Diffusion.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000173](#)

Exception notifying '{}' of '{}'.

Description

An exception occurred while processing an internal asynchronous event.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000174](#)

Unable to submit event '{}' to '{}' for execution.

Description

A failure has occurred while submitting a notification event for execution.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000175](#)

SHA1 algorithm not found, unable to respond to WebSocket request.

Description

The WebSocket protocol requires the SHA1 algorithm, but it was not found. This can only happen with an incomplete or incompatible Java installation.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000177](#)

Received invalid message '{}' from '{}': {}.

Description

The server module dealing with Introspector clients has received an invalid message.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000178](#)

Exception publishing message on topic '{}'.
'{}': {}.

Description

The server module dealing with Introspector clients caught an exception sending a message to the given client.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000180](#)

Topic '{}' might not be removed by Introspector for client {}.

Description

The given client has attempted to remove a topic that cannot be removed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Diffusion monitoring console](#) on page 759

A web console for monitoring the Diffusion server.

[PUSH-000181](#)

Topic '{}' not removed for client {}: {}.

Description

The given topic was not removed for the given client and reason.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000182](#)

Topic '{}' removed by Introspector client {}.

Description

The given topic was removed for the given client.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000183](#)

Message channel '{}' closed - {}.

Description

A communication error has occurred on a message channel.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000185](#)

Failed to accept connection on connector '{}'.

Description

This can occur when a socket connection has been made to Diffusion, but a failure occurred while initializing it.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000187](#)

Second attempt to write output, try increasing timeout or output buffers '{}'.

Description

Diffusion has tried to obtain a selector for writing and failed and is trying again. If this problem persists, try increasing the write timeout or the output buffer size.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000188](#)

Shutting down selector '{} due to fatal error.

Description

An error has occurred in a selector thread and the selector will be shutdown.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000190](#)

SSL read error.

Description

An error occurred while decoding SSL-wrapped data.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000191](#)

Connector '{}:' - Unable to accept connection: {}.

Description

An exception occurred while attempting to accept a socket connection.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000193](#)

SSL Keystore loaded from '{}'

Description

SSL keystore has been loaded from the specified location.

Related concepts

[Network security](#) on page 724

This section describes how to deploy network security, which can be used in conjunction with data security.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000194](#)

JMX cannot expose '{}' as a topic as dimensions exceed {}.

Description

A limited number of dimensions to an arrays can be mirrored to a topic.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000195](#)

JMX: Cannot register object '{}' at MBean ObjectName '{}'

Description

An error occurred registering the given object with the given JMX ObjectName with the JMX server.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000196](#)

'{}:' failed to update Diffusion license.

Description

An error occurred while attempting to update the license file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000197](#)

Unable to create directory '{}' for the public certificates.

Description

The HTTP service which can be used for updating the license needs to be able to store the license in a directory, but that directory does not exist and cannot be created.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000198](#)

Licence is invalid.

Description

The license is invalid.

Related concepts

[License restrictions](#) on page 544

The Diffusion license can include restrictions on how the Diffusion server is used.

Related tasks

[Updating your license file](#) on page 545

You can update your Diffusion license file without having to restart the Diffusion server. Copy the new file over the old and ensure that the timestamp is updated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000199](#)

License is not valid for this version of Diffusion (license='{}' vs '{}').

Description

The license file does not match this version of Diffusion.

Related concepts

[License restrictions](#) on page 544

The Diffusion license can include restrictions on how the Diffusion server is used.

Related tasks

[Updating your license file](#) on page 545

You can update your Diffusion license file without having to restart the Diffusion server. Copy the new file over the old and ensure that the timestamp is updated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging

framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000200

Product '{} ' not licensed for this address scheme."

Description

A product is not licensed for either IP address or MAC addressing schemes.

Related concepts

[License restrictions](#) on page 544

The Diffusion license can include restrictions on how the Diffusion server is used.

Related tasks

[Updating your license file](#) on page 545

You can update your Diffusion license file without having to restart the Diffusion server. Copy the new file over the old and ensure that the timestamp is updated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000201

Soft user limit ({}) for {} exceeded. Hard limit at {}.

Description

Once the number of connections for a product reaches its soft limit, a warning is emitted. The product might cease to function if this number reaches the hard limit.

Related concepts

[License restrictions](#) on page 544

The Diffusion license can include restrictions on how the Diffusion server is used.

Related tasks

[Updating your license file](#) on page 545

You can update your Diffusion license file without having to restart the Diffusion server. Copy the new file over the old and ensure that the timestamp is updated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000202](#)

Product license expires in {} day(s).

Description

The installed license file is nearing expiration. A new license will be required soon for Diffusion to continue running.

Related concepts

[License restrictions](#) on page 544

The Diffusion license can include restrictions on how the Diffusion server is used.

Related tasks

[Updating your license file](#) on page 545

You can update your Diffusion license file without having to restart the Diffusion server. Copy the new file over the old and ensure that the timestamp is updated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000203](#)

License has expired.

Description

The Diffusion license has expired.

Related concepts

[License restrictions](#) on page 544

The Diffusion license can include restrictions on how the Diffusion server is used.

Related tasks

[Updating your license file](#) on page 545

You can update your Diffusion license file without having to restart the Diffusion server. Copy the new file over the old and ensure that the timestamp is updated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000204](#)

License hard user limit ({} for {} exceeded.

Description

Once the number of client connections has reached or exceeded a the hard limit, the product might be disabled.

Related concepts

[License restrictions](#) on page 544

The Diffusion license can include restrictions on how the Diffusion server is used.

Related tasks

[Updating your license file](#) on page 545

You can update your Diffusion license file without having to restart the Diffusion server. Copy the new file over the old and ensure that the timestamp is updated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging

framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000205](#)

Licence for Introspector has expired.

Description

Licence for Introspector has expired.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000206](#)

Licensor stopping Diffusion. Invalid license.

Description

The Diffusion license is invalid.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000207

License check failed : Unable to check server-side addressing for product [IP] '{}'.
The Diffusion server produces connection summaries.

Description

During the license check, Diffusion attempts to check the server's IP addresses but it was unable to do so.

Related concepts

[License restrictions](#) on page 544

The Diffusion license can include restrictions on how the Diffusion server is used.

Related tasks

[Updating your license file](#) on page 545

You can update your Diffusion license file without having to restart the Diffusion server. Copy the new file over the old and ensure that the timestamp is updated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000208

License check failed : Unable to check server-side addressing for product [MAC] '{}'.
The Diffusion server produces connection summaries.

Description

Diffusion is licensed to run on a server with a specific MAC address, but an error occurred while looking for MAC addresses on the server.

Related concepts

[License restrictions](#) on page 544

The Diffusion license can include restrictions on how the Diffusion server is used.

Related tasks

[Updating your license file](#) on page 545

You can update your Diffusion license file without having to restart the Diffusion server. Copy the new file over the old and ensure that the timestamp is updated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000209](#)

License check failed : Unable to match MAC address for ['{}'].

Description

The product is licensed for a particular MAC address, but the server does not have this MAC.

Related concepts

[License restrictions](#) on page 544

The Diffusion license can include restrictions on how the Diffusion server is used.

Related tasks

[Updating your license file](#) on page 545

You can update your Diffusion license file without having to restart the Diffusion server. Copy the new file over the old and ensure that the timestamp is updated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000210](#)

License check failed : Unable to parse CIDR Address: '{}'.

Description

The product is licensed to run on machines with IP addresses matching a CIDR expression, but the expression could not be parsed. Contact Push Technology for an updated license file.

Related concepts

[License restrictions](#) on page 544

The Diffusion license can include restrictions on how the Diffusion server is used.

Related tasks

[Updating your license file](#) on page 545

You can update your Diffusion license file without having to restart the Diffusion server. Copy the new file over the old and ensure that the timestamp is updated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000211](#)

License version is empty, this is not allowed.

Description

License version is empty, this is not allowed. Contact Push Technology for a replacement license file.

Related concepts

[License restrictions](#) on page 544

The Diffusion license can include restrictions on how the Diffusion server is used.

Related tasks

[Updating your license file](#) on page 545

You can update your Diffusion license file without having to restart the Diffusion server. Copy the new file over the old and ensure that the timestamp is updated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging

framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000212](#)

Log level set to '{}' for '{}'.

Description

The logging level of the specified log file has been changed as indicated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Logging](#) on page 784

Diffusion uses the Simple Logging Facade for Java (SLF4J) API to log messages from the Diffusion server or from publishers running on the Diffusion server. SLF4J separates the logging of messages in the Diffusion server from the logging framework. This separation enables you to configure an independent back-end implementation to format and write out the log messages.

[PUSH-000214](#)

Invalid JMX credentials.

Description

The supplied JMX credentials are incorrect.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000215](#)

Remote JMX management service is disabled.

Description

The remote JMX service is configured not to start.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000216](#)

Remote JMX management service has started. Listening to {}.

Description

The remote JMX service has started.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000226](#)

Multiplexer error while processing client {}.

Description

A multiplexer tried to send messages for a client, but failed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000227](#)

Multiplexer event is delayed as the queue current size is beyond threshold '{}', capacity = '{}'.
{}

Description

A multiplexer cannot queue an event because the queue is full, but is continuing to try to do so.

Additional information

The depth of the multiplexer event queue has exceeded the value of the `max-event-queue-size` element in the `Server.xml` configuration file.

Set this value to 0 to make the size of the queue unbounded. If the value is not set to 0, it must be at least as large as the expected number of concurrent connections to avoid the possibility of deadlock.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000228](#)

Multiplexer event is significantly delayed as the queue current size is beyond threshold '{}', capacity = '{}'.
'{}'.

Description

Because the queue is full, a multiplexer cannot queue an event after a significant amount of time, but it is continuing to try to do so.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000229](#)

Error while handling a multiplexer event.

Description

Error while handling a multiplexer event.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging

framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000230](#)

Failed to schedule event.

Description

A multiplexer tried to schedule an event for some time in the future, but was not able to do so.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000231](#)

Long multiplexer cycle [cycle {}]: processed {} events in {} ms; processed {} clients in {} ms].

Description

A multiplexer processing cycle exceeded the configured notification threshold. On the server, this can be due to subscription processing for many clients. Consider allocating more CPU cores and increasing the number of multiplexers correspondingly.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000232

No multiplexers configured.

Description

No multiplexer definitions have been configured so a default has been assumed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000233

Multiplexer '{}' started.

Description

A multiplexer has started.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000234](#)

Properties loaded: '{}'.

Description

The specified properties file has been loaded.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000235](#)

Properties reloaded: '{}'.

Description

The specified properties file has been reloaded.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000236](#)

Cannot establish JMX attribute '{}' for MBean {}.

Description

The given MBean/Attribute pair is not supported by Diffusion.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000239](#)

Cannot create topic for MBean {}.

Description

It was not possible to create a topic to mirror the content of the given JMX MBean.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000241](#)

Cannot remove topic for MBean {}.

Description

It was not possible to remove a topic that mirrors the content of the given JMX MBean.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000242](#)

JMX configuring property '{}'={}

Description

The JMX Adapter configuration is being applied.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000244](#)

JMX exception interacting with MBean '{}'.

Description

An error occurred creating the tree of topics to represent a JMX MBean.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000245](#)

JMX exception while updating topic '{}'.

Description

An error occurred while polling an MBean attribute that the given topic reflects.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging

framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000246](#)

Caught exception removing notification listener from JMX '{}'.
`{}`

Description

Diffusion is no longer listening to notification from the given MBean, and this has caused a problem.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000247](#)

Cannot find MBean '{}' to listen for notifications.
`{}`

Description

Diffusion is attempting to add a notification to the given MBean, and this has caused a problem.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000249](#)

Cannot send message given notification object {}.

Description

A JMX notification was detected and serialized as a message, but it was not possible to send the message.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000250](#)

Unexpected notification object '{}'.

Description

A JMX notification of an unexpected type was detected.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000251](#)

Unexpected notification type '{}'.

Description

MBeanServerNotification received that was neither about an MBean registering or deregistering.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000252](#)

Unable to construct topics for DiffusionPublisher Log Handler.

Description

The Diffusion publisher log handler encountered an error trying to create its topics.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000253](#)

Exception caught in {} method of authorization handler {}.

Description

An exception has been thrown by the specified method of the user-written authorization handler - the exception has been logged and the requested action rejected.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000254](#)

Authorization manager started with authorization handler class '{}'.

Description

Authorization manager has been started with the specified handler class.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000255](#)

Publisher {} connected to server {} as {}.

Description

A publisher has successfully connected to another Diffusion server as a client.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000256](#)

Deploying publishers compiled for Diffusion version '{}'.
'{}'.

Description

Publishers compiled for the specified version of Diffusion are being deployed.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000260](#)

Diffusion-Version not present in manifest file, not deploying.

Description

The publisher will not be deployed because Diffusion-Version is not present in its DAR manifest file.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000261](#)

Error deploying publisher from file '{}'.
'

Description

An exception has occurred while trying to deploy a publisher from a DAR file.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000262

Error writing file for deployment: {}.

Description

When hot-deploying publishers, the contents of DAR files must be extracted to disk. Check for write permissions to the deployment directory and that there is sufficient free disk space.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000263

Exception caught in publisher '{}' in method '{}'.
{}

Description

A publisher implementation has thrown an exception in the given method. The exception has been logged but otherwise ignored.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000264

Exception caught in publisher '{}' in systemStarted method.

Description

Publishers declared in etc/Publishers.xml can be informed that Diffusion is ready by overriding the systemStarted() method. This message is emitted if an exception is thrown by the publisher's systemStarted() method.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000265

Failed to load publisher '{}'.

```
Failed to load publisher '{}'.

```

Description

Diffusion was unable to load a publisher, probably because of a problem with the publisher's configuration files.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging

framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000266](#)

This DAR file is for Diffusion version {} or greater. Not deploying on Diffusion version {}.

Description

The META-INF/MANIFEST.MF of the DAR file contains the "Diffusion-Version" attribute that specifies with which versions of Diffusion it is compatible. The DAR is marked as incompatible with this version of Diffusion and will not be deployed.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000267](#)

Introspector publisher defined, but not licensed.

Description

Diffusion is attempting to start the publisher which is required for the Introspector, but the license file does not allow the Introspector to be run.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000268

Publisher '{}' is not stoppable, not undeploying.

Description

An attempt has been to undeploy a publisher, but its `isStoppable()` method returns false. The publisher will not be undeployed.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000269

JMX adapter threw exception while starting.

Description

An unexpected exception prevented the JMX adapter from starting.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000270](#)

Configured not to poll MBeans.

Description

The JMX adapter is configured not to poll those JMX MBean attributes with a matching topic subscription.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000272](#)

Polling MBeans every {} milliseconds.

Description

The JMX adapter is configured to configured to poll required MBean attributes for the given frequency.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000273](#)

JMX adapter starting.

Description

The JMX adapter has begun construction.

Related concepts

[JMX](#) on page 735

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000274

DeploymentMonitor: The DAR file '{}' has not been fully undeployed. The publisher(s) {} have not been undeployed. The DAR file will be ignored from now on.

Description

An error occurred during an attempt to undeploy a publisher, which had been deployed using the "hot deploy" feature. This can happen if a DAR contains multiple publishers, but not all of them return "true" from the `isStoppable()` method.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000275

Publisher '{}' received notification of non-acknowledgment of message '{}' by {} clients.

Description

A publisher has received a notification that one or more clients have not acknowledged a message requiring acknowledgment, but the publisher has not implemented the `messageNotAcknowledged` method to handle it.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000282

Removed publisher '{}'.

Description

The given publisher has been removed from the server.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000283

Publisher to server connection '{}' closing due to send failure: {} {}.

Description

A connection between a publisher and a Diffusion server is being closed due to a failure while sending messages to the server.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000284](#)

Publisher '{}' unable to connect to server '{}' - no recovery policy specified.

Description

A publisher server connection has failed to connect and no recovery policy has been specified.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000285](#)

Started publisher {}.

Description

A publisher has been started.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000286](#)

Starting publisher '{}'.

Description

A publisher is starting.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000287](#)

Failed to start statistics handler for publisher '{}'.

Description

A failure has occurred while trying to start the statistics handler for the specified publisher.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000288](#)

Stopped publisher '{}'.

Description

The given publisher has been stopped, and can be restarted.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000289](#)

Stopping Diffusion server due to stop-server-if-not-loaded flag for publisher '{}'.

Description

Diffusion failed to start a publisher which has the "stop-server-if-not-loaded" configuration value set to "true" in Publishers.xml. Diffusion will now be stopped.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Publishers.xml](#) on page 606

This file specifies the schema for the publisher properties.

[PUSH-000290](#)

Failure fetching state of {} from publisher {}.

Description

While handling a fetch request for a topic with no topic data, an error occurred while generating a topic load message.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000292](#)

Publisher server connection {} encountered an unexpected error while sending - closing.

Description

A publisher server connection encountered an unexpected error while sending and will be closed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000293](#)

Publisher '{}' unable to connect to server '{}' - will retry every {} milliseconds.

Description

A publisher server connection cannot be established. The recovery policy indicates that the connection be retried and so an automatic retry will occur at the interval specified.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000294](#)

Failed to read DAR file {}'.

Description

Diffusion tried to deploy a publisher from a DAR file, but the DAR was unreadable. Check file permissions and for corruption with the "jar" tool.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000295

Publisher '{}' not currently deployed, cannot undeploy.

Description

An attempt was made to undeploy a publisher which is not deployed. Either check that this is the case, or that you have the correct spelling of the publisher's name.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000296

DeploymentMonitor: One or more of the publishers in '{}' cannot be stopped. Not undeploying file. The DAR file will be ignored from now on.

Description

An attempt was made to undeploy a hot-deployed publisher, but that publisher does not return "true" from its isStoppable() method.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000298

No queue definitions configured : Adding DefaultQueue.

Description

No queue definitions have been configured therefore a default queue configuration is being assumed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000299

A conflation merge error has occurred in '{}'.
'

Description

An instance of MessageMerger has thrown an exception from its merge method. The exception is logged and no conflation occurs.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000300](#)

Message of size {} is too large for {} which has a limit of {}.

Description

An attempt has been made to send a message to a client that is larger than it is possible to transmit to that client, possibly because of buffer limitations.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000301](#)

Default queue definition {} does not exist. '{}' used.

Description

There is no configured queue definition with the name as specified as the default queue therefore the first configured queue is assumed to be the default.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000302](#)

Default queue definition not configured. '{}' used.

Description

No default queue definition has been specified in the configuration therefore the first configured queue definition has been assumed to be the default.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000303](#)

Queue definition '{}' not known - using default.

Description

The specified queue definition is not known within the configuration therefore the default queue definition has been assumed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000306](#)

Deployment service failed to process HTTP request.

Description

The publisher deployment service failed to return an HTTP response to the deploying client.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000309](#)

REST Failed to process service request.

Description

REST service failed to process a request.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000310](#)

Method '{}}' from Diffusion servlet called, not being processed.

Description

The Diffusion servlet's implementation of the given method does nothing.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000311](#)

Diffusion servlet: {}.

Description

Indicates the startup status of the Diffusion servlet.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000312

Client statistics log name '{}' is not configured.

Description

The log named in the client statistics configuration does not exist in the logs configuration.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Statistics.xml](#) on page 631

This file specifies the schema for the properties defining statistics collection. The statistics are broken into sections: client, topic, server and publisher.

PUSH-000313

Client statistics enabled but no log name supplied.

Description

No log name has been specified in the client statistics configuration.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Statistics.xml](#) on page 631

This file specifies the schema for the properties defining statistics collection. The statistics are broken into sections: client, topic, server and publisher.

[PUSH-000316](#)

Failed to update statistics file '{}'.

Description

It was not possible to write the given statistics file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Statistics](#) on page 754

Diffusion provides statistics about the server, publishers, clients, and topics.

[Statistics.xml](#) on page 631

This file specifies the schema for the properties defining statistics collection. The statistics are broken into sections: client, topic, server and publisher.

[PUSH-000317](#)

Statistics manager failure.

Description

An unexpected error occurred while starting the statistics manager.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Statistics](#) on page 754

Diffusion provides statistics about the server, publishers, clients, and topics.

[Statistics.xml](#) on page 631

This file specifies the schema for the properties defining statistics collection. The statistics are broken into sections: client, topic, server and publisher.

[PUSH-000318](#)

Registering statistics reporter '{}'.
`{}`

Description

A configured statistics reporter of the given name is about to be registered.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Statistics](#) on page 754

Diffusion provides statistics about the server, publishers, clients, and topics.

[Statistics.xml](#) on page 631

This file specifies the schema for the properties defining statistics collection. The statistics are broken into sections: client, topic, server and publisher.

[PUSH-000319](#)

Failed to register statistics reporter '{}'.
`{}`

Description

An attempt to register the named configured statistics reporter has failed - the reporter has therefore been ignored.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000320

Started statistics reporter '{}'.

Description

The named statistics reporter task has started.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000321

Statistics service failed to load statistics reporter '{}'.

Description

A failure has occurred while attempting to load a configured statistics reporter.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Statistics](#) on page 754

Diffusion provides statistics about the server, publishers, clients, and topics.

[Statistics.xml](#) on page 631

This file specifies the schema for the properties defining statistics collection. The statistics are broken into sections: client, topic, server and publisher.

[PUSH-000322](#)

Error starting statistics service.

Description

An error has occurred while starting the statistics service.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000323](#)

Background thread pool caught exception executing '{}'.
'{}' is a placeholder for a message.

Description

A task being executed by the background thread pool has thrown an exception.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000324](#)

Thread pool '{}' could not create new thread.

Description

A thread pool encountered an error while attempting to create a new thread.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000326](#)

Thread pool {} execution of {} failed.

Description

Failure executing a task in the specified thread pool.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000327](#)

Thread pool {} lower limit notification failed.

Description

The specified thread pool was unable to schedule the notification of the lower threshold limit being reached.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000328](#)

Thread pool {} notification of {} to {} failed.

Description

The execution of a notification to a ThreadPoolNotificationHandler failed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000331](#)

Thread pool '{}' queue full - aborting execution of '{}'.

Description

Specified thread pool queue has become full and the rejection policy is to abort the task. Consider increasing the pool maximum size.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000332](#)

Thread pool {} queue full - task running in current thread.

Description

Specified thread pool queue has become full and the rejection policy is to run the task in the current thread, which might be inefficient. Consider increasing the pool maximum size.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000333](#)

Thread pool {} was unable to schedule the execution of notification of rejected execution.

Description

The specified thread pool failed to schedule the notification of a rejected task.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000334](#)

Thread Pool {} upper threshold notification failed.

Description

The specified thread pool failed to schedule notification of upper threshold reached.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000335](#)

Uncaught exception in thread '{}'.

Description

An uncaught exception has been thrown from a specified thread and logged.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000336](#)

Algorithm '{}' not found.

Description

The given cryptographic module could not be found.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000337](#)

File service: [{}] adding virtual hosts {} and mapping to {}.

Description

A virtual host has been added to the web server's file service.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000338](#)

Unable to process alias entry in file '{}': {}.

Description

Diffusion tried to add a new alias to the web server, but failed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000339](#)

HTMLTopicDataSegment: unable to process topic data for {}.

Description

The web server tried to include the contents of a topic's TopicData into the response, but a failure occurred.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000340](#)

HTMLIncludeSegment: unable to populate buffer [{}].

Description

The web server tried to include the contents of another file in the response, but a failure occurred writing to a buffer.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000341](#)

Missing attribute '{}' from DiffusionTag {} [{}].

Description

The given DiffusionTag incorrect omits the given attribute.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000342](#)

Topic '{}' has '{}' TopicData instead of single value TopicData. DiffusionTag (TopicData) [{}].

Description

The given topic lacks SingleValueTopicData, making it impossible to insert here.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000343](#)

Topic {} does not have topic data. DiffusionTag (TopicData) [{}].

Description

The given topic lacks TopicData and therefore any state, making it impossible to insert here.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000344](#)

Cannot find file attribute in DiffusionTag [{}].

Description

The given DiffusionTag incorrectly omits the 'file' attribute.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000345](#)

Cannot load include file '{}' from DiffusionInclude '{}'.
`Cannot load include file '{}' from DiffusionInclude '{}'.
DiffusionInclude: {}`

Description

An error occurred loading the given file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000346](#)

Cannot read specified file for DiffusionTag '{}' [{}].
`Cannot read specified file for DiffusionTag '{}' [{}].
DiffusionTag: {}`

Description

An error occurred loading the given file specified in the DiffusionTag.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000347](#)

Unknown publisher in DiffusionTag {} [{}].

Description

The publisher named in this DiffusionTag is not loaded, or does not exist.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000348](#)

Unknown topic for DiffusionTag (TopicData) {} [{}].

Description

The topic named in this DiffusionTag does not exist.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000349](#)

Unknown type attribute '{} for DiffusionTag [{}].

Description

The type attribute of this DiffusionTag holds an unexpected value.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000350](#)

Web server service cannot process HTML tag '{}' for the publisher '{}'.

Description

An error occurred while processing an HTML embedded tag.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000351](#)

`{}`: Connection rejected as request was missing the HTTP header field '`{}`'.

Description

This connection was rejected as it was missing the given HTTP header. The connection is likely not from a Diffusion client, or an intermediary (firewall/load-balancer) has removed the header.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000352](#)

`{}`: Connection rejected as `[[{}]]` did not match '`{}`'.

Description

The connection from an untrusted host was rejected, in accordance with CORS configuration.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000353](#)

`{}`: Connection rejected as `[{}]` did not match `'{}`'.

Description

The connection was rejected because its WebSocket origin did not match the regular expression stored in the configuration.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000354](#)

`{}`: cors-origin not defined or invalid. Aborting request.

Description

CORS (Cross Origin Resource Sharing) request has been received by the web server client service but no "cors-origin" has been configured or the configured value was invalid. The request has been aborted.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000355](#)

{ } unable to process HTTP Request '{}'.
}

Description

Web server file service was unable to process the given HTTP request.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000356](#)

Client service {}: HTTP processing error on connector '{}'.
}

Description

An error occurred handling an HTTP request.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000357](#)

`{}`: Invalid regular expression '{}' for CORS origin '{}' - feature disabled.

Description

An invalid regular expression was given for the CORS origin configuration.

Related concepts

[Cross domain policies](#) on page 656

Cross domain policies grant permission to communicate with servers other than the one the client is hosted on.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[WebServer.xml](#) on page 617

This file specifies the schema for the web server properties.

[PUSH-000358](#)

`{}`: Invalid regular expression '{}' for WebSocket origin '{}' - feature disabled.

Description

An invalid regular expression was given for the WebSocket origin configuration.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000359](#)

Poll request for client {} from '{}' failed because no matching client session can be found. The client session might have been closed previously. Poll request will be ignored.

Description

The server has received a poll request for an unknown client session. This might be because the client has been closed by the server, or indicate an incorrectly configured load balancer routing an HTTP connection to a server that was not hosting the session. Consider enabling session-stickness for HTTP clients.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000360](#)

FileService [{}]: not adding virtual hosts '{}' mapping to '{}' as it is not a directory.

Description

Failed to add a virtual host to the web server's file service because the configured mapping does not point to a directory.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000361](#)

Unable to service HTTP request for service '{}'.

Description

An HTTP service was invoked, but it failed to process the incoming request.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000362](#)

Started HTTP service for '{}'.

Description

An HTTP service was successfully started.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000363](#)

{ } Unable to create log.

Description

An HTTP service has its own log file defined, but Diffusion was unable to create the logger.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000364](#)

Web server { } failed to instantiate HTTP service '{ }' : { }.

Description

A web server failed to instantiate the named configured HTTP service.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000365](#)

Web server failed to start cache service for virtual host '{}'.
`{}`

Description

An error occurred while starting the cache service for the given virtual host.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000366](#)

Virtual host HTTP request `{}` is invalid.
`{}`

Description

An HTTP request being handled by a virtual host is trying to break out of the virtual root.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000367](#)

Virtual Host HTTP request {} - unable to read file {}.

Description

A failure has occurred trying to read the specified HTML file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000368](#)

{}: websocket-origin not defined or invalid. Aborting request.

Description

Web Server Client Service has received a Web Socket request but "websocket-origin" has not been configured or is invalid. The request has been aborted.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000369](#)

Failed to schedule a periodic tidy of the Whols cache.

Description

A failure has occurred while trying to schedule a periodic tidy of the Whols cache.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000370](#)

Whols Provider class instantiation failure calling '{}'.
'{}'.

Description

Unable to instantiate the configured Whols provider. Will use the default Whols provider.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000372](#)

Unable to load GeolIP Database '{}'.

Description

An error occurred while loading or initializing the GeolIP database.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Server.xml](#) on page 563

This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

[PUSH-000373](#)

Whols Service failed to lookup address '{}'.

Description

A failure has occurred in the Whols service when trying to look up the given address.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Server.xml](#) on page 563

This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

[PUSH-000374](#)

Whols service starting with {} thread(s).

Description

The Whols service is starting.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000375](#)

Deprecated connector type RTMP found in Connectors.xml and will be ignored.

Description

Deprecated connector type RTMP found in Connectors.xml and will be ignored.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000376](#)

Deprecated element '{} ' found in {}.xml.

Description

A deprecated element has been found in specified properties file and should be removed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000377](#)

Failed to load XML properties '{} '.

Description

Failed to load XML properties.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000378](#)

Failed to load publisher '{}' configuration from XML properties.

Description

Diffusion was unable to load the configuration for the named publisher from an XML properties file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Publishers.xml](#) on page 606

This file specifies the schema for the publisher properties.

[PUSH-000379](#)

Failed to load subscription validation policy '{}' from file '{}': {}.

Description

A failure has occurred loading an XML subscription validation policy file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[SubscriptionValidationPolicy.xml](#) on page 634

This file specifies the schema for the subscription validation policy.

[PUSH-000380](#)

Failed to parse {} at line {}, column {}: {}.

Description

An XML validation event has occurred while validating an XML property file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000381](#)

Loaded XML {} properties from '{}'.
'{}'.

Description

The specified XML properties have been loaded from the file indicated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000382](#)

Unable to load publisher configuration file {}.

Description

Attempted to load a publisher's configuration file (Publishers.xml), but it cannot be found or is unreadable.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000383](#)

Unable to check server-side addressing for product {} [{}]: {}.

Description

An error occurred while attempting to check server-side addressing in the licence file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

This file specifies the schema for the subscription validation policy.

[PUSH-000386](#)

Failed to load connection validation policy {} from {}.

Description

A failure has occurred loading XML connection validation policy from the specified file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000387](#)

License check failed to match MAC address for '{}'.

Description

Unable to match a MAC address for the specified product.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000388](#)

Failed to save XML properties '{}'.
`{}`

Description

A failure has occurred while saving XML properties.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000389](#)

Server connection '{}' has been added for publisher '{}'.
`{}`

Description

A configured server connection has been successfully added for a publisher.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000390](#)

Failed to add server connection '{}' for publisher '{}'.

Description

SSL keystore file has successfully been loaded for a publisher to server connection.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000391](#)

Server connection '{}' for Publisher '{}' loaded SSL keystore '{}'.

Description

A publisher has created a server connection that has loaded a keystore to establish the SSL context. The keystore is used to securely store key, certificate, and trust information.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000392](#)

Publisher '{}' has lost connection to server '{}' - publisher closing.

Description

A publisher has lost connection to a server and the recovery policy indicates that the publisher must be closed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000393](#)

Publisher '{}' has lost connection to server '{}' - no recovery policy specified.

Description

A publisher has lost connection to a server and no recovery policy has been specified.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000394](#)

Publisher '{}' has lost connection to server '{}' - will retry every {} milliseconds.

Description

A publisher has lost connection to a server and the recovery policy is to retry therefore a retry will occur at the specified interval.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000395](#)

Publisher '{}' unable to schedule retry connection to '{}' - publisher stopping.

Description

A failure has occurred attempting to schedule a connection retry between a publisher and a server. The publisher will be closed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000396](#)

Unable to perform substitution within property value '{} ' due to syntax error.

Description

A syntax error has been detected while trying to perform environment variable substitution on a configuration property.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000397](#)

Deploying publishers in DAR file '{} '.

Description

Deploying the publishers in a single DAR file. There can be multiple publishers in a single DAR file. The publishers will be extracted from the DAR file and the additional configuration files loaded.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000399](#)

Directory '{}' does not exist or is not a directory. Using '{}' as a log directory.

Description

The directory of a log definition or the default log directory does not exist or it is not a directory. In this case the temporary file directory is used instead. This is specified by 'java.io.tmpdir'. If the intended directory does exist the problem might be that the relative path is incorrect because the working directory is not what you expect. The simplest way to resolve this is to use absolute file paths to reference log directories. This message might be shown when the log is created and when checking if it needs to be rotated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000400](#)

Unable to add listener when reconnecting to {}.

Description

While reconnecting, the JMS adapter was unable to add a listener for a JMS destination.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000401](#)

Unable to create consumer when reconnecting to {}.

Description

While reconnecting, the JMS adapter was unable to recreate a JMS consumer.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000402](#)

Unable to create session when reconnecting to {}.

Description

While reconnecting and re-establishing the existing JMS consumers, the JMS adapter failed to create a JMS session.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000404](#)

'{}': No poll has been opened by client within poll timeout.

Description

HTTP-based connections require an open polling connection to receive messages. If the client has no open polling connection for a fixed period, the client will be closed as unresponsive.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000406](#)

Application handler threw exception {}. [cid={}].

Description

An application handler threw an exception when called.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000408](#)

A response was received from a peer for a finished or unknown conversation and ignored. [cid={}
response={}]

Description

A response with an unknown conversation ID has been received from a peer. Typically this indicates the conversation has timed out or an exception was thrown earlier in the conversation. It can also be for a session using an earlier client library that does not support reliable reconnection if it sends or receives a response from the server that was queued before reconnection.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000409](#)

Unable to close socket channel '{}'

Description

The socket channel failed to close.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000410](#)

Failed to queue background close of the publisher server connection '{}'.

Description

An attempt to queue the close of the publisher server connection failed, therefore the connection has been closed in the current thread.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000411](#)

Multiplexer undefined. The PublisherServerConnection '{}' might not be connected.

Description

The publisher server connection attempted to send with a null multiplexer. This might be due to the PublisherServerConnection not being connected.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000412](#)

Executor {}: task finished abnormally.

Description

A background task has thrown an exception. See the log for more information.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000413](#)

Received request from client {} for the details of client session {}.

Description

The server has received a client request for details of a client session.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000414](#)

Received request from client {} for the details of client session {}. The client session does not exist.

Description

The server has received a client request for details of an unknown client session.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000415](#)

Received request from client {} to close client {}.

Description

The server has received a request from a client to close a client session.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000416](#)

Received request from client {} to close client {}. The client session does not exist.

Description

The server has received a request from a client to close an unknown client session.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000417](#)

Fetch operation {} for {} was aborted due to missing responses for the following topics: {}.

Description

A fetch operation failed, possibly because of an issue at the publisher or state providing client. Look for earlier messages in the server log.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000420](#)

One or more clients have registered to handle queue events, but none was available for a {} event for client {}.

Description

No client was available to handle the queue event, or one was chosen and failed while processing.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000421](#)

Received request from client {} to set conflation for client {} to '{}'

Description

The server has received a request from a client to change the conflation setting for a client queue.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000422](#)

Received request from client {} to set conflation for client {} to '{}'. The client session does not exist.

Description

The server has received a request from a client to change the conflation setting for an unknown client session.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000423](#)

Received request from client {} to throttle message queue for client {} using the {} policy, limit={}

Description

The server has received a request from a client to change the throttling policy of a client session.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000424](#)

Received request from client {} to throttle message queue for client {} using the {} policy, limit={}. The client session does not exist.

Description

The server has received a request from a client to change the throttling policy of an unknown client session.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000426](#)

{}: command service call failed, reporting error to {}: {}.

Description

An internal service call has failed. An error will be reported to the client session listener.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000428](#)

'{}': Has rejected an attempt to connect from a Unified API client. Unified API clients are not supported by this connector.

Description

The connector has been configured to accept connections only of certain client types. An attempt was made to connect by a client of an unsupported type. Either the connector is configured for the wrong type, the client should have connected to a different connector, or an attempt was made to connect a client that has greater functionality than you allow on this connector.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000429](#)

'{}': Has rejected an attempt to connect from a Classic API client. Classic API clients are not supported by this connector.

Description

The connector has been configured to accept connections only of certain client types. An attempt was made to connect by a client of an unsupported type. Either the connector is configured for the wrong type or the client should have connected to a different connector.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000430](#)

There are currently no controllers for {}.

Description

A control binding has been established, but there are no available controllers.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000431](#)

Rejected attempt by client {} to bind a control handler: {}.

Description

This might indicate that another control group has established a binding, or that the client already has a binding.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000432](#)

Attempt by client {} to register or unregister from an unknown control point: {}.

Description

This is unexpected and possibly indicates the server and client versions are incompatible.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000433](#)

Application handler threw exception {} when called with a response. [cid={} response={}].

Description

An application handler threw an exception when called.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000434](#)

Diffusion finished deploying {} components.

Description

Diffusion deployments completed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000435](#)

Task failure in selector thread '{}'.

Description

A task failed to be executed in a selector thread.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000436](#)

Unable to create file handler for logger '{}'. Log messages will still appear in the console.

Description

A file handler was unable to be created for the requested logger. Messages will continue to be logged to the console, but will not be recorded to their own file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000437](#)

A message has been received with an unknown topic alias '{}'. It cannot be parsed.

Description

When topic aliasing is enabled a short alias is used to replace topic names. This reduces the size of messages sent. A client still needs to know the topic name so the topic name and alias are sent together in the topic load. If a client does not receive a topic load it is unable to handle the deltas with only the alias. Diffusion assumes that a client will receive a topic load message if a load message is created for that topic. If the message is not sent, this error message will be logged.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000439](#)

The authentication manager failed to start because an authentication handler of class {} could not be created.

Description

An instance of a configured authentication handler could not be created.

Related concepts

[User-written authentication handlers](#) on page 143

You can implement authentication handlers that authenticate clients that connect to the Diffusion server or perform an action that requires authentication.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000440](#)

Authentication handler {} threw an exception, denying authentication to {}.

Description

A configured authentication handler failed to process an authentication request.

Related concepts

[User-written authentication handlers](#) on page 143

You can implement authentication handlers that authenticate clients that connect to the Diffusion server or perform an action that requires authentication.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000441](#)

The authorization manager failed to start because an authorization handler of class {} cannot be created.

Description

An instance of a configured authorization handler cannot be created.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000443](#)

Publisher '{}' returned null for fetch of topic '{}', indicating an asynchronous response. This is not supported by the client {}. The fetch response (if any) will be discarded.

Description

The client does not support asynchronous fetch responses from a publisher.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000444](#)

Unable to read aliases file '{}'.
`{}`

Description

The configured aliases file could not be read.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Aliases.xml](#) on page 624

This file specifies the schema for the aliases properties used in a web server.

[PUSH-000445](#)

Discarding Whols request for {} due to backlog. There are currently {} pending requests.
`{}`

Description

A Whols request has been discarded because there is currently a large number of pending requests. The WholsListener will be notified with basic locale information based on the caller's IP.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000446](#)

Unable to read subscription polices from '{}'.
`'{}'`.

Description

The configured subscription policy file could not be read.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[SubscriptionValidationPolicy.xml](#) on page 634

This file specifies the schema for the subscription validation policy.

[PUSH-000447](#)

Failed to load publisher '{}'.
`'{}'`.

Description

Diffusion was unable to register a publisher, probably because of a problem with the publisher's definition.

Related concepts

[Deploying publishers on your Diffusion server](#) on page 640

If you developed a publisher as part of your Diffusion solution, you must deploy the publisher on the Diffusion server for it to run.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000448](#)

An error occurred registering a handler with parameters '{}: {}'.

Description

An internal error occurred registering a handler.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000450](#)

An error occurred unregistering a handler with parameters '{}: {}'.

Description

An internal error occurred unregistering a handler.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000452](#)

A topic update failed because no updater could be found.

Description

A topic update failed because the client has unregistered the updater.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000453](#)

A topic update failed unexpectedly: {}.

Description

A topic update failed. See the log message for further information.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000456](#)

Notifying session {} of error '{}'

Description

A session error has occurred. The session error handler will be called.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000457](#)

More multiplexers configured than available CPU cores ({} vs {}).

Description

Assigning more multiplexers than there are available CPU cores might lead to degraded performance.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Server.xml](#) on page 563

This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

[PUSH-000458](#)

Application classpath entry '{}' in is not a directory. Check user libraries in the server configuration.

Description

The names in the user library configuration must identify directories.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Server.xml](#) on page 563

This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

[PUSH-000459](#)

The configured auto-deployment directory '{}' does not exist.

Description

The configured auto-deployment directory '{}' does not exist.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000460](#)

Class loader failed to enumerate files in directory '{}'.

Description

An error occurred reading the files within the given directory.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000461](#)

Unable to add file '{}' to the class loader.

Description

A server class loader failed to load the given file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000462](#)

Multiplexer '{}' overflowed before start up - event discarded.

Description

A multiplexer queue overflowed before the multiplexer was started.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000464](#)

Update failed for topic '{}'.

Description

An error was encountered while attempting to update a topic.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000465](#)

Connector '{}' has pings disabled. Unresponsive HTTP clients will not be detected.

Description

The server relies on pinging in order to detect unresponsive HTTP clients. Unresponsive clients might go unnoticed if the ping frequency is not set.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Connectors.xml](#) on page 583

This file specifies the schema for the connectors properties.

[PUSH-000466](#)

Message stream '{}' threw exception.

Description

A message stream callback raised an exception when called.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000467](#)

Unable to establish a tunnel through the proxy at {}:{}.

Description

There was a problem in establishing a tunnel through the specified proxy. There might be an issue in reading from or writing to the proxy.

Related concepts

[Connecting through an HTTP proxy](#) on page 250

Clients can connect to the Diffusion server through an HTTP proxy by using the HTTP CONNECT verb to create the connection and tunneling any of the supported transports through that connection.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000468](#)

Topic stream '{}' threw exception.

Description

A topic stream callback raised an exception when called.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000469](#)

Failure to forward message from {} to client {} for topic '{}'.

Description

The server failed to forward a message from a client to another client. The message might have been delivered. See the log for further detail.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000470](#)

The server failed to forward a message from {} to unknown client {} for topic '{}'.

Description

The server failed to forward a message as the target client session does not exist.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000472](#)

Service call to server from session {} failed: {}.

Description

The server rejected a service call due to an error. See the log message for further details.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000473](#)

Service call to client {} failed: {}.

Description

The client rejected a service call due to an error. See the log message for further details.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000474](#)

Failed to deliver missing topic notification for subscription or fetch to selector '{}' by session '{}'.

Description

A request made by the server to a missing topic handler resulted in an error.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000475](#)

The '{}' secret key encryption algorithm is not available.

Description

The specified password encryption algorithm is not available which means that a less secure default mode of password encryption will be used.

Related concepts

[Network security](#) on page 724

This section describes how to deploy network security, which can be used in conjunction with data security.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000476](#)

No authentication handlers are configured. All sessions will be anonymous.

Description

No authentication handlers are configured. All sessions will be anonymous.

Related concepts

[User-written authentication handlers](#) on page 143

You can implement authentication handlers that authenticate clients that connect to the Diffusion server or perform an action that requires authentication.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000477](#)

Configurations that specify only control authentication handlers are invalid.

Description

If a control authentication handler is configured, you must also configure a local authentication handler (such as the system authentication handler) to allow a client that registers a control authentication handler to connect.

Related concepts

[User-written authentication handlers](#) on page 143

You can implement authentication handlers that authenticate clients that connect to the Diffusion server or perform an action that requires authentication.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000478

"{}" at line {} column {}.

Description

An error occurred while parsing security store language commands at the indicated line and column number.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000479

Recovery of {} after a write error failed - store must be manually recovered from backup.

Description

Writing of a store file failed and automatic recovery from backup was attempted but also failed. In this event the store has been disabled and must be recovered manually from the backup file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000480](#)

Recovery of "{}" required as backup "{}" exists.

Description

A store backup file has been found which suggests that recovery of the named store might be required. Inspect the store file and replace it with the backup if necessary, and then delete the backup.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000481](#)

Errors detected while parsing persistent store file "{}".

Description

Errors were detected while parsing the specified persistent store file - a detailed list of the errors will follow in the log.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000482](#)

Error reading persistent store file "{}".

Description

A read error has occurred on a security persistence file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000483](#)

Failure writing to the {} persistence store.

Description

An error has occurred while writing to the named persistence store. The store file could be corrupt and might need to be recovered from a backup.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000484](#)

Failure to forward message from {} to {} for topic {}.

Description

The server failed to forward a message from a client to a client that handles messages for the given topic path. The message might have been delivered. See the log for further information.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000485](#)

{}: backlog during initialization - {} queued events sent so far, {} remain on queue.

Description

A session details dispatcher is taking a while to deliver the backlog of pending events to the control client session.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000486](#)

`{}`: closed.

Description

A session details dispatcher was closed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000487](#)

`{}`: draining queued updates.

Description

A session details dispatcher is starting. It is about to process a backlog of pending events.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000488](#)

{}: initialized.

Description

A session details dispatcher has completed initialization.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000489](#)

{}: initialising. Sending {} initial client notifications.

Description

A session details dispatcher is starting.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000490](#)

Session error handler {} threw an exception.

Description

An application session error handler threw an exception when called.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000491](#)

Session listener {} threw an exception.

Description

An application session listener threw an exception when called.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000492](#)

Failed to register topic update source for path '{}' for session '{}'.

Description

The server rejected a request to register an update source.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000493](#)

Whols Service failed unexpectedly.

Description

The Whols Service failed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000494](#)

Failed to execute will {} for session {} because of insufficient permissions.

Description

A client session will to remove a topic could not be executed because the session lacks the required permissions.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000495](#)

Failed to execute will {} for session {} as the topic does not exist.

Description

A client session will to remove a topic cannot be executed because the topic does not exist.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000496](#)

Unexpected error while sending message.

Description

An error occurred when trying to send a message.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000497](#)

Connector '{}' does not support clients.

Description

A connector has received an attempted connection which indicates that it is a Diffusion client but the connector does not support client connections.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000498](#)

Connector '{}' has received data on {} that has not come from a proxy.

Description

The connector has been configured to require all connections use a proxy protocol but a connection has been made that has not provided proxy information.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000500](#)

Introspector client {} has insufficient permissions for '{}' request.

Description

The specified Introspector client does not have sufficient permissions to perform the request.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000501](#)

Diffusion publisher failed to publish a log message to internal topics, disabling handler.

Description

The Diffusion publisher failed to publish a log message. The log handler has been disabled.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000503](#)

A blocking operation failed because the multiplexers failed to process it within {} milliseconds.

Description

This indicates that the server is severely overloaded or deadlocked.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000504](#)

No control handler found for routing topic '{}', subscription for client {} will be deferred until one is available.

Description

A routing handler must be registered for routing subscriptions to complete.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000506](#)

Flow control pressure {}%.

Description

Flow control back pressure has reached this value.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000507](#)

Flow control off.

Description

Back pressure has reduced, flow control is no longer being applied.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000508](#)

Client {} sent message to unknown topic '{}'

Description

Classic API clients can only send messages to existing topics.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000510](#)

Unable to rotate metrics file '{}'.

Description

The internal FileMetricsWriter was unable to rotate out the specified file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000511](#)

Unable to write metrics to file '{}'.

Description

The internal FileMetricsWriter was unable to write to the specified file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000512](#)

Session property {} not allowed.

Description

An attempt has been made to define a session property with a name that is not allowed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000513](#)

{} - failed to create load message.

Description

An error was encountered while updating topic data.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000514](#)

Cannot parse "server-name/sessionId" expression required for delivery to Diffusion client: {}.

Description

The nominated routing property lacks a valid "server-name/sessionId" expression.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000515](#)

Cannot place JMS session to provider {}.

Description

An error occurred when trying to place a JMS session to the given provider.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000516](#)

Cannot publish message to topic {}.

Description

An error occurred while publishing a Diffusion topic update.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000518](#)

Connected to JMS provider '{}'.
{}

Description

The JMS adapter has connected to the JMS provider with the given name.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000519](#)

Connection attempt to '{}' failed.

Description

It was not possible to connect to the given JMS provider.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000520](#)

Exception closing JMS Connection to {}.

Description

An error occurred while the JMS connection was being closed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000521](#)

Failed to deliver message to client {}.

Description

An error occurred during an attempt to deliver a message to a Diffusion client.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000523](#)

Retrying delivery of message to {}: {}.

Description

Following a failed delivery the delivery re-attempted once.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000524](#)

Returning message '{}' to origin: {}.

Description

A message received from a client could not be delivered to a JMS destination, and is being returned to the sender.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000525](#)

Routing-property {} absent from message.

Description

To relay a message to an individual Diffusion client a "server-name/sessionId" expression must be retrieved from a JMS header or property. The configured header/property is absent from this message.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000527](#)

Cannot establish publication to destination {}: {}.

Description

It was not possible to build a MessageProducer for the given JMS destination.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000528](#)

Connected to JMS provider {} v{}, JMS {}.

Description

Recording JMS provider and version, and JMS API version in use.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000529](#)

Failed to generate snapshot for topic {}.

Description

The attempt to generate a snapshot for distribution through replication failed. See the stacktrace for more information on why the snapshot could not be generated.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000530](#)

A subscription failed to complete within the {} millisecond timeout.

Description

A publisher API subscription operation failed to complete. This indicates the server is heavily loaded.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000531](#)

`Topic.publishMessage()` methods should not be used for topics with topic data, but topic '{}' is of type '{}'. This operation will fail in a future release. This message is logged for the first occurrence only.

Description

Topics with topic data should be updated using the `PublishingTopicData` API because `Topic.publishMessage()` does not update the topic data. Calling `Topic.publishMessage()` for a topic with topic data will not be supported in future releases.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000532](#)

Failed to initialize Diffusion client.

Description

Fatal error initializing the Diffusion client.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000533](#)

Request to remote routing subscription handler for client {} / routing topic '{}' failed.

Description

A request to a remote routing subscription handler failed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000534](#)

Routing subscription for client {} to routing topic '{}' failed because no topic exists with resolved topic path '{}'.
'{}'.

Description

A routing topic failed to subscribe a client to a topic because the handler returned an unknown topic path.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000535](#)

Fan-out connection to primary server at '{}' closed.

Description

A fan-out connection to a primary server has been closed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000536](#)

Fan-out connection to primary server at '{} has failed ({}).

Description

A fan-out connection has failed to connect to the primary server.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000537](#)

Fan-out connection '{}' to primary server at '{}' has been lost.

Description

A fan-out connection to a primary server has been lost.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000538](#)

Fan-out connection will attempt to connect to '{}' again every {} milliseconds.

Description

A fan-out connection to a primary server could not be established or has been lost and will now try to connect again at the specified interval.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000539](#)

Unexpected session error in fan-out client '{}' for primary server at '{}': '{}'.

Description

A fan-out connection has received an unexpected session error.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000541](#)

Fan-out link '{}' closed.

Description

A fan-out link replicated from a primary server has been closed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000542](#)

Fan-out link '{}' started.

Description

A fan-out link replicated from a primary server has been started.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000543](#)

Fan-out started.

Description

Fan-out processing has been started.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000544](#)

Fan-out link '{}' failed to create replicated topic {}.

Description

A failure has occurred creating a replicated topic for a specified fan-out link.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000545](#)

Fan-out link '{}' can not replicate topic {} as it is of type {} which is not supported.

Description

A topic has been encountered within the specific link of a fan-out connection which cannot be replicated as it is of a type that is not supported for fan out. The topic will be ignored and will not be created locally.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000546](#)

Fan-out update of topic '{}' by '{}' has failed.

Description

A failure has occurred whilst updating a fan-out secondary topic.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000547](#)

Failed to close selector '{}'.

Description

An error occurred closing a selector.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000548](#)

Failed to interrupt thread for selector '{}'.
`{}`

Description

A selector thread could not be stopped cleanly.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000549](#)

Stopping selector thread '{}'.
`{}`

Description

A selector thread is being stopped.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000550](#)

Both rotate-daily element and rotation-period attribute are defined. Only the rotation-period will be used.

Description

Both rotate-daily element and rotation-period attribute are defined in the Log.xml. The rotation-period attribute overrides the daily-rotate element.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000551](#)

rotate-daily is deprecated. Use the log attribute rotation-period instead.

Description

Warning message that the rotate-daily element has been deprecated and that the rotation-period attribute should be used.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000552](#)

Remote JMX management service could not be started.

Description

The remote JMX management service was not started correctly. The rmiConnectorServer is not active.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000553](#)

Multiplexer blocked because it has a maximum-sized batch of {} notifications to deliver to the publisher(s) {}.

Description

A multiplexer has encountered a severe backlog dispatching publisher notifications. The notifications will be delayed. This message indicates the publishers cannot keep up with the rate of notifications. The server may be overloaded or the publisher may be blocked.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000554](#)

Delayed dispatch of {} notifications because the notification queue for publisher '{}' is full.

Description

A publisher is failing to keep up with the rate of multiplexer notifications. The server may be overloaded or the publisher may be blocked.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000555](#)

Outbound message queue for '{}' is full. Queue details: {}.

Description

The number of messages in the outbound queue for a session has reached the configured limit. The session will be closed. This can indicate problems with the network performance or the receiving process.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000557](#)

`{}: closed.`

Description

A session properties dispatcher was closed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000558](#)

`{}: draining queued updates.`

Description

A session properties dispatcher is starting. It is about to process a backlog of pending events.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000559](#)

{}: initialised.

Description

A session properties dispatcher has completed initialisation.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000560](#)

{}: initialising. Sending {} initial client notifications.

Description

A session properties dispatcher is starting.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000561](#)

`PublishingTopicData.publishMessage()` methods called without an update block for topic '{}' of type '{}'. This operation will fail in a future release. This message is logged for the first occurrence only.

Description

`PublishingTopicData.publishMessage()` should be combined with an update to the topic data. Calling this method outside an update block will not be supported in future releases.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000562](#)

Multiplexer configuration using deprecated method. Define multiplexer using `<multiplexer>` instead of `<multiplexers>`.

Description

The `<multiplexers>` element in `Server.xml` is deprecated. Use `<multiplexer>` to configure your multiplexer.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000563](#)

A message could not be sent to client as it is not subscribed to the topic {}.

Description

A classic client can only receive messages on a topic to which it is subscribed. Either the publisher aborted the subscription, or the topic or client has been deleted.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000564](#)

Established fan-out connection '{}' to primary server {}'.

Description

A fan-out connection to the primary server has been established.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000565](#)

Failed to register topic event listener for path '{}' by session '{}'.

Description

The server rejected a request to register a topic event listener.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000566](#)

Daily statistics have been written in the ConnectionCount file.

Description

The server is shutting down - the daily statistics registered so far have been written in the ConnectionCount file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000567](#)

Memory to calculate delta message exceeds limit of {} bytes - the full message will be used.

Description

A binary difference calculation failed because insufficient memory was available. This is a rare condition that only occurs when there are many differences between two huge messages.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000569](#)

Unable to apply the message of type '{}' received from the data grid to topic '{}'.

Description

The server received data from the data grid that it was able to process as a topic message but the message could not be published on its topic. A stack trace describing the problem with publication is provided.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000570](#)

Unable to process data received as a message of type {}.

Description

The server received data from the data grid that it is unable to process as a Diffusion topic message. Either there was a problem with the data sent or a message of the wrong type was sent.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000571](#)

Failed to start required connector, '{}'.
'{}'.

Description

A required connector, in Connectors.xml, has not been started.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000572](#)

Fan-out connection '{}' is removing topics '{}'

Description

A fan-out connection is removing a set of replicated topics.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000573](#)

Fan-out client {} registration for paged topic update registrations has failed : {}.

Description

A fan-out client has been unable to register for paged topic update notifications and therefore no updates to paged topics can be applied.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000574](#)

Fan-out client {} update of paged topic '{}' has failed - no more updates will be applied to this topic.

Description

A fan-out update of a paged topic has failed - no more updates will be applied to the topic.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000575](#)

HTTP poll rejected - Invalid message channel '{}' cannot be cast to HTTPMessageChannel.

Description

HTTP poll attempted on transport that does not support polling.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000576](#)

A third-party SLF4J logger is installed. The Diffusion log configuration will be ignored.

Description

The Diffusion classpath has been modified to use a third-party SLF4J logging library.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000577](#)

No server log has been configured.

Description

The log configuration does not specify a server log. Messages will only be logged to the console (stderr).

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000578](#)

`{}, {}, {} {}`.

Description

Diffusion and Java product versions information.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000579](#)

Failed to register paged notification listener for path `{}` by session `{}`.

Description

The server rejected a request to register a paged notification listener.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000581](#)

Client service {}: Unrecognized HTTP request received on connector '{} from address '{}'. Request: {}.

Description

The server failed to understand an HTTP request.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000582](#)

JMS adapter cannot apply configuration update from {}.

Description

Configuration changes loaded from the JMS configuration file cannot be applied.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000583](#)

JMS adapter configuration file {} cannot be loaded.

Description

The JMS adapter configuration file cannot be loaded.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000584](#)

JMS adapter cannot roll back the configuration change from {}, step {}.

Description

Following a failed configuration change a roll back was attempted which also failed. Behavior of the JMS adapter here on is unknown.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000585](#)

The JMS adapter has reloaded its configuration from {}.

Description

The JMS adapter has reloaded its configuration file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000586](#)

JMS adapter configuration file {} has been removed.

Description

The JMS adapter configuration file has been removed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000587](#)

The value of {} ms configured for '{}' is excessive and has been limited to {} ms.

Description

The maximum value of the connect and write timeouts has been limited. A future version of Diffusion will enforce the new limits by failing to accept configurations with excessive values.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000588](#)

The connection activity monitor has detected that the connection '{}' has been idle and it has been closed, attempting to recover.

Description

Clients can use a connection activity monitor to listen for the system ping sent by the server. If the expected system pings are not received the connection will be closed and the client will enter a recovery state.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000589](#)

SSL channel {} was closed with data still pending.

Description

SSL connection was closed with data still pending.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000590](#)

Host JVM/JDK failed to provide an expected feature {}.

Description

The JVM/JDK failed to behave as expected at runtime. Please check the platform is supported.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000591](#)

Both file-append and a rotation setting have been configured for the log files. File-append will be used.

Description

Both file-append and a rotation setting (either rotate-daily element and rotation-period attribute) have been configured in the Logs.xml configuration file. The logs will be appended, i.e. only one log file will be output.

Additional information

If you want a log to use rotation, set the file-append element for that log to false (or delete the element).

If you set a rotation-period or enable rotate-daily for a log that has file-append set to true, the rotation setting is ignored.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000592](#)

Sending {} message(s) of {} bytes to {} was delayed by {} ms.

Description

The message took an unreasonably long time to be sent. This could be caused by network backpressure, flow-control or application delays.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000593](#)

Cannot get value attribute {} on MBean {}.

Description

An exception was thrown retrieving an attribute from a given JMX MBean.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000594](#)

JMX adapter stopping.

Description

The JMX adapter is stopping and removing its topics.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000595](#)

Unable to look up session {} in the data grid.

Description

A protocol 4 reconnection was attempted. A new identity will be issued to the client. See the exception for more information on the cause of the failure.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000596](#)

Unable to look up session by token in the data grid.

Description

A protocol 5 or above reconnection was attempted and has failed. See the exception for more information on the cause of the failure.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000597](#)

Unable to recover the session {} from the data grid.

Description

During session fail over Diffusion updates the data grid and recovers information from it. This update or recovery did not succeed both the data grid and the session may have stale information. See the exception for more information on the cause of the failure.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000598](#)

Unable to remove sessions from the data grid.

Description

A server failed and the sessions did not fail over within the timeout. The sessions were not removed from the data grid. They will continue to take up memory in the data grid. See the exception for more information on the cause of the failure.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000599](#)

Unable to remove session {} from the data grid.

Description

The session has been closed but not removed from the data grid. It will continue to take up memory in the data grid. See the exception for more information on the cause of the failure.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000600](#)

Unable to store session {} in the data grid.

Description

An attempt to store a new session in the data grid failed. The session is known to a single server. See the exception for more information on the cause of the failure.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000601](#)

Failed to replicate change to session {}.

Description

An attempt to update the data grid with changes to a principal, properties or subscription level of a session failed. The data grid may have stale information. See the exception for more information on the cause of the failure.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000602](#)

Unable to replicate change to session principal, properties or subscription level for session {}. The session could not be found.

Description

An attempt was made to update the data grid with changes to the principal, properties or subscription level of a session but the session was unknown.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000603](#)

Unable to replicate change to topic selections for session {}.

Description

An attempt to update the data grid with topic selections for a session failed. The data grid may have state information. See the exception for more information on the cause of the failure.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000604](#)

Suppressed {} further {} messages.

Description

Repeated log messages have been suppressed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000605](#)

An error occurred while stopping the remote JMX management service.

Description

The remote JMX management service was not stopped correctly.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000607](#)

Authentication failed for connection {}.

Description

An AuthorisationManager rejected a connection attempt.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000608](#)

Validation failed for connection {}.

Description

An connector validator rejected a connection attempt.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000609](#)

Reconnection aborted because server did not receive {} messages from {}.

Description

Reconnection aborted because messages sent from a client session were not recoverable.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000610](#)

Reconnection aborted because {} messages sent by the server were not received by {}.

Description

Reconnection aborted because messages sent to a client session were not recoverable.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000611](#)

Unable to complete reconnection, recovery failed for unknown session {}.

Description

A session could not be reconnected because it was unknown.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000612](#)

Reconnection aborted because {} messages sent to the server were not received by {}.

Description

Reconnection aborted because messages sent to the server were not recoverable.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000613](#)

{} reconnected, but messages may have been lost.

Description

A session has re-establish communication with a server, but messages may have been lost. This can happen when reconnecting to another server in an HA cluster.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000614](#)

Failed to register topic update source for path {} by session {} because of an unchecked exception.

Description

While registering an update source with the distributed update source registry an unchecked exception was thrown. This may have happened on a different member of the cluster.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000615](#)

Failed to send a checkpoint to a new member of the cluster.

Description

A new node attempted to join the cluster but the cluster failed to send the current state of the distributed update source registry to it.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000616](#)

Failed to remove update source {} from distributed update source registry.

Description

While removing an update source from the distributed update source registry an unchecked exception was thrown. This may have happened on a different member of the cluster.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000617](#)

Connection '{}' was closed due to timeout.

Description

A network connection was timed out due to inactivity on the channel.

Additional information

The Diffusion server gives a client connection a limited time to complete its handshake processing. If the network connection takes longer than the timeout, the connection is closed and the Diffusion server PUSH-000617 is logged.

This issue can be caused by the following circumstances:

- Diffusion is heavily loaded.
 - To reduce the number of refused connections, you can increase the connection timeout value in one of the following ways:
 - Update the `connection-timeout` element in the `Server.xml` configuration file to increase the default timeout value for all connectors. In the default configuration, this value is set to 5s. If the value is not defined here, a value of 2s is used.
 - Update the `connection-timeout` element in the `Connectors.xml` configuration file to increase the timeout value for a specific connector. If a timeout value is not defined for a connector, the value set in the `Server.xml` configuration file is used instead.
- Connections are being made by an application that pings or investigates the Diffusion connector ports without using the Diffusion API.

If this is the case, you can suppress the log messages about invalid connections by using the `ignore-errors-from` element in the `Connectors.xml` configuration file to specify the source IP address of the invalid connection.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[Connectors.xml](#) on page 583

This file specifies the schema for the connectors properties.

PUSH-000618

Input queue for inbound thread '{}' of size {} overflowed due to large number of connections. Risk of deadlock.

Description

The queue of work for an inbound thread overflowed. To avoid the risk of deadlock, reconfigure the inbound pool to have a queue at least as large as the number of concurrent connections.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

PUSH-000619

Thread pool {} queue full - blocking calling thread.

Description

The calling thread is blocked until the specified thread pool can accept a new task. Consider increasing the pool maximum size.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000620](#)

Cannot retrieve session properties for {}: {}.

Description

The JMS adapter cannot retrieve the session properties for the given session.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000621](#)

Client {} closing - {} - {}. {}.

Description

A client session was closed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000622](#)

Unsupported service {} requested by peer {}.

Description

The peer has requested an unsupported internal service. This can be due to a version mismatch between client and server. Some services are not unsupported for deprecated network protocols.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000623](#)

Fan-out connection '{}' to primary server at '{}' lost - attempting reconnection.

Description

A fan-out connection has been lost and is now attempting to reconnect.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000624](#)

Connection '{}' closed - {}.

Description

A network communication error occurred. The connection has been closed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000625](#)

Session {} has received a message for path '{}', but has registered no streams that match that path.

Description

A session has received a message from the server that cannot be delivered because the application has not registered any matching streams.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000626](#)

Session {} has received a message for path '{}' but is not using the Messaging feature so has no registered streams that match that path.

Description

A session has received a message from the server that cannot be delivered because the application is not using the Messaging feature.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000627](#)

Failed to redeliver missing topic notification for subscription or fetch to selector '{}' by session '{}' after {} attempts.

Description

Redelivery of a missing topic notification has been canceled after retrying a number of times.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000628](#)

Handler {} callback method threw an exception.

Description

A handler callback raised an exception when called. If the handler was open it has been closed with the `CALLBACK_EXCEPTION` `ErrorReason`. See the log for more information.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000629](#)

Created {} of {} topics.

Description

Logged every 5s by the JMS Adapter if creating all configured topics exceeds that threshold.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000630](#)

Removed {} of {} topics.

Description

Logged every 5s by the JMS Adapter if removing previously configured topics exceeds that threshold.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000631](#)

Failed to propagate missing topic notification for subscription or fetch to selector '{}' by session '{}'
because there is no connection to primary server '{}'

Description

Fan-out propagation of a missing topic notification failed because the primary server is disconnected.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000632](#)

Failed to propagate missing topic notification for subscription or fetch to selector '{}' by session '{}' to primary server {}.

Description

Fan-out propagation of a missing topic notification failed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000633](#)

Failed to redeliver missing topic notification for subscription or fetch to selector '{}' by session '{}' because there is no longer a suitable registered handler.

Description

Redelivery of a missing topic notification has been canceled by the server because there is no longer a registered handler.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000634](#)

'{}': reconnect attempt failed.

Description

An attempt to reconnect has failed. Further attempts to reconnect may be made depending on the reconnection strategy.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000635](#)

'{}': reconnection rejected by server.

Description

The server has rejected the reconnection attempt. No further attempts to reconnect will be made - the session will be closed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000636](#)

'{}': reconnection failed due to timeout.

Description

The timeout to successfully reconnect has been reached. The session will be closed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000637](#)

The Security.store file cannot be found.

Description

The Security.store file was not found, a default security store has been created.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000638](#)

The '{}' file cannot be found.

Description

The store file was not found. A default store has been created.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000640](#)

Started the JMS adapter with configuration {}.

Description

The JMS adapter was started with the given configuration file.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000642](#)

Allocating larger buffer to accommodate a message larger than the default configured input buffer size. Consider increasing the input buffer size for channel {}.

Description

Client has sent a message larger than the input buffer size.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000644](#)

Completed diagnostic report to {}.

Description

A multiplexer diagnostic report has completed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000645](#)

Failed to write diagnostic report to {}.

Description

A multiplexer diagnostic report could not be produced.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000646](#)

Starting diagnostic report to {}.

Description

A multiplexer diagnostic report has started.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000669](#)

Uncaught exception in inbound thread.

Description

An uncaught exception was thrown in an inbound thread.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000672](#)

Shutting down server. Failed to merge Diffusion server into cluster.

Description

While trying to merge a Diffusion server into the cluster a fatal inconsistency was found. The server was shutdown to preserve cluster consistency. This will most likely happen after recovering from a network partition. A new server should be able to join the cluster as usual.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000682](#)

Session {} closed due to {} - This is due to the session failing to respond to a ping from the server.

Description

The frequency of server pings is dictated by the system-ping-frequency element for each connector in Connectors.xml. Consider configuring this value.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

[PUSH-000713](#)

Fan-out connection '{}' changed from '{}' to '{}'

Description

The state of a connection from a fan-out secondary server has changed.

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

[Connection counts](#) on page 1049

The Diffusion server produces connection summaries.

Connection counts

The Diffusion server produces connection summaries.

At one minute past midnight Diffusion creates an entry in the file `logs/ConnectionCount`, and resets the counter.

The value in the third column is the total number of client connections that have been made that day.

The value in the fourth column is the maximum number of concurrent connections that have been active that day.

If you shutdown the Diffusion server, the server updates this file with the client connection information for the day up to the point of shutdown. However, if the Diffusion server is killed instead of shut down, it does not update the file.

An example is shown here:

```
2020-09-27 00:01:40      5.9.24_01      128      31
2020-09-28 00:01:48      5.9.24_01      139      28
2020-09-29 00:01:56      5.9.24_01      177      28
2020-09-30 00:01:05      5.9.24_01      118      41
2020-10-01 00:01:22      5.9.24_01      207      36
2020-10-02 00:01:31      5.9.24_01      188      19
2020-10-03 00:01:41      5.9.24_01      244      44
2020-10-04 00:01:41      5.9.24_01      188      26
2020-10-05 00:01:41      5.9.24_01      195      39
```

Related reference

[Logging back-end](#) on page 785

The work of formatting and writing out messages logged by the Diffusion server and publishers running on the Diffusion server is done by the logging back-end. The logging back-end is a logging framework that is independent of the Diffusion server. Diffusion provides a default logging framework, but you can configure the Diffusion server to use other SLF4J implementations.

[Logging reference](#) on page 786

Messages logged by the Diffusion server are logged at different levels depending on their severity.

[Log messages](#) on page 790

The Diffusion server outputs log messages. Each log message contains an ID, a message, and a description.

Integration with Splunk

How to achieve basic integration between Diffusion and the Splunk™ analysis and monitoring application

About

Splunk is a third-party application from Splunk, Inc., which provides monitoring and analysis of other applications, primarily by parsing their logs and extracting information of interest. The information is displayed through a web interface, which allows the creation of dashboards and alerts on user-defined events. Splunk is available for all major operating systems.

The Diffusion log format is designed to be consistent and to allow for easy parsing by monitoring tools, not limited to Splunk.

Installation

Installation typically takes just a few minutes, see the appropriate section of the [Splunk Installation Manual](#). For simplicity, we assume that Diffusion and Splunk are installed on the same machine.

Basic configuration

This is easier to do with existing log files to import, so configure Diffusion to write log files. To better demonstrate Splunk, set the server log file to TRACE logging in `etc/Logs.xml` and start Diffusion.

```
<!-- Example server log configuration -->
<log name="server">
  <log-directory>../logs</log-directory>
  <file-pattern>%s.log</file-pattern>
  <level>TRACE</level>
  <xml-format>>false</xml-format>
  <file-limit>0</file-limit>
  <file-append>>false</file-append>
  <file-count>1</file-count>
  <rotate-daily>>false</rotate-daily>
</log>
```

On startup, access the Splunk web UI at <http://localhost:8000>. After logging in (and changing the default admin password), choose the **Add data** option.

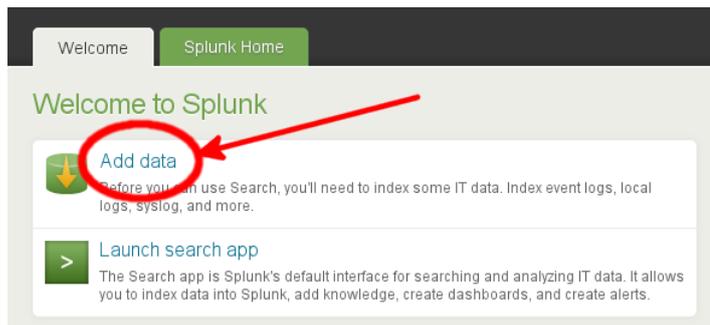
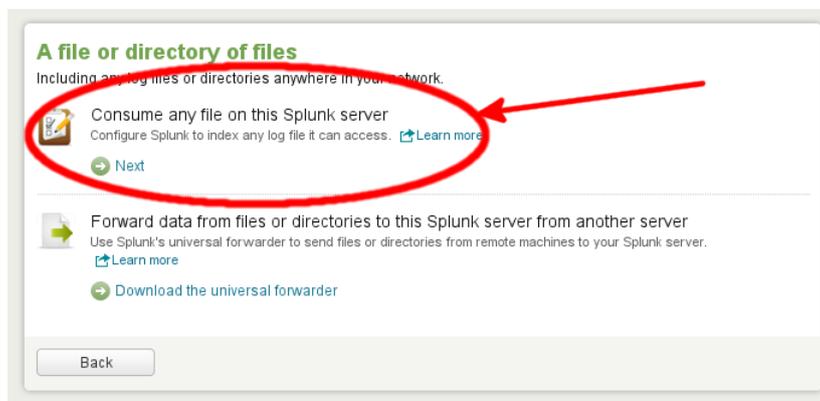


Figure 98: Welcome tab of the Splunk web UI

In the **Add Data to Splunk** screen that follows, choose the link **A file or directory of files** followed by **Consume any file on this Splunk server**.



Splunk might not be able to immediately identify the format of the log files; if this is the case, a dialog box similar to the following is presented. Select **csv** from the existing source types. Diffusion uses a pipe symbol rather than a comma as a separator but this is acceptable to the Splunk CSV parser.

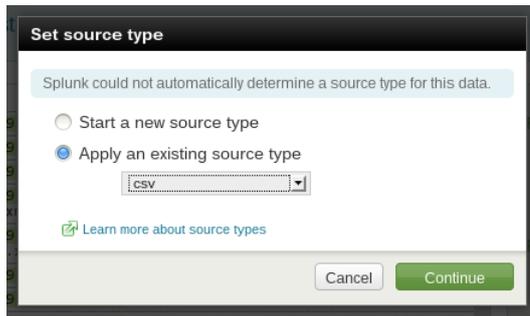


Figure 99: The Splunk Set source type dialog

The next dialog allows you to select the Diffusion `logs/Server.log` file under the **Preview data before indexing** option, which Splunk reads and parses. On the **Data Preview** screen, there are numbered log entries with the timestamp highlighted. This indicates that the log file has been correctly parsed. Accept this, and on the next screen, set the source to be continuously indexing the data. You can leave the parameters in **More settings** at their default values. Once this is done, you have given the new data source a name (for example, Diffusion Server Log) and finally accepted the settings, you can begin searching and generating reports based on the log contents.

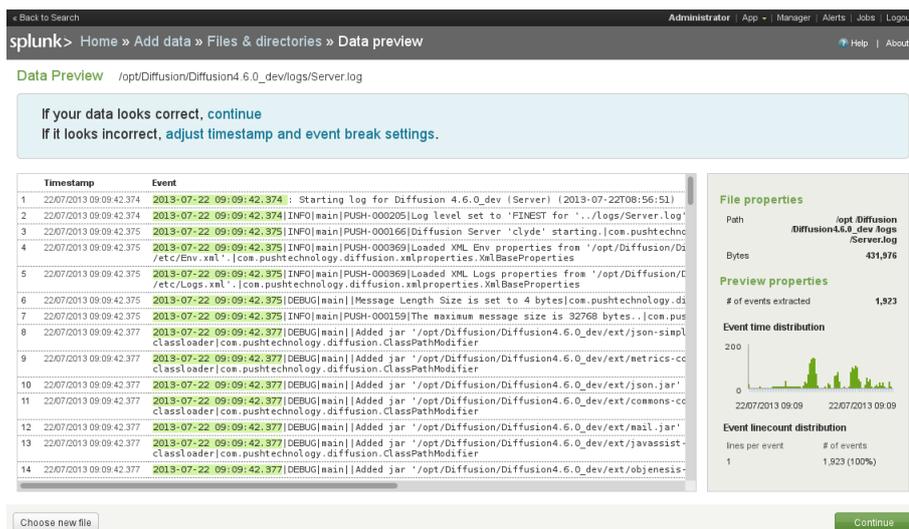


Figure 100: The Data Preview panel

Simple searches

Now we have a data source configured, we can start to execute basic searches.

On the Splunk launch page, select the **Search** option. On the **Search Summary** page that opens, select the Source relating to the file `logs/Server.log` previously imported. The page changes to include the source in the Search area. Additional search terms can be added to the end, for example, “Started Publisher”.

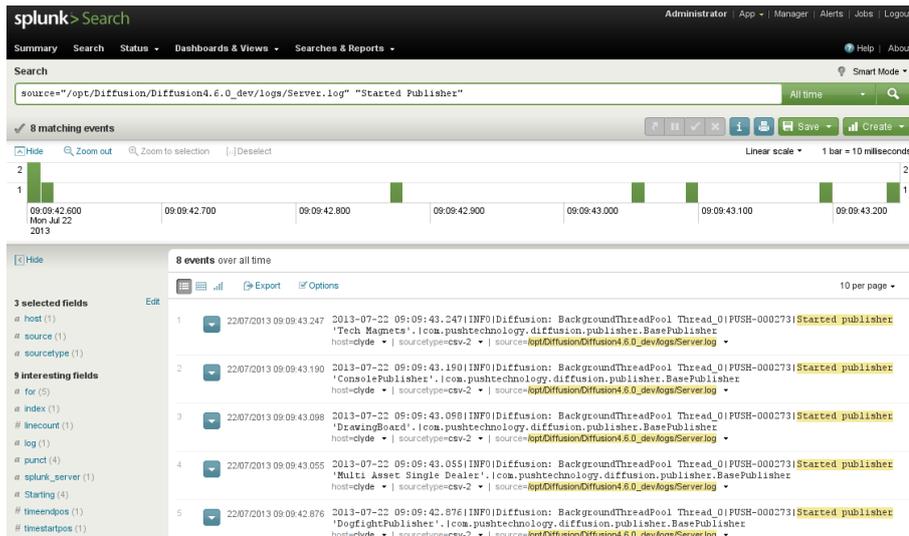


Figure 101: The Splunk search summary panel

Related concepts

[JMX on page 735](#)

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

[DEPRECATED: Introspector on page 770](#)

An introduction to the Introspector Eclipse plugin.

Related reference

[Statistics on page 754](#)

Diffusion provides statistics about the server, publishers, clients, and topics.

[Diffusion monitoring console on page 759](#)

A web console for monitoring the Diffusion server.

[Logging on page 784](#)

Diffusion uses the Simple Logging Facade for Java (SLF4J) API to log messages from the Diffusion server or from publishers running on the Diffusion server. SLF4J separates the logging of messages in the Diffusion server from the logging framework. This separation enables you to configure an independent back-end implementation to format and write out the log messages.

Related information

<http://docs.splunk.com>

Tuning

Aspects of tuning Diffusion for better performance or resilience

This section covers aspects of configuring Diffusion to achieve higher levels of performance and covers some of the more advanced features which enable users to get more out of Diffusion.

Concurrency

Diffusion is a multi-threaded server and utilizes concurrent processing to achieve maximum performance. Java NIO technology is utilized so that a separate thread is not required for each concurrent connection and very large numbers of concurrent connections can be handled.

Because Diffusion is a multi-threaded environment it is necessary to have an understanding of concurrency issues when writing publishers and when configuring Diffusion for best performance.

This section discusses issues of threading and concurrent processing.

Publisher threads

The processing that occurs within the user-written code of a publisher can be executed in different threads as discussed below. Any publisher method can be called at the same time as another. Because of this all publisher processing must be thread safe and it is the user's responsibility to synchronize processing as required. It is recommended that synchronization is maintained at the smallest scope possible to avoid performance bottlenecks.

Inbound threads

Any input that is received on an NIO connection is processed by a thread from the inbound thread pool. This includes most publisher notifications from clients, event publishers or other publishers with the exception of control notifications (such as `initialLoad`, `publisherStarted`) which occurs in the controlling thread.

Note: The act of publishing or sending messages to clients is asynchronous that is to say that the message is queued for the client or clients. Publisher processing is not blocked whilst messages are delivered to clients. For best performance it is recommended that any code executed in the inbound threads is non-blocking (for example, avoid database access, locking, and disk IO as much as possible).

Client notification threads

If a publisher uses client notifications, the publisher has its own dedicated thread to process those notifications.

By default there is one notification thread per publisher, no matter how many listeners are defined. Each event is processed by the thread in the order in which they occur and two client notification event methods are not called concurrently. If the order of such events is not critical, you can specify that a user thread pool is used for client notifications this increasing throughput.

User threads

Publishers or other users of the Diffusion Java API can make use of the Java threads API to schedule tasks for processing of their own in a separate thread of processing.

You can execute any object of a class that implements the `RunnableTask` interface using one of the `ThreadService.schedule` methods. You can request a one-off execution of a task, periodic execution at a given interval or execution according to a schedule. Periodic processing can be important to publishers that pull data updates from elsewhere.

Such tasks issued using the thread service are executed using threads from the background thread pool.

Alternatively, users can define their own thread pools to use using the thread service and execute tasks using these thread pools.

NIO Threads

Each connector that is configured in `etc/Connectors.xml` comprises a connector thread that listens for incoming socket connections, accepts them and registers them with an acceptor thread that handles any incoming data notifications. Message decoding, routing to publishers and appropriate publisher callbacks are all run in the inbound thread pool. Connector and acceptor threads are occupied for the minimum amount of time and are completely non-blocking.

Though performance can be improved in extreme case by adjusting the numbers of these NIO threads, no significant processing occurs within them.

Client multiplexers

A client multiplexer is a separate thread which is responsible for processing messages on the publisher event queue, queuing for clients (conflating if necessary), taking messages from client queues and sending them to the client or clients. A number of these multiplexers can be configured to improve concurrent processing when there are a large number of clients.

The number of multiplexers can be configured. By default, the number of multiplexers is the same as the number of available cores on the host system of the Diffusion server.

Multiplexers typically batch these output messages into output buffers according to the output buffer size configured for the client connectors.

Thread pools

Diffusion maintains a number of configurable thread pools which are used for a number of purposes

For more information, see [Thread pools](#) on page 1059. Thread pools can also be accessed programmatically using the `ThreadService` class of Diffusion server API. Refer to the API documentation for more information about this.

The various types of thread pools are as follows:

Inbound thread pool

This is used to obtain a thread to process any inbound message received on an NIO connection. The maximum number of threads configured for this pool must cater for the maximum required concurrency for incoming requests.

Diffusion does not maintain a separate thread for each client connection but rather passes each inbound request from a connection to the inbound thread pool for processing.

For example, when a client subscribes, the input processing happens on an inbound thread from the pool, the subscribe method and topic loader methods are run in one of these threads.

Connector inbound thread pools

Individual connectors can configure their own separate inbound thread pool to override the use of the default. This cannot be required if you want different behaviors for each connector or if there are a lot of connectors. Due to locking on the inbound thread pool, you get better performance if each connector to have its own inbound thread pool.

Background thread pool

The background thread pool is used for executing scheduled tasks. These tasks can be issued by Diffusion itself or using a publisher using the Java threads API.

Diffusion uses scheduled tasks for various reasons such as:

1. Timing out ACK messages. A scheduled task executes if a message sent to a client is not acknowledged within its required timeout period.
2. Retrying connections. If a Diffusion server cannot connect to another server and there is a retry policy, a scheduled task will be used to retry the connection.

If any publisher uses a lot of scheduled tasks, the number of threads in this pool might have to be increased waiting tasks might queue.

Unlike other types of pool when the specified number of threads are in use, tasks are queued in an unbounded queue.

User thread pools

Within the Java threads API user can define thread pools that can be used for multi-threaded processing.

Buffer sizing

There are a number of places within the configuration of Diffusion where buffer sizes must be specified and getting these right can have a significant impact upon performance.

Connector output buffers

An output buffer size must be configured for each connector.

The output buffer size configured for a connector must be at least as large as the largest message that is expected to be sent to any client connecting through that connector. However, the buffer size can be much larger so that the messages can be batched at the server, which improves performance.

Each connected client is assigned a socket buffer of the specified size if possible. A warning is logged if a smaller socket buffer was allocated than requested.

In addition each client multiplexer has a buffer of the configured size (as a multiplexer writes to only one of its clients at any one time). The multiplexer buffer is used to batch messages from the client queue before writing and, if the socket buffer does end up being smaller than the configured buffer and the throughput is high, the allocated socket buffer size might become a bottleneck.

Getting the correct output buffer size is vital. Make this too small and the Diffusion server does not batch and write messages to clients at optimal rates. Make them too big, extra memory is consumed or messages might time out and cause the client connection to be closed.

If the output buffer size is larger than the TCP output buffer size, this can cause problems if the client is slow consuming. If a slow-consuming client does not clear messages from the TCP buffer fast enough, messages on the connector buffer which are waiting to move to the TCP output buffer can time out. You can avoid this problem by setting the TCP output buffers for your operating system and the connector output buffers for your Diffusion server to the same value. You can also increase your message timeout interval.

Note: For maximum performance, ideally all clients configure their input buffer size to match the connector's output buffer size.

Client output buffers

As at the server, the output buffer sizes in use must be configured for a client.

In the Java client this is specified in the ServerDetails object used to make the connection. As the Java client does not buffer messages, this only has to be large enough to cater for the largest message size that is sent to the server.

Publisher client output buffers

A publisher client (a connection made from a publisher to another Diffusion server) is slightly different from a normal client in that it does queue and buffer messages for sending. It is advantageous to throughput to use a larger output buffer size.

The output buffer size is configured in the server element in `Publishers.xml` or in the `ServerDetails` object depending upon how the connection is being made.

Connector input buffers

Each connector also specifies an input buffer size.

Input buffers receive messages from clients. This buffer must be as large as the largest message expected. If you specify an input buffer size that is less than the maximum message size, the maximum message size is used as the input buffer size.

This size is also used to allocate a receive socket buffer for the client. The socket buffer allocated might actually be less than requested in which case a warning is logged.

For maximum performance, the size used for this buffer must match up with the output buffer size used by clients.

Client input buffers

Clients must also specify the buffer size to use for input.

In the Java client this is specified in the `ServerDetails` object used to make the connection (or possibly `Publishers.xml` for a publisher client connection).

Matching buffer sizes

For optimal throughput it is desirable to match the size of buffers at each end of every connection. The input buffer size used by clients ideally matches the output buffer size at the connector that they connect to. Also the output buffer size specified by clients must match the input buffer size of the connector they connect to.

Note: Because publisher server connections queue and batch messages at both ends, use a separate connector for such connections such that optimal buffer sizing can be achieved.

Message batching

The size of output buffers configured for a connector can be much larger than the largest expected message size because the server uses these buffers to batch client messages which improves performance. Ideally the buffer size is a multiple of the average message size.

Note: When using for HTTP clients, allow between 20 and 250 bytes extra for control information.

Each client multiplexer assigns an output buffer of each size specified by client connectors. So if there were 3 client connectors, each specifying a different output buffer size, and 2 client multiplexers, each multiplexer assigns 3 different buffers (6 in total). When using HTTP Duplex connections (deprecated), each multiplexer assigns an additional buffer for chunked encoding.

When a client multiplexer is unable to write the contents of an output buffer to a client in one go, the writing is deferred and the multiplexer takes a copy of the remaining data in the output buffer into its own temporary buffer.

Message sizing

The sizing of messages that are sent to clients is very important to the overall performance and this must be carefully considered within the design of publishers.

Every topic message has a fixed header of 6 bytes. It then has the topic name terminated by one byte, plus any user header information that is also included with the message.

It is important to work out the size of the message so that the connector buffers can be set correctly, otherwise Diffusion is unable to put the messages on the wire quickly enough.

Byte pinching

With any messaging system, the smaller the messages, the lower the latency and the faster the system performs. There is a consultancy exercise that Push Technology performs as a service to analyze the messages and reduce them as much as possible. The following list includes a few of the best practices to use:

- Only send data that is required by the client.
- Look at the data format and strip any fat off the message.
- Is the information being sent a true delta?
- Client side data models

Message encoding

If you are sending large messages, it is worth compressing the messages. This happens only once on the server, and then the clients have the technology to decompress them, this also includes JavaScript clients. If other encoding is used, it is worth bearing in mind the CPU overhead required.

Client queues

A maximum queue depth can be configured for client queues so that clients are closed if their message backlog becomes too large.

The maximum queue depth must be chosen carefully as a large size might lead to excessive memory usage and vulnerability to Denial of Service attacks, whilst a small size can lead to slow clients being disconnected too frequently.

Client queues do not take any memory, as Diffusion uses a Zero Copy paradigm, but there are consequences in setting them too small or too large. If the client queue is set too small, once the client has filled its queue the Diffusion server closes the client.

When considering queue depth take into account the average message size and publication rate. Messages that are held in the client queue are not garbage collected and can get promoted, which increases their impact on GC pressure. If messages in the client queue build up, consider the maximum delay in the context of your application. For example: Assuming 100 bytes is the average message size and the application is publishing an average of 100 messages per second. If the client queue is setup to have a maximum depth of 1000 messages this means we allow messages to build up for a slow client for up to 10 seconds, during this time a slow client is building up a cache of 100,000 bytes of messages to be sent.

Note: It is natural for queues to build up a little with spikes in publication rate or momentary bandwidth limits, but the tolerance to such delays is expressed in the client queue depth and must be considered in that context.

Client multiplexers

Tuning multiplexers for optimal performance

The load of batching, conflating and merging messages being sent from publishers to outbound clients is spread across client multiplexers. The number of configured client multiplexers must take into account the expected message load and concurrent client connections. The more clients are assigned to a multiplexer the more load it must contend with.

By default, the number of client multiplexers is equal to the number of cores on the host system of the Diffusion server

A client multiplexer processes all client messages into the client queue. Clients are added to the multiplexers according to a round-robin load balancing policy.

Publishers either broadcast on a topic to all subscribed clients or send clients direct messages. When broadcasting all multiplexers are notified and go on to find all subscribed clients which are assigned to the particular multiplexer. When a message is sent to a particular client only that client's multiplexer is notified. It is more efficient to broadcast than it is to send the same message to a large number of clients by iterating over them.

Client multiplexers are non-blocking, high priority threads so having too many can be detrimental, as they are competing for the same resource (CPU). As a rule of thumb, the number of multiplexers must not exceed the number of available logical cores. If a client multiplexer becomes over-subscribed, message latency can increase. For maximum throughput, the number of multiplexers can be set to the number of available cores, but this configuration is only recommended in the case where other threads are assumed to be mostly idle (for example, little inbound traffic, low publisher overhead).

Client multiplexers performance is influenced by the use of merge and conflation policies as those are executed in the multiplexer thread. It is recommended that conflation policy changes and in particular changes to merge conflation logic be profiled and written with performance in mind. In particular the use of locks or any other blocking code is highly discouraged.

Each multiplexer uses a different buffer for each output buffer size that is specified to any connector. If there were three connectors with different output buffer sizes specified, each multiplexer assigns three different buffers. Each multiplexer might also assign an extra buffer for HTTP use. A larger output buffer enables more efficient batching of messages per write, as large writes are generally more efficient but care must be taken to not overwhelm client connections regularly and causing them to be blocked for any period of time.

When a multiplexer is unable to write a message to a client because the buffer has become full, a selector thread is notified. The selector thread is responsible for watching the client and notifying the multiplexer when it becomes writable. The multiplexer remains responsible for writing the message.

Connectors

You can tune your connectors to handle multiple connections and improve performance.

Configuring multiple connectors

When there is more than one publisher application running on a server, configure a separate connector for each publisher so that buffer requirements can be specific to the connector.

It might also be beneficial to configure different connectors for different client types as their requirements can be different. This is especially true for publisher clients where there are low numbers of connections which benefit from very large buffer sizes in both directions.

Buffers

As Diffusion can have tens of thousands of connections at any one time on a machine it is important to make sure that the buffers are set correctly.

To reduce the memory footprint, the Diffusion server will condition the input and output buffers. If the buffers are set too small, Diffusion cannot write the messages in one go and delegate the task to a writer selector.

For more information, see [Buffer sizing](#) on page 1055.

Thread pools

Thread pools are used within Diffusion to optimize the use of threads.

It is important to understand balance when tuning thread usage for a system. There must be sufficient thread resources required but not so many as to starve other parts of the system. At the end of the day there are only so many threads that a system can provide.

In general, when provisioning threads, separate the blocking and non-blocking activities. While it is beneficial to have more threads than cores for blocking tasks it is detrimental to the server if more threads than cores are runnable at any given time.

There are a number of places where thread pools are used within Diffusion. For more information, see [Concurrency](#) on page 1053.

Configurable properties

The following key values can be configured for a thread pool to influence its behavior and use of resource:

Table 69: Values that can be configured for a thread pool

Property	Usage
Core size	<p>The core number of threads to have running in the thread pool.</p> <p>Whenever a thread is required a new one is created until this number is reached, even if there are idle threads already in the pool. After reaching this number of threads then at least this number of threads is maintained within the pool.</p>
Maximum size	<p>The maximum number of threads that can be created in the thread pool before tasks are queued.</p> <p>If this is specified as 0, the pool is unbounded and so the task queue size value is ignored. Generally an unbounded pool is not recommended as it can potentially consume all machine resources.</p>
Queue size	<p>The pool queue size. When the maximum pool size is reached then tasks are queued.</p> <p>If the value is zero, the queue is unbounded. If not zero then the value must be at least 10 (it is automatically adjusted if it is not).</p>
Keep-alive time	<p>The time limit for which threads can remain idle before being terminated.</p> <p>If there are more than the core number of threads currently in the pool, after waiting this amount of time without processing a task, excess threads are terminated.</p>

Property	Usage
	A value of zero (the default) causes excess threads to terminate immediately after executing tasks.
Notification handler	A thread pool can have a notification handler associated with it to handle certain events relating to the pool. This allows for user written actions to be performed (for example, sending an email) when certain pool events (like too much task queuing) occur. See below for more details.
Rejection handler	A thread pool can have a rejection handler associated with it to handle a runnable task that has been rejected. This allows user written actions to handle a runnable task that can not be executed by the thread pool. See below for more details.

Notification handler

A thread pool notification handler can be configured to act upon certain thread pool events.

These events are:

Table 70: Events that a thread pool notification handler can act on

Event	Description
Upper threshold reached	A specified upper threshold for the pool has been reached. This means the pool size has reached the specified size. The event is notified only once and is not notified again until the lower threshold reached event has occurred.
Lower threshold reached	A specified lower threshold for the pool has been reached after an upper threshold reached event has been notified. This means the pool size has now shrunk the specified size.
Task rejected	The pool has rejected a task because there are no idle threads available and the task queue has filled. What happens to the rejected task depends upon the type of pool. Typically, the task is run within the thread that passes the task to the pool, which is not desirable. This is why the thread ought to be notified when it occurs. This differs from the rejection handler in that it does not expose the runnable task. This means it can be used only for notification.

The notification handler is a user written class which must implement the `ThreadPoolNotificationHandler` interface in the threads Java API. The name of such a class can be configured for in-bound or out-bound thread pools or for connector thread pools in which case an instance of the class is created (and must have a no arguments constructor) when the thread pool is created.

Rejection handler

A thread pool can have a rejection handler associated with it to handle a runnable task that has been rejected.

Two rejection handlers are provided with Diffusion. These are the `ThreadService.CallerRunsRejectionPolicy` and `ThreadService.AbortRejectionPolicy`.

The `ThreadService.CallerRunsRejectionPolicy` executes the runnable task in the thread that tried to pass the runnable task to the thread service. This can cause inconsistencies and out of order processing.

The `ThreadService.CallerRunsRejectionPolicy` does not execute the task and instead generates an exception.

Note: By default, the thread that tried to pass the runnable task to the thread service blocks until there is space on the thread pool queue.

The rejection handler is a user written class which must implement the `ThreadPoolRejectionHandler` interface in the threads Java API. The name of such a class can be configured for inbound or outbound thread pools or for connector thread pools in which case an instance of the class is created (and must have a no arguments constructor) when the thread pool is created.

Adjusting the configuration

Adjust thread pools gradually. Ideally, duplicate expected maximum loads in test environment. This environment can be used to tune the thread pools to satisfy the load. Tune the thread pools so they are just able to cope with the maximum load, increasing them beyond this might degrade overall performance.

Background thread pool:

In general, the defaults suffice for the tasks assigned to the background thread pool by Diffusion. If you assign tasks to the pool yourself, consider increasing the number of threads.

Inbound thread pool:

This pool is used to handle inbound connections and messages. Increasing the thread pool allows new connections and received messages to be handled over a greater number of threads. However, much of the behavior in this pool can involve locking the clients or parts of the topic tree. This can cause lock contention that delays processing.

Due to the underlying implementation of Java NIO sockets a high rate of threads being added/removed from the incoming thread pool will result in the allocation of off-heap byte buffers. In extreme cases this can result in an out of memory exception being thrown as the server runs out of off heap allocation space.

Client reconnection

You can configure the client reconnection feature by configuring the connectors at the Diffusion server to keep the client session in a disconnected state for a period before closing the session.

Normally when a client application loses its connection to the server, perhaps due to some communications error, the only option is for it to connect again and then re-establish the state of the topics to which it was subscribed. There is however, the facility for clients to be able to reconnect a lost connection without losing its topic state or messages that were queued for it whilst it was not connected.

This can be useful for mobile clients where connections are less reliable.

Server configuration

To enable clients to reconnect, connectors must be configured to keep client sessions in the DISCONNECTED state for a period during which the client can reconnect. To do this a reconnection timeout must be specified for the connector.

Specify a reconnection timeout, maximum queue depth, and recovery buffer size by using the `<reconnect>` element in the `etc/Connectors.xml` configuration file.

Reconnection timeout (`keep-alive`)

How long a disconnected client's session remains available on the server before being closed. By default, this is 60 seconds.

Maximum queue depth (`max-depth`)

Optional maximum limit on the number of messages to queue for a disconnected client session. By default, this is the same as the queue depth for a connected client session, which is defined by the queue definitions in `Connectors.xml` and `Server.xml`.

Recovery buffer size (`recovery-buffer-size`)

The maximum number of sent messages to keep in a buffer. These messages can then be recovered on reconnection.

```
<connector>
...
<reconnect>
  <keep-alive>60s</keep-alive>
  <max-depth>1000</max-depth>
  <recovery-buffer-size>64</recovery-buffer-size>
</reconnect>
...
</connector>
```

A client can reconnect to the server through this connector within 60 seconds of becoming disconnected. While the client is disconnected, up to 1000 messages are queued for it. These messages are delivered to the client when it reconnects. A buffer of up to 64 sent messages are retained in the recovery buffer. When a client reconnects, the Diffusion server use this buffer to re-send any messages that the client has not received.

If a client signals that it wants to disconnect, the client state on the server is removed when the client disconnects. However, in all other circumstances where the client loses connection, the client goes into the DISCONNECTED state, where the subscriptions are retained and messages are queued as normal for the amount of time specified by the reconnection timeout of the connector.

If the server is configured to expect reconnecting clients, clients that are currently disconnected and might reconnect are excluded from the regular system pings that the server sends to clients.

If the client then reconnects during the period that the session is in DISCONNECTED state, the sending of messages to the client resumes from the point when the failure occurred. Messages that were in transit at the time of disconnection might be lost.

The only way to ensure the delivery of messages on reconnection is for the publisher to mark the messages as requiring acknowledgment as any such messages that have been sent but not acknowledged on reconnection are requeued for the client. The delivery of acknowledged messages is assured. However, because an ack from the client might have been lost during the disconnection, there is the possibility that a message might be delivered to the client twice in this scenario.

It is important when using acknowledged messages to ensure delivery after reconnect that the ack timeout set for messages is sufficiently long to allow for the time that a message can be queued for a client plus the reconnection timeout configured for reconnection.

If an ack timeout expires before a message is even dequeued for a client, the non-acknowledgment is notified and the message is not sent to the client.

Message queue management

Managing message queues when using client reconnection

When a client session is in DISCONNECTED state, messages for the client continue queuing for the client until the reconnection timeout expires or the client reconnects. This puts an unusual load on the client queue and the facility exists to adjust the maximum client queue depth for the period of disconnection.

This is done by specifying a queue depth which is greater than the normal maximum queue depth. When disconnected the queue can expand to the higher value and when reconnection occurs and the queue starts to drain, when the queue size goes down to a value of 80% of its previous limit, the maximum queue depth reverts to the normal value.

The queue depth has an effect only if it represents a value higher than the normal maximum queue depth.

Client reconnection

Configuring the reconnection of clients

Not all clients support reconnection but those that do have a reconnect method which they can call on notification of a lost connection.

If the reconnection succeeds, the client is subscribed to all of the same topics as before and starts to receive messages again, including all of those queued whilst the client was disconnected.

Messages in transit at the time of disconnection might be lost, however any message marked as requiring acknowledgment and sent by the server that was not acknowledged by the client is retransmitted on reconnect. The delivery of acknowledged messages from client to server is assured on reconnect although there is the possibility that the client might receive a message it had acknowledged before the connection again after reconnection if the ack had never reached the server.

A reconnection might not succeed, either because a reconnection timeout is not specified on the connector that the client has connected to, or the specified time period has expired. In this case a normal new connection is established with the same topic set as was specified on the original connection.

How to test reconnection in my environment?

To simulate a communication error, we use a proxy between the client and the server.

- Start Diffusion. (By default it uses port 8080)
- Set the proxy to listen on a different port (for example, 9090) and redirect the connection to 8080

- Connect the client through the proxy on port 9090.
- Kill the proxy.
- Start the proxy.
- Reconnect the client.

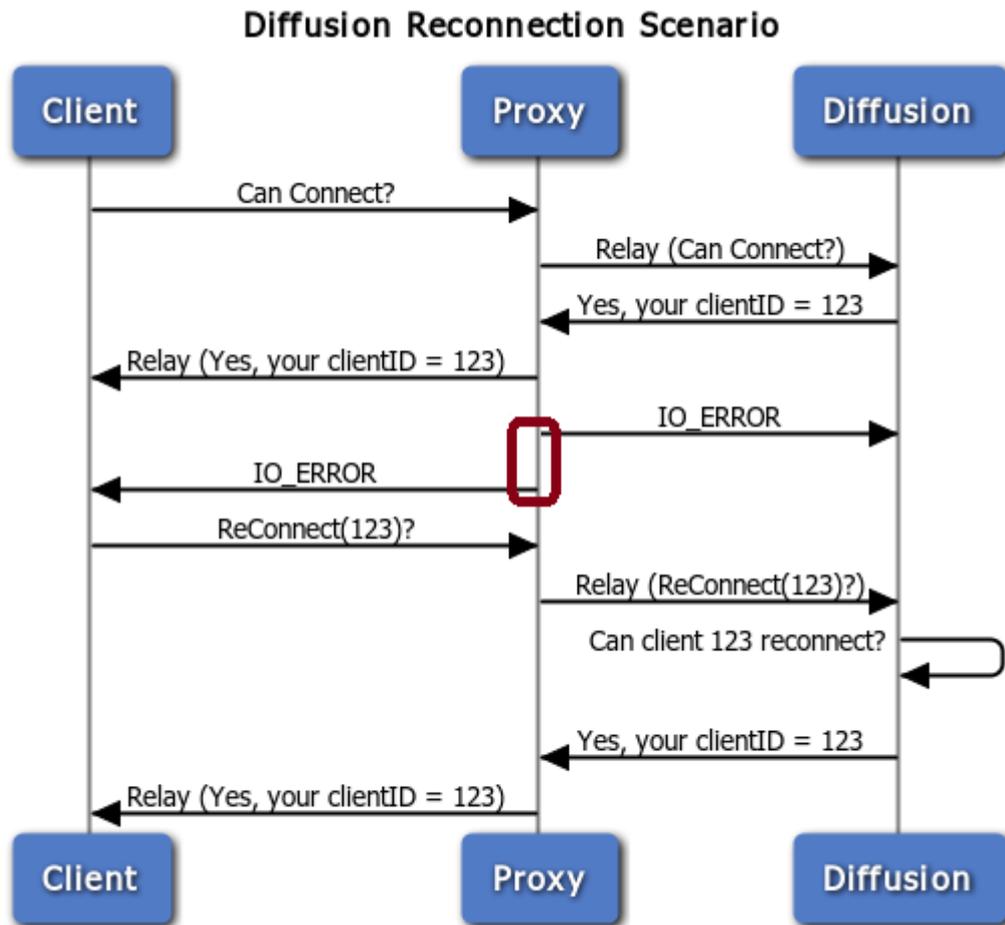


Figure 102: Reconnection scenario

Note: The proxy behaviors are different depending on the operating system and the TCP/IP stack configuration.

Common errors

1. From the client, request a connection close and call reconnect.
After a close request, the client cannot reconnect. In this case, the client will establish a new connection with a different client ID assigned by Diffusion.
2. Unplug the network wire from the computer where the client is running
This will not throw an `IO_ERROR` in the other end of the connection.

Client failover

You can configure a client to fail over to another Diffusion server after it loses connection to the Diffusion server it was previously connected to.

Client failover is when a client loses its connection to a server and attempts to connect to a different one. The client is provided with a list of servers. If a client loses its connection to a server it can automatically attempt to connect to the next server in the list. If it fails to connect or loses its connection to that server, it tries the next server on the list. This is referred to as autofailover. Generally the list of servers to connect to must be provided before attempting to make the connection. How the list of servers is provided differs between client APIs and the JavaScript client does not support autofailover but it can be implemented using the callback methods.

Using automatic failover

If a client has an established connection that it loses, autofailover attempts to open a new connection in the next connection in the list. This is not compatible with reconnection because reconnection attempts to preserve the state of the client (the client ID and the subscribed topics). As the new server has no knowledge of the client it is unable to preserve this state. Autofailover must be enabled and a list of servers to connect to provided.

Using load balancing with autofailover

You can enable load balancing in conjunction with autofailover. When load balancing is enabled and a client loses connection, the list of servers is shuffled before the client selects the next server to attempt to connect to.

In Java, for example, you can enable load balancing by using the `setLoadBalancing` method on the `ConnectionDetails` object.

Using server cascading

When a client attempts to place a connection, if the attempt fails, the next server in the list is chosen. Server cascading is similar to autofailover except this logic is applied prior to a connection, whereas autofailover applies once a connection is in place.

In Java, for example, you can enable server cascading by using the `setCascading` method on the `ConnectionDetails` object.

Note: Server cascading is different to protocol cascading, which attempts to connect to the same server using different protocols before a connection has been opened.

Java failover

Configuring failover in Java

In Java the `ConnectionDetails` object supports a collection of `ServerDetails` objects. The server details are used to control failover between servers. The `ConnectionDetails` factory methods provide several options for creating `ConnectionDetails` with multiple `ServerDetails` objects. A collection of `ServerDetails` object can be passed as a parameter, a `varargs` method supports `ServerDetails` and another `varargs` method supports `String` URLs, which `ServerDetails` objects are constructed from. After construction the `ServerDetails` objects used by the `ConnectionDetails` can be altered by calling the `setServerDetails(Collection<ServerDetails>)` method.

The following code supports autofailover from the server with the IP address 192.168.0.1 to the server 192.168.0.2. If the client loses connection to 192.168.0.1 it tries to connect to 192.168.0.2. It uses the `varargs` method to create a `ConnectionDetails` object with multiple `ServerDetails` objects constructed from String URLs.

```
ConnectionDetails details =
    ConnectionFactory.createConnectionDetails(
        "dpt://192.168.0.1:8080",
        "dpt://192.168.0.2:8080");
details.setAutoFailover(true);
ExternalClientConnection client = new
    ExternalClientConnection(listener, details);
client.connect();
```

For further information refer to the Java API documentation for `ConnectionDetails` and `ServerDetails`.

JavaScript failover

Using failover in JavaScript

The JavaScript client does not support autofailover. Support for failover is limited. If the connection attempt fails `DiffusionClient.connect(DiffusionClientConnectionDetails)` can be called with a different object. You must provide the logic to do this on connection failure.

ActionScript failover

Using failover in ActionScript

In ActionScript the `ConnectionDetails` object supports an array of `ServerDetails` objects. The server details are used to control failover between servers. The `ConnectionDetails` constructor has a mandatory `ServerDetails` object. After construction additional `ServerDetails` objects used by the `ConnectionDetails` can be altered by calling the `addServerDetails(ServerDetails)` method and the `setServerDetailsArray(Array)` method.

This code:

```
var server0:ServerDetails = new
    ServerDetails("dpt://192.168.0.1:8080");
var server1:ServerDetails = new
    ServerDetails("dpt://192.168.0.2:8080");
var details:ConnectionDetails = new ConnectionDetails(server0);
details.addServerDetails(server1);
details.setAutoFailover(true);
var client:DiffusionClient = new DiffusionClient();
client.connect(details);
```

supports autofailover from the server with the IP address 192.168.0.1 to the server 192.168.0.2. If the client loses connection to 192.168.0.1, it tries to connect to 192.168.0.2. It uses the `addServerDetails(ServerDetails)` method to add a single additional server to connect to. For further information refer to the [Flex Classic API documentation](#) for `ConnectionDetails` and `ServerDetails`.

Client throttling

Throttling is a method of ensuring that the Diffusion server limits (throttle) the volume of messages it transmits to a client within a specified period of time. This can be used to limit bandwidth usage or to prevent more messages being sent to a client than it can cope with.

How does throttling work?

Throttling is applied to a client queue so that the number or volume of messages sent to that client is restricted. Diffusion only dequeues a message to send to a client if that client has not breached its throttling limit.

Throttle types;

- Messages per second (Only a specified number of messages are sent every second.)
- Bytes per second (Only a specified number of bytes are sent every second)
- Message interval (A single message is sent every n milliseconds.)
- Buffer interval (A full output buffer (or the equivalent of) is sent every n milliseconds)

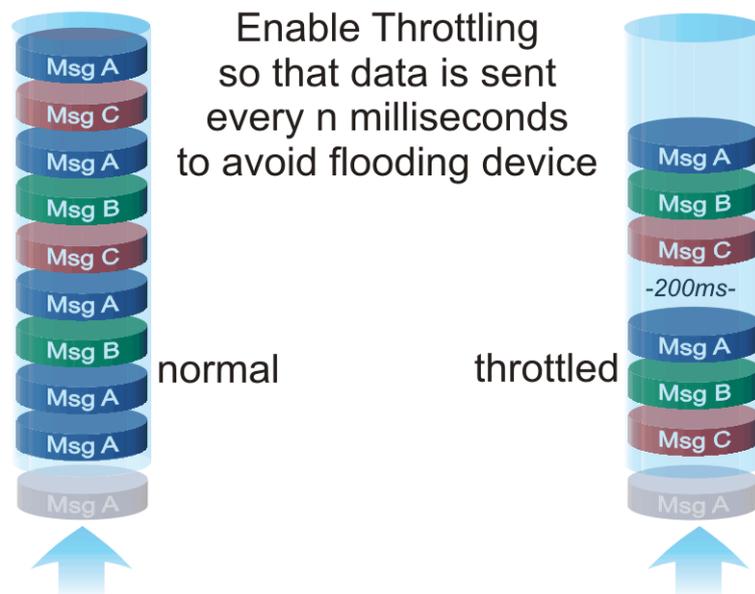


Figure 103: Normal and throttled client queues

Enabling throttling

Throttling can be enabled on a client-by-client basis from within the publisher.

To throttle a `Client`.`throttle` method which allows you to specify the type of throttling and a limit. A `clientThrottler` reference is returned.

If the throttle method is called for a client that is already throttled, it has the effect of removing the old throttler and adding a new one.

Call the `Client.removeThrottler` method to stop throttling.

The `Client.isThrottling` method can be used to determine whether a client is currently throttled and, if it is, the `getThrottler` method can be used to determine the type of throttling and the current limit.

Java memory usage

Typically you do not have to tune the Java VM's use of native memory. However, in certain conditions, consider using runtime options to change the default behavior.

If you do not do SSL offloading at the Diffusion server

If your clients make secure connections which are SSL offloaded at a load balancer instead of at the Diffusion server or your connections to the Diffusion server or if all your connections are insecure, consider tuning the following runtime options:

-Xmx

Sets the maximum heap size.

Consider setting the maximum heap size to around 80% of the RAM available on your system.

If you use SSL offloading at the Diffusion server

If your clients make secure connections to the Diffusion server and these connections are SSL offloaded at the Diffusion server, ensure that you tune the following runtime options:

-Xmx

Sets the maximum heap size.

-XX:MaxDirectMemorySize

Sets the maximum total size (in bytes) of direct-buffer allocations. By default, the JVM chooses the size for direct-buffer allocations automatically.

Diffusion uses direct memory to offload SSL connections.

Ensure that the combined total of these two values does not exceed 80% of the RAM available on your system.

Platform-specific issues

To run Diffusion it might be necessary to increase the number of sockets and reduce timewait.

It might also be necessary to increase the number of open files that is allowed on UNIX or Linux systems

Socket issues

To fix these problems, complete the following steps based on platform.

Windows

Setting values on Windows

Setting TCP timed wait

This parameter determines the length of time that a connection stays in the TIME_WAIT state when it is closed. When a connection is in the TIME_WAIT state, the socket pair cannot be reused. This is also known as the 2MSL state because the value is twice the maximum segment lifetime on the network. See RFC 793 for further details.

Add TcpTimedWaitDelay registry values as a workaround. You can set these values through REGEDIT command.

Set TcpTimedWaitDelay to 30:

1. Select **Start > Run**.
2. In the available field, enter `regedit`.
3. Go to the key directory file: `HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/Tcpip/Parameters/TcpTimedWaitDelay`. The value type is `REG_DWORD`.
4. Double-click **TcpTimedWaitDelay**.
5. Select **Decimal**.
6. Type 30 in the **Value** data field. The default value for this field is 0xF0 (240 decimal). The valid range is 30-300 (decimal).

Setting MaxUserPort

This parameter controls the maximum port number used when an application requests any available user port from the system. Normally, short-lived ports are allocated in the range from 1024 through 5000. Setting this parameter to a value outside of the valid range causes the nearest valid value to be used (5000 or 65534).

Add MaxUserPort registry values as a workaround. You can set these values through REGEDIT command.

Set the MaxUserPort to 65534

1. Select **Start > Run**.
2. In the available field, enter `regedit`.
3. Go to the key directory file: `HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/Tcpip/Parameters/MaxUserPort`. The value type is `REG_DWORD`.
4. Double-click **MaxUserPort**.
5. Select **Decimal**.
6. Type 65534 in the **Value** data field. The default value for this field is 0x1388 (5000 decimal). The valid range is 5000 – 65534(decimal).

Linux

Configuring sockets values on Linux

Decrease the time wait before closing the sockets by entering:

```
# echo 3 > /proc/sys/net/ipv4/tcp_fin_timeout
```

Sometimes systems are now configured to prevent one from using a large number of ports, check the port range and modify if required.

```
# cat /proc/sys/net/ipv4/ip_local_port_range
```

This can be increased by issuing the following command

```
# echo "1025 65535" > /proc/sys/net/ipv4/ip_local_port_range
```

To have these new values take effect you might have to do (as root)

```
# /etc/rc.d/init.d/network restart
```

If you want these new values to survive across reboots you can add them to `/etc/sysctl.conf`.

```
# Allowed local port range
net.ipv4.ip_local_port_range = 1025 65535
# net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_fin_timeout = 3
```

UNIX

Increasing the number of files a process on a UNIX system can open also increases the number of sockets that process can open. The operating system uses file descriptors to handle filesystem files as well as pseudo files, such as connections and sockets.

You need the following number of sockets for each client connection:

- DPT, WS — one socket per connection
- HTTP Full Duplex (deprecated), HTTP Polling, HTTP Chunked Streaming — two sockets per connection

Use the `ulimit` command to increase the number of open files. You can do this in one of the following ways:

- As a global setting.
This can be set by your network administrator.
- In the start script for Diffusion.

Edit the `diffusion_installation/bin/diffusion.sh` file to add the following line at the start:

```
ulimit -n open_files
```

Where the value of `open_files` is any suitable integer value, for example 8192.

You can use the `java.lang:type=OperatingSystem` MBean to inspect the number of files on your UNIX operating system. See the following properties:

MaxFileDescriptorCount

The total number of files that a process on a UNIX system can open. This is the number that you can set with `ulimit -n`.

OpenFileDescriptorCount

The number of files that are currently open.

The difference between these values is the number of files you have available to use for sockets.

Publisher design

Considerations when designing a publisher

Consider the following points when designing and writing a publisher:

Data modeling

The way that the data is fed to a publisher and the way in which the state of the data is maintained within a publisher is key to good performance. Keep message sizes to a minimum and this can be achieved using fine data granularity enabled by the topic tree.

Caching

Cache messages wherever possible rather than building new ones every time one must be sent. This particularly applies to topic load messages which can be cached to send to every new client that subscribes, and rebuilt only when the data actually changes. The ideal place to keep such cached messages is with a data object attached to the topic (see topic data pattern).

String handling

Building of Strings by concatenation is very inefficient in Java. Keep String concatenation to a minimum. When String content is used, message caching can help to some degree and wherever possible cache Strings that must be built.

Conditional processing

Excessive use of conditional processing (Checking of topic names, and so on) can be expensive. Use of the topic data pattern can significantly reduce the need for such processing when many topics are in use.

Topic naming

As every message must carry the topic name, long topic names can lead to a large amount of data traffic which can be disproportional to the data being carried. This can be a particular problem when using hierarchical topics. A solution is to use the topic aliasing feature so that only short aliases of the topic names are transmitted in messages.

Concurrency

Concurrent programming means that access to data often must be synchronized but care must be taken not to synchronize more than is necessary as performance can be significantly affected.

Demos

Diffusion comes with demo applications that demonstrate certain features of Diffusion.

If you have chosen to include demos as part of the installation process, the demos are included as DAR files in the `demos` directory of your Diffusion installation and as source in the `demos/src` directory of your Diffusion installation. You can build the demos from source using *mvndar*.

Demos

The demos in the **Demos** section of the default Diffusion installation web page demonstrate various applications of Diffusion. They also contain a connection widget, allowing you to modify the transports used as well as inspect messages incoming and outgoing from the client.

To access the Diffusion web page and demos, start the web server and type in the following URL into a browser: <http://localhost:8080>.

Table 71: Demos provided with the Diffusion server

Tech Magnets demo	This demo provides an example of shared state handled by Diffusion. Tiles are created on the server in response to client actions. When a tile is moved by the client, a message is sent to the Diffusion server, the state is modified and an exclusive message is sent to all other subscribers. Open this demo in multiple browsers to see the realtime DOM updates.
Drawing Board demo	Multiple users draw on a virtual blackboard with colored chalk. All clients are updated in real time with strokes from the other clients. Open this demo in multiple browsers to see the responsiveness of client interactions. Requires Canvas support in the browser.

Dogfight (game) demo	Aerial combat game for up to 8 players. Use the arrow keys to control your plane and the space key to fire. Uses HTML 5 features.
Asset Trader	View and trade on our cross-platform FX trading demo where price feeds, instruments, news, and user interactions are all distributed in real time.

Building the demos using mvndar

You can use the Maven plugin *mvndar* with the provided `pom.xml` file to build the demos.

The `pom.xml` file included in the `demos/src` directory of your Diffusion already contains the required reference to *mvndar*.

To build the demos using *mvndar*, complete the following steps:

1. From the command line, go to the `diffusion_installation/demos/src` directory.
2. Set the `DIFFUSION_HOME` environment variable to the absolute path of your Diffusion installation.
 - On Linux or OS X/macOS, type `export DIFFUSION_HOME=/diffusion_installation`
 - On Windows, type `set DIFFUSION_HOME=C:/diffusion_installation`
3. Run the `mvn clean install` command.

Related concepts

[Using Maven to build Java Diffusion applications](#) on page 513

Apache™ Maven is a popular Java build tool and is well supported by Java IDEs. You can use Apache Maven to build your Diffusion applications.

[Build server application code with Maven](#) on page 518

The Diffusion API for server application code is not available in the Push Technology public Maven repository. To build server components, you must install the product locally and depend on `diffusion.jar` using a Maven system scope.

Tools

If the tools were installed during the installation process, there are some tools that can help with the monitoring of Diffusion plus some handy utilities.

There are additional tools and utilities that are available in public repositories, such as GitHub and Maven.

Tools for Amazon Elastic Compute Cloud (EC2)

This is a description of a number of tools and adaptations which are provided for use when using Diffusion with Amazon EC2™.

Diffusion includes in `tools/ec2/` a number of files useful when deploying Diffusion to an Ubuntu® image on an EC2 virtual machine.

diffusion.conf

Ubuntu makes use of the [Upstart daemon](#) as a replacement for `init`. Copied to `/etc/init/diffusion.conf` this file contains configuration for Upstart to begin Diffusion at boot-time in a background process as a unprivileged user. It establishes [iptables](#) rules to route traffic from privileged ports to Diffusion.

Users can stop and start Diffusion using Upstart commands, for example, to start the server

```
service diffusion start
```

To stop Diffusion

```
service diffusion stop
```

To check the status of Diffusion

```
service diffusion status
```

In the event that something goes amiss Upstart writes a log file to `/var/log/upstart/diffusion.log`

etc/Connectors.xml

Except for two port number changes this is an otherwise regular copy of `etc/Connectors.xml` normally found in a Diffusion installation. This edition however binds the Flash policy connector and the Silverlight policy connector to port numbers greater than 1024, making it possible to run Diffusion as an unprivileged process. Use this file in conjunction with the `iptables` rules established in `diffusion.conf`

ec.xml

An illustrative Apache Ant script that can be used to start, stop, and get status from a Diffusion server running on an Amazon EC2 Linux host. It also demonstrates an inelegant means of deploying and undeploying a DAR file to/from the remote server, by copying the file to the remote server, then moving it into the deploy directory.

Table 72: Targets

Property	Purpose
start	Runs <code>sudo service diffusion start</code> on remote host using SSH
stop	Runs <code>sudo service diffusion stop</code> on remote host using SSH
status	Runs <code>sudo service diffusion status</code> on remote host using SSH
deploy	Uploads local DAR file to staging dir, then moves it into Diffusion deploy directory
undeploy	Removes DAR file from deploy directory, signaling Diffusion to undeploy related publishers (where possible)

The script is driven by named properties:

Table 73: Properties for targets start, stop and status

Property	Purpose
remote.host	EC2 host running sshd
remote.username	Authentication username, default of <code>ubuntu</code>
remote.keyfile	PEM encoded key use during authentication

Table 74: Additional properties for targets deploy and undeploy

Property	Purpose
dar.file	Name of a DAR file to deploy or undeploy
remote.diffusion.dir	Root directory of the remote Diffusion installation

Example deployment:

```
ant -Dremote.host=54.235.65.36 \  
-Dremote.keyfile=$HOME/.ssh/ec2-push1.pem \  
-Ddar.file=$HOME/Applications/Diffusion5.9.24/demos/dogfight.dar \  
-Dremote.diffusion.dir=/home/ubuntu/Diffusion/Diffusion5.9.24 \  
deploy
```

The script uses no proprietary Diffusion code and is open to extension during development of a solution. Both the `sshexec` and `scp` tasks depend on the [jsch library](#) which might have to be downloaded.

Tools for Joyent

This is a description of adaptations which are provided for use when using Diffusion within the Joyent Cloud.

Diffusion includes in `tools/joyent/` files useful when deploying Diffusion to an SmartOS virtual machine in the [Joyent cloud](#).

diffusion.xml

SmartOS makes use of [SMF](#) to start services at boot-time. Once tailored for the host and installed into SMF this file contains configuration for SMF to begin Diffusion at boot-time in a background process as an unprivileged user called `push`. This file does not create or assert the existence of a `push` user.

Users can stop and start Diffusion using SMF commands. For example, to start the server

```
sudo svcadm enable diffusion
```

To stop the server

```
sudo svcadm disable diffusion
```

Part VI

Upgrading Guide

If you are planning to move from an earlier version of Diffusion to version 5.9, review the following information about changes between versions.

We recommend that you upgrade to the latest version of Diffusion as soon as you can.

When upgrading across multiple versions, ensure that you review the release notes and upgrade steps for all intermediate versions. For example, if you are upgrading from version 4.x to version 5.1, first follow the upgrade steps from version 4.x to 5.0, then follow the steps to upgrade from version 5.0 to 5.1.

Release notes are available at the following location: <http://download.pushtechnology.com>

For more information about [Diffusion versions](#) and [support and upgrade policy](#), see the [Support Center](#).

Related concepts

[What's new in Diffusion 5.9?](#) on page 25

The latest version of Diffusion contains new functions, performance enhancements and bug fixes.

In this section:

- [Interoperability](#)
- [Upgrading from version 4.x to version 5.1](#)
- [Upgrading from version 5.1 to version 5.5](#)
- [Upgrading from version 5.5 to version 5.6](#)
- [Upgrading from version 5.6 to version 5.7](#)
- [Upgrading from version 5.7 to version 5.8](#)
- [Upgrading from version 5.8 to version 5.9](#)
- [Upgrading to a new patch release](#)
- [Known issues in Diffusion 5.9](#)

Interoperability

If you plan to use different versions of Diffusion servers and clients together, review the following information that summarizes support between versions.

Support for new topic types

Diffusion version 5.7 introduced the following new topic types: JSON and binary. These topic types are not supported by previous versions of Diffusion or by Classic API clients in any version.

If you use JSON or binary topics, ensure that you use 5.7 or later and that you use the Unified API. Attempting to use the Classic API with JSON or binary topics is not supported and results in undefined behavior.

Interoperation between clients and servers

The following table describes which Unified API client versions interoperate with which server versions.

Note: Interoperability is only tested and guaranteed between the Diffusion server and clients of the same version and the immediately previous version. For example, 5.9 clients and 5.8 clients are certified to work with a 5.9 Diffusion server. All older clients are supported on a best-effort basis, unless explicitly marked as unsupported in the following table.

Table 75: Unified API interoperation

Client version	Server version				
	5.1	5.6	5.7	5.8	5.9
5.1 Unified API	✓	✗	✗	✗	✗
5.6 Unified API	✗	✓	✓	✓	✓
5.7 Unified API	✗	✗	✓	✓	✓
5.8 Unified API	✗	✗	✗	✓	✓
5.9 Unified API	✗	✗	✗	✗	✓

The following table describes which Classic API client versions interoperate with which server versions:

Table 76: Classic API (deprecated) interoperation

Client version	Server version				
	5.1	5.6	5.7	5.8	5.9
5.1 Classic API	✓	✓	✓	✓	✓
5.6 Classic API	✓	✓	✓	✓	✓

Client version	Server version				
	5.1	5.6	5.7	5.8	5.9
5.7 Classic API	✓	✓	✓	✓	✓
5.8 Classic API	✓	✓	✓	✓	✓
5.9 Classic API	✓	✓	✓	✓	✓

Interoperation between servers

Replication

All Diffusion servers within a cluster must be of the same level.

Server versions	5.1	5.6	5.7	5.8	5.9
5.1	✓	✗	✗	✗	✗
5.6	✗	✓	✗	✗	✗
5.7	✗	✗	✓	✗	✗
5.8	✗	✗	✗	✓	✗
5.9	✗	✗	✗	✗	✓ See note

Note: If some of the Diffusion servers in your cluster are version 5.9.4 and earlier and others are 5.9.5 and later, this change can cause inconsistent behaviors when replicating branches of the topic tree that contain slave topics. To ensure consistent behavior when replicating slave topics, update all of your Diffusion servers to 5.9.5 and later.

Fan out

All servers later than 5.6 interoperate.

To receive propagated missing topic notifications through fan-out connections, all servers must be version 5.9 or later.

Server versions	5.1	5.6	5.7	5.8	5.9
5.1	✗	✗	✗	✗	✗
5.6	✗	✓	✓	✓	✓
5.7	✗	✓	✓	✓	✓

Server versions	5.1	5.6	5.7	5.8	5.9
5.8	✗	✓	✓	✓	✓
5.9	✗	✓	✓	✓	✓

Publishers

DEPRECATED: Publishers deployed to Diffusion servers can connect to and communicate with publishers deployed to Diffusion servers of different versions.

Server versions	5.1	5.6	5.7	5.8	5.9
5.1	✓	✓	✓	✓	✓
5.6	✓	✓	✓	✓	✓
5.7	✓	✓	✓	✓	✓
5.8	✓	✓	✓	✓	✓
5.9	✓	✓	✓	✓	✓

Related concepts

[Upgrading from version 4.x to version 5.1](#) on page 1079

Consider the following information when upgrading from Diffusion version 4.x to version 5.1.

[Upgrading from version 5.1 to version 5.5](#) on page 1085

Consider the following information when upgrading from Diffusion version 5.1 to version 5.5.

[Upgrading from version 5.5 to version 5.6](#) on page 1091

Consider the following information when upgrading from Diffusion version 5.5 to version 5.6.

[Upgrading from version 5.6 to version 5.7](#) on page 1095

Consider the following information when upgrading from Diffusion version 5.6 to version 5.7.

[Upgrading from version 5.7 to version 5.8](#) on page 1098

Consider the following information when upgrading from Diffusion version 5.7 to version 5.8.

[Upgrading from version 5.8 to version 5.9](#) on page 1102

Consider the following information when upgrading from Diffusion version 5.8 to version 5.9.

[Upgrading to a new patch release](#) on page 1105

When upgrading to a new patch release there are typically no changes to the configuration values or the APIs. All that is required is to copy your existing files from the old installation to the new installation.

[Known issues in Diffusion 5.9](#) on page 1106

Be aware of the following issues when using Diffusion 5.9.

Upgrading from version 4.x to version 5.1

Consider the following information when upgrading from Diffusion version 4.x to version 5.1.

Upgrading your applications

Server-side components

Recompile all Java application components that are deployed to the Diffusion server, such as publishers and authorization handlers, against the new version `diffusion.jar` file. This file is located in the `lib` directory of your new Diffusion server installation.

Some features that your Java application components might use have been removed or deprecated. Review the API changes information in the following section to see if these changes affect your applications.

Remote control

The remote control APIs are no longer supported. Reimplement your remote control as a control client using the Unified API control features. For more information, see [and](#) .

Event publishers

The event publisher APIs are deprecated. Reimplement your event publisher as a control client using the Unified API control features. For more information, see [and](#) .

Clients

You can choose not to recompile your client applications and continue to use client libraries from a previous release. If you choose to use client libraries from a previous release, ensure that the libraries are compatible with the new server. For more information, see [Interoperability](#) on page 1076.

You can choose to upgrade your client applications to use the new client libraries. To do this, recompile the client applications against the client libraries located in the `clients` directory of your new Diffusion server installation. Some features that your client applications might use have been removed or deprecated. Review the API changes information in the following section to see if these changes affect your applications.

API changes

Further information about removed or deprecated features is available in the release notes provided in the `docs` directory of your Diffusion installation.

The following table lists API classes and methods that have been removed. If you attempt to recompile application code that uses these classes or methods against the version 5.1 APIs, it fails. Rewrite your application code to not include these features.

Table 77: API features removed in version 5.0 and 5.1

API affected	Removed feature	Suggested alternative
Java API .NET API	Remote control	Reimplement your remote control applications as control clients using the Unified API. For more information, see and .
Java API	Methods in the <code>APIProperties</code> class: <ul style="list-style-type: none"> <code>setInboundThreadPoolSize</code> <code>getInboundThreadPoolSize</code> 	Use the <code>ThreadsConfig</code> class instead. For more information, see Java Unified API documentation .
Android API	Methods in the <code>DiffusionClient</code> class: <ul style="list-style-type: none"> <code>getCredentials</code> <code>setCredentials</code> 	Use the methods in <code>ServerDetails</code> or <code>ConnectionDetails</code> instead. For more information, see Android Unified API documentation .
Java API	<code>MessageComparator</code> interface and <code>compareTo</code> and <code>equals</code> methods on all <code>Message</code> classes.	Use conflation policies instead. For more information, see Java Unified API documentation .
Java API	<code>TopicDetails</code> class	Use the <code>TopicDefinition</code> class instead. For more information, see Java Unified API documentation .
Java API	Methods in the <code>ThreadsConfig</code> class: <ul style="list-style-type: none"> <code>setWriterSelectors</code> <code>getWriterSelectors</code> 	No longer used and no alternative required.
Java API	<code>Management</code> , <code>Proxy</code> , and <code>ServerProxy</code> interfaces	No longer used and no alternative required.
Java API	<code>Publisher.consoleLogLevelChange</code>	No longer used and no alternative required.
Java API	<code>ThreadServer.getOutboundThreadPool</code>	
Publisher API Event Publisher API	The capability to set the maximum queue size to -1, which specified an unbounded queue size, using <code>Client.setMaximumQueueSize()</code> .	Set the maximum queue size value to a positive integer.
Java Classic API	<code>TopicFetchHandler</code>	Use the Publisher API or Unified API.

The following table lists API classes and methods that have been deprecated. If your application code uses these classes or methods, consider rewriting your application code to not include these features.

Table 78: API features deprecated in version 5.0 and 5.1

API affected	Deprecated feature	Suggested alternative
Java API	Using authorization handlers for authentication and the <code>AuthorisationHandler.canConnect</code> method.	Use authentication handlers instead. For more information, see User-written authentication handlers on page 143.
Java API	<code>APIProperties</code> class	Use methods in the <code>Utils</code> or <code>RootConfig</code> classes instead. For more information, see Java Unified API documentation .
Java API .NET API	Event publishers	Reimplement your event publishers as control clients using the Unified API. For more information, see and .
Java	<code>Client.getNumberOfMessagesSent</code> and <code>Client.getNumberOfMessagesReceived</code>	Use <code>Client.getStatistics</code> instead. For more information, see Java Unified API documentation .
Java	Methods that navigate up from a configuration item to its parent configuration item.	Instead navigate down from the root configuration item.
Java	<code>MNode.getMessage</code>	No longer used and no alternative required.
Unified API	<code>TopicUpdateControl.TopicSource</code> <code>TopicUpdateControl.TopicSource.Default</code> <code>TopicUpdateControl.TopicSource.Topic</code>	<code>TopicUpdateControl.UpdateSource</code> <code>DefaultUpdateControl.UpdateSource.Default</code> <code>TopicUpdateControl.Updater</code>
Java Unified API	<code>Messaging.Listener</code> and associated methods	<code>Messaging.MessageStream</code>
Java Unified API	<code>Topics.Listener</code> and associated methods	<code>Topics.TopicStream</code>
Java Unified API	<code>Updater.update()</code> methods that take both <code>Content</code> and <code>UpdateOptions</code> as parameters	<code>Updater.update()</code> methods that take <code>Update</code> as a parameter
Java Unified API	<code>comparator()</code> and <code>duplicatesPolicy()</code> methods in the <code>PagedTopicDetails.Builder</code> class	<code>order(String)</code> , <code>order(Duplicates, String)</code> , or <code>unordered()</code>
Java Unified API	<code>getComparator()</code> and <code>getDuplicatesPolicy()</code> methods in the <code>PagedTopicDetails.Attributes</code> class	<code>getOrderingPolicy()</code>

API affected	Deprecated feature	Suggested alternative
.NET Unified API	All <code>SetProperty()</code> methods in the <code>ISessionFactory</code> class, where <code>Property</code> is the name of the value you want to change	<code>Property()</code>
Unified API	The <code>autoSubscribe</code> method in the <code>TopicDetails.Builder</code> interface. In future, <code>auto-subscribe</code> is always true.	None
JavaScript Classic API	The functions <code>setCrypted()</code> and <code>getCrypted()</code>	<code>setEncrypted()</code> and <code>isEncrypted()</code>
Java Server API	The functions <code>getStartTimeMillis()</code> , <code>getUptime()</code> , and <code>getUptimeMillis()</code> on the <code>c.p.d.api.topic.Subscription</code> class	You can use a publisher to get equivalent functionality.
Java Unified API	All static fields in the <code>c.p.d.client.types.Constants</code> class	These fields are now available in the <code>c.p.d.client.content.Record</code> class.
Java Unified API, .NET Unified API	<code>RecordContentReader.hasMore()</code>	<code>RecordContentReader.hasMoreRecords()</code> , <code>RecordContentReader.hasMoreFields()</code>

The following list includes behavior that has changed in the API. If your application code relies on the previous behavior, rewrite your application code to take into account the new behavior.

- The Publisher API methods that add topics no longer block until automatic pre-emptive subscriptions have been processed. Matching pre-emptive subscriptions are be completed in the background.
- The Java API now enables you to set auto-subscribe using a `TopicDefinition`
- The format of the generated client IDs has changed
- `getStatistics` no longer returns null if statistics recording is disabled. Instead it returns a value of -1.
- Clients that subscribe to topics that they are already subscribed to, no longer receive an initial topic load.
- An anonymous user (a user with an empty string as username) cannot authenticate with a password. If a client provides a password but no username or an empty username, the authorization handler cannot retrieve the password from the `getCredentials()` method.
- The UpdateSource API that replaces the TopicSource API in the Unified API behaves in a very similar way, but has some differences.
 - UpdateSource includes an `onRegister` callback, which provides a `RegisteredHandler` that the client can use to deregister as an update source. Previously, `TopicSource` only provided the ability to deregister as an update source to the client that was the active update source.
 - UpdateSource includes an `onError` callback, which indicates to a client when an update source is prematurely closed.
 - Updaters do not accept an `UpdateOptions` object as a parameter. Instead an `Update` object is used to contain the update content and any additional information about the update.

For more information about how update sources work, see [Updating topics](#) on page 331.

- In the Unified API, the logging level at which the `Topics.Listener.Default` logs updates has changed from “warn” to “debug”.
- In the Unified API, topic and messaging listeners are called in the order that they were registered. In previous releases, these listeners might have been called in any order.
- In the Unified API, you must set a fallback topic or messaging listener explicitly. In previous releases, a fallback topic or messaging listener was set by default.
- In the Unified API, a notification occurs for any type of selector that does not match with any topics. In previous releases, a missing topic notification occurred only if a topic path selector was used for subscribe or fetch and there was no such topic.
- In the Unified API, you can add multiple session listeners and remove session listeners. In previous releases, you could add only one session listener.

Upgrading your server installation

To upgrade your Diffusion server installation, complete the following steps:

1. Use the graphical or headless installer to install the new version of Diffusion.

For more information, see [Installing the Diffusion server](#) on page 535.

2. You can copy your existing license file from your previous installation to the `etc` directory of your new installation.
3. You can copy your existing configuration files from the `etc` directory of your previous installation to the `etc` directory of your new installation. When you do, consider making the following changes:

- In the `WebServer.xml` configuration file for your production installation, remove or comment out the configuration for the HTTP deploy service.

Access to this service is not restricted. If you enable the deploy service, you must restrict access to the deploy URL by other methods to prevent unauthorized or malicious access. For example, by setting up restrictions in your firewall.

- Remove the `writer-selector` configuration from the `Server.xml` configuration file. Writer selectors are no longer used.

Warning: Do not confuse writer selectors with write selectors.

- If you now use authentication handlers for authentication, configure these handlers in the `Server.xml` configuration file.
- If you now use the replication high availability features, configure these in the `Replication.xml` configuration file.
- If you use the Whols service, but do not explicitly configure it, you must now configure the service in the `Server.xml` configuration file.

In previous releases, if no configuration was specified for the Whols service, the service started with the default configuration. In this release, the service does not start unless configuration is present in the `Server.xml` configuration file.

- In the `Server.xml` and `Connectors.xml` configuration files, if you have set maximum client queue depths to be unbounded (0), change these values. Unbounded outbound client queues are no longer allowed.

The validation of the configuration files has been relaxed. The order of the element within the files is less strict.

4. If you start the Diffusion server from your own scripts or Java programs, you must update them to take into account the following changes:

- The Java license agent has been removed. Remove the following argument from the Java command you use to start the server:

```
-javaagent:../lib/licenceagent.jar=../etc/licence.lic,../etc/publicKeys.store
```

- New system properties are required by the Diffusion server.

Include the following properties in the Java command that starts the server:

```
-Ddiffusion.license.file=diffusion_installation/etc/licence.lic  
-Ddiffusion.keystore.file=diffusion_installation/etc/publicKeys.store  
-Ddiffusion.home=diffusion_installation/lib
```

You can also supply these properties as VM arguments.

For more information, see [Running from within a Java application](#) on page 637

If you use the start scripts provided with the Diffusion installation, you do not need to make any changes.

Behavior changes at the Diffusion server

The following list includes behavior that has changed at the server. If your solution relies on the previous behavior, adjust your solution to take into account the new behavior.

- In previous releases, messages that required acknowledgment were prioritized over other messages. This might have caused ordering problems. From 5.1, messages that require acknowledgment are queued for sending in the order of receipt. You might have to increase your acknowledgment timeout value to allow for the additional queuing time.
- The non-configurable 5s timeout between HTTP polls has been removed. Use `<system-ping-frequency>` for HTTP connectors.

Related concepts

[Upgrading from version 5.1 to version 5.5](#) on page 1085

Consider the following information when upgrading from Diffusion version 5.1 to version 5.5.

[Upgrading from version 5.5 to version 5.6](#) on page 1091

Consider the following information when upgrading from Diffusion version 5.5 to version 5.6.

[Upgrading from version 5.6 to version 5.7](#) on page 1095

Consider the following information when upgrading from Diffusion version 5.6 to version 5.7.

[Upgrading from version 5.7 to version 5.8](#) on page 1098

Consider the following information when upgrading from Diffusion version 5.7 to version 5.8.

[Upgrading from version 5.8 to version 5.9](#) on page 1102

Consider the following information when upgrading from Diffusion version 5.8 to version 5.9.

[Upgrading to a new patch release](#) on page 1105

When upgrading to a new patch release there are typically no changes to the configuration values or the APIs. All that is required is to copy your existing files from the old installation to the new installation.

[Known issues in Diffusion 5.9](#) on page 1106

Be aware of the following issues when using Diffusion 5.9.

Related reference

[Interoperability](#) on page 1076

If you plan to use different versions of Diffusion servers and clients together, review the following information that summarizes support between versions.

Upgrading from version 5.1 to version 5.5

Consider the following information when upgrading from Diffusion version 5.1 to version 5.5.

Upgrading your applications

Server-side components

Recompile all Java application components that are deployed to the Diffusion server, such as publishers and authorization handlers, against the new version `diffusion.jar` file. This file is located in the `lib` directory of your new Diffusion server installation.

Some features that your Java application components might use have been removed or deprecated. Pay attention to new deprecation warnings and compilation failures that occur during recompilation and review the API changes information in the following section to see if these changes affect your applications.

Event publishers

The event publisher APIs have been removed. Reimplement your event publisher as a client using the Unified API control features. For more information, see .

JMS adapter

The legacy JMS adapter has been deprecated and replaced with a new version. These JMS adapters are not compatible.

To move to the latest JMS adapter, use the `JMSAdapter.xml` configuration file to configure the behavior. Refer to the provided XML and XSD file because the configuration has changed since the previous version. For more information, see [JMSAdapter.xml](#) on page 695.

Note: Not recommended: To continue using the legacy JMS adapter, version 5.1, rename the `JMSAdapter.xml` configuration file used with the previous versions of Diffusion to `JMSAdapter51.xml`.

Clients

You can choose not to recompile your client applications and continue to use client libraries from a previous release. If you choose to use client libraries from a previous release, ensure that the libraries are compatible with the new server. For more information, see [Interoperability](#) on page 1076.

You can choose to upgrade your client applications to use the new client libraries. To do this, recompile the client applications against the client libraries located in the `clients` directory of your new Diffusion server installation and repackage your client application with the new library.

Alternatively, the Java library is available in the [Push Public Maven Repository](#) and the JavaScript library from [NPM](#).

Some features that your client applications might use have been removed or deprecated. Review the API changes information in the following section to see if these changes affect your applications.

Note: Java clients: When you recompile your Java clients with the new version of the libraries, be aware that the Diffusion log framework is no longer included in the Java client library. Applications should add an SLF4J implementation to their classpath, such as Logback, Log4j 2, or the SLF4J bridge to java.util.Logging.

Note: .NET clients: When you recompile your .NET clients with the new version of the libraries, be aware that the latest .NET client libraries require version 4.5 or later of the .NET Framework. In addition, only a single Diffusion DLL is now required to compile a Diffusion .NET client. For more information, see [.NET](#) on page 162.

API changes

Further information about removed or deprecated features is available in following locations:

- The release notes provided in the docs directory of your Diffusion installation or online at <http://docs.pushtechnology.com/docs/5.5.1/release/ReleaseNotice.html>
- The API documentation located at <http://docs.pushtechnology.com/5.5>

The following table lists API classes and methods that have been removed. If you attempt to recompile application code that uses these classes or methods against the version 5.5 APIs, it fails. Rewrite your application code to not include these features.

Table 79: API features removed in version 5.5

API affected	Removed feature	Suggested alternative
Event Publisher API	All	Use the Unified API.
Java API	<code>getConflation</code> method on <code>RootConfig</code>	<code>getConflation</code> method on <code>ServerConfig</code>
.NET Classic API	<code>Credentials</code>	<code>V4Credentials</code> This change was made to disambiguate between the credentials object in the Classic API and that in the Unified API.

The following table lists API classes and methods that have been deprecated. If your application code uses these classes or methods, consider rewriting your application code to not include these features.

Table 80: API features deprecated in version 5.5

API affected	Deprecated feature	Suggested alternative
Java Unified API, .NET Unified API, C Unified API	<code>Session.start()</code> and the associated 'initialising' state. Now a no-op.	The asynchronous 'open' method on <code>SessionFactory</code> which has callback to notify session opened (or error).

API affected	Deprecated feature	Suggested alternative
Java Classic API	<code>Client.setCredentials()</code> , <code>Client.getCredentials()</code>	None
Publisher API	<code>AuthorisationHandler.credentialsSupplied()</code>	None
Java Unified API	<code>TopicSubscriptionHandler</code>	Use routing topics instead
Publisher API	<code>XMLPropertiesListener</code> , <code>XMLProperties</code>	None
Publisher API	<code>MultiplexerConfig.getLoadBalancer()</code> and <code>MultiplexerConfig.setLoadBalancer()</code>	None These methods enable you to specify the load balancing policy for multiplexers. Previously, round robin and least clients policies were available. In future, only the default, round robin, policy will be provided.
Java API	<code>com.pushtechnology.com.api.config</code> and related methods in <code>com.pushtechnology.com.api.config</code>	Instead use the system authentication store or a custom authentication handler to configure remote IM users.
.NET API	<code>Session.getFetchFeature</code> and the Fetch feature	<code>Session.getTopicFeature</code> . The fetch capabilities are included in the Topics feature.
.NET API, Java API	Headers interface	<code>ReceiveContext.getHeaderList()</code> and <code>SendOptions.headers(List<String>)</code> . Headers are now represented in the API as a list of Strings. .
Java API	<code>RootConfig.getMessageLengthSize</code> , <code>RootConfig.setMessageLengthSize</code>	These methods are both deprecated and no-ops. The message length size is now hard-coded to 4 bytes.
Publisher API	<code>TopicProperty.AUTO_SUBSCRIBE</code> property. This is ignored. Auto-subscribe is always true.	None
Unified API	<code>ClientControl.close</code> methods that include a String reason parameter	Use <code>ClientControl.close</code> methods that do not include this parameter. The <code>reason</code> parameter is not passed to the client being closed. If you want to notify the client being closed of the reason for its closure, use the <code>MessagingControl</code> feature to send a message to the client. To ensure that the message is received before closing the client session, wait for callback to return before calling <code>ClientControl.close</code> .

API affected	Deprecated feature	Suggested alternative
Configuration API	<code>writeSelectorConfig</code> This is ignored. Write selectors have been unified with other types of selector.	<code>SelectorThreadPoolConfig</code> All selectors are now drawn from the selector thread pool.

The following list includes behavior that has changed in the API. If your application code relies on the previous behavior, rewrite your application code to take into account the new behavior.

- You can now create a `SessionId` object from the session ID String in the Java Unified API and .NET Unified API.
- The result of calling the `AuthorisationHandler.canWrite()` is no longer cached. If an authorisation handler is registered, it is called every time a client sends a message to the server.

Note: We recommend you use role-base security instead of authorisation handlers.

- Auto-subscription is enabled for every topic and cannot be disabled. If a client attempts to subscribe to a topic that does not exist, the subscription request is saved and when the topic is created, the client is automatically subscribed to the topic.
- The `Publisher.subscription()` method no longer sends a load message for a topic that has state. When a topic with state — that is, a topic whose data type is not `TopicDataType.NONE` — is first subscribed to, the topic load message is sent before `Publisher.subscription()` is called. The default implementation of the `Publisher.subscription()` method does nothing.
- Topic loaders are only called for topics that have no state — that is, topics whose data type is `TopicDataType.NONE`.
- `SlaveTopicData` no longer extends `PublishingTopicData`.
- Updates to slave topics no longer update the master topic.
- Changes to how the number of subscribers to a topic are counted:
 - Slave topics only count subscriptions made directly to the slave topic.

In previous releases, subscriptions to the master topic and other slave topics of the same master were counted as subscriptions to a slave.
 - Master topics count all subscriptions made directly to the master topic and all subscriptions made indirectly through slave topics.
 - Topics subscribed to through a routing topic count both direct subscriptions to the topic and indirect subscriptions through the routing topic.

These changes affect the return values from methods that query whether a topic has subscribers or the number of topic subscribers.

- Classic API clients are no longer required to be subscribed to topic paths that they send messages on. However, a topic must exist at the topic path for a Classic API client to receive a message through the topic path.

Classic API clients are still required to be subscribed to topic paths to receive messages on those topic paths.

Upgrading your server installation

Note:

At release 5.5, the Diffusion server is tested and supported on Java HotSpot Development Kit 8 (latest update).

The Diffusion server also runs on Java HotSpot Development Kit 7. However, Oracle withdrew support for Java 7 in April 2015. We recommend that you move to the latest update of Java 8 as soon as possible.

To upgrade your Diffusion server installation, complete the following steps:

1. Use the graphical or headless installer to install the new version of Diffusion.

For more information, see [Installing the Diffusion server](#) on page 535.

2. You can copy your existing license file from your previous installation to the `etc` directory of your new installation.
3. You can copy most of your existing configuration files from the `etc` directory of your previous installation to the `etc` directory of your new installation. When you do, consider making the following changes:
 - The structure of `JMSAdapter.xml` file has changed. Do not copy your existing `JMSAdapter.xml` file to the `etc` directory of your new installation. Instead copy it to `etc/JMSAdapter51.xml`. This change is because the legacy JMS adapter is deprecated and is replaced with a new version. For more information, see [Configuring the JMS adapter](#) on page 686
 - The `SubscriptionValidationPolicy.xml` configuration file is now deprecated. Use the roles and permissions provided in the new security model to define which clients can subscribe to which topics.
 - Configuring remote JMX users in the `Management.xml` configuration file is now deprecated. Instead use the system authentication store or a custom authentication handler to configure remote JMX users.
 - When configuring log levels in the `Logs.xml` configuration file, use the SLF4J log levels: ERROR, WARN, INFO, DEBUG, or TRACE. The `java.util.Logging` log level values (SEVERE, WARING, INFO, FINE, and FINEST) are still supported, but are deprecated.
 - You can now configure the format of the date in log file names in the `Logs.xml` configuration file by using the optional `<date-format>` element.
 - Configuring the multiplexer load balancing policy in the `Server.xml` configuration file is now deprecated. In future, only one load balancing policy will be provided for multiplexers: the default, round robin, policy.
 - In the `Server.xml` configuration file, when defining the file name of the GeoIP database file you must use an absolute path or a path relative to the Diffusion installation directory. Backwards compatibility with version 4.6 has been removed. You can no longer specify a path relative to the configuration directory.
 - The connector for port 8081 has been removed from the `Connectors.xml` configuration file. If your clients connect on this port, you must either configure this port in `Connectors.xml` or change the port that your clients connect on.
 - You are now required to configure a selector thread pool definition. Update your `Server.xml` configuration file to ensure that it includes this definition.
 - The write selector configuration in the `Server.xml` configuration file is deprecated and any configuration associated with write selectors is ignored. All selectors are now drawn from the same pool. Use the selector thread pool definition elements to define the number and behavior of selectors.

Behavior changes at the Diffusion server

The following list includes behavior that has changed at the server. If your solution relies on the previous behavior, adjust your solution to take into account the new behavior.

- The message fragmentation capability has been removed from Diffusion.

Topic message fragmentation was intended to prevent head-of-line blocking by large messages. The API allowed messages for a given topic messages to be delivered out of order, which is incompatible with snapshot/delta processing.

If you applied topic message fragmentation to work around the maximum message size limitations, particularly for large topic load messages, we recommend instead that you increase the maximum message size to accommodate the largest possible application message.

Increasing the maximum message size to support large topic load messages will also require increasing the client input buffer and server output buffer sizes. The peak memory requirement is lower than needed when topic message fragmentation is enabled, but is approximately twice the maximum message size. In a future release, we will improve the buffer handling to allow the maximum message size to exceed the network buffer size.

The default value of the input buffer size has been increased to 1M.

- The default number of multiplexers has changed from 2 to the number of available processors.
- Changes to how input buffers are allocated improve the performance and memory usage of the Diffusion server.

Input buffers are no longer bound to clients, instead they are shared by all reading tasks. Where previously the number of connected clients defined the number of input buffers, now the maximum number of input buffers is bounded by the configured thread pool size.

The maximum amount of memory used for input buffers is less than the thread pool size multiplied by the input buffer size plus any small memory usage resulting from partial reads.

- It is possible to define a set of selector thread pools and have a connector refer to a member of this set name. If no pool is defined, each connector is assigned a default pool with size = 1.
- Reconnection is enabled by default, with a reconnection timeout of 60 seconds. If you do not want client sessions to be able to reconnect, disable `reconnect` in the `Connectors.xml` configuration file.

When reconnection is enabled, the Diffusion server continues to queue messages for a client session for the whole reconnection period. This can affect performance.

- The `etc` directory of your Diffusion installation is no longer on the classpath. If you have included any files in the `etc` directory that you require to be on the classpath, ensure that these files are included in the classpath by other means:
 - Place resource files such as `hazelcast.xml` that must be on the JVM system classpath in the `data` directory of the Diffusion installation.
 - Place resource files that are only loaded by deployed application code, such as publishers or authentication handlers, in the `ext` directory of the Diffusion installation.
- Wrapper scripts for `jstatd` and `jstack` are no longer provided in the `tools` directory of your Diffusion server installation.
- Messages sent on topic paths are no longer counted in the topic statistics. Only updates to the topic are counted.
- The message length size is no longer configurable. In all cases it is hard-coded to 4 bytes. If you have earlier versions of the Diffusion server as part of your solution, ensure that their message length size is configured to be 4 bytes or the servers will not interoperate.
- The names of the keystores provided in the Diffusion installation have changed.
 - `publicKeys.store` is now `licence.keystore`
 - `keystore` is now `sample.keystore`

The start scripts provided by the Diffusion installation have been updated accordingly. If you use these scripts, you do not need to make any changes.

If you start the Diffusion server from your own scripts, you must update them to take into account this change. Update the following property in the Java command that starts the server to point to the new keystore name:

```
-Ddiffusion.keystore.file=diffusion_installation/etc/
licence.keystore
```

If you start the server from a Java program and supply the keystore name as a VM arguments, update the VM to point to the new keystore name. For more information, see [Running from within a Java application](#) on page 637

- The supported version of Google protocol buffers is now 2.6.1. In previous versions it was 2.4.1.
- Due to the addition of the new JavaScript Unified API, the JavaScript API you might have been using with previous versions of Diffusion is now called the Classic API.

The library for the JavaScript Classic API is now called `diffusion-js-classic-version.js` and the API documentation for it is now located in the `js-classic` folder of the documentation.

Related concepts

[Upgrading from version 4.x to version 5.1](#) on page 1079

Consider the following information when upgrading from Diffusion version 4.x to version 5.1.

[Upgrading from version 5.5 to version 5.6](#) on page 1091

Consider the following information when upgrading from Diffusion version 5.5 to version 5.6.

[Upgrading from version 5.6 to version 5.7](#) on page 1095

Consider the following information when upgrading from Diffusion version 5.6 to version 5.7.

[Upgrading from version 5.7 to version 5.8](#) on page 1098

Consider the following information when upgrading from Diffusion version 5.7 to version 5.8.

[Upgrading from version 5.8 to version 5.9](#) on page 1102

Consider the following information when upgrading from Diffusion version 5.8 to version 5.9.

[Upgrading to a new patch release](#) on page 1105

When upgrading to a new patch release there are typically no changes to the configuration values or the APIs. All that is required is to copy your existing files from the old installation to the new installation.

[Known issues in Diffusion 5.9](#) on page 1106

Be aware of the following issues when using Diffusion 5.9.

Related reference

[Interoperability](#) on page 1076

If you plan to use different versions of Diffusion servers and clients together, review the following information that summarizes support between versions.

Upgrading from version 5.5 to version 5.6

Consider the following information when upgrading from Diffusion version 5.5 to version 5.6.

Upgrading your applications

Server-side components

Recompile all Java application components that are deployed to the Diffusion server, such as publishers and authorization handlers, against the new version `diffusion.jar` file. This file is located in the `lib` directory of your new Diffusion server installation.

Some features that your Java application components might use have been removed or deprecated. Pay attention to new deprecation warnings and compilation failures that occur during recompilation and review the API changes information in the following section to see if these changes affect your applications.

Clients

You can choose not to recompile your client applications and continue to use client libraries from a previous release. If you choose to use client libraries from a previous release, ensure that the libraries are compatible with the new server. For more information, see [Interoperability](#) on page 1076.

You can choose to upgrade your client applications to use the new client libraries. To do this, recompile the client applications against the client libraries located in the `clients` directory of your new Diffusion server installation and repackage your client application with the new library.

Alternatively, the Java library is available in the [Push Public Maven Repository](#) and the JavaScript library from [NPM](#).

Some features that your client applications might use have been removed or deprecated. Review the API changes information in the following section to see if these changes affect your applications.

API changes

Further information about removed or deprecated features is available in following locations:

- The release notes provided in the `docs` directory of your Diffusion installation or online at <http://docs.pushtechnology.com/5.6>
- The API documentation located at <http://docs.pushtechnology.com/5.6>

The following table lists API classes and methods that have been removed. If you attempt to recompile application code that uses these classes or methods against the version 5.6 APIs, it fails. Rewrite your application code to not include these features.

Table 81: API features removed in version 5.6

API affected	Removed feature	Suggested alternative
Java	Methods that navigate up from a configuration item to its parent configuration item. This excludes the <code>getServerConfig</code> method in <code>ReplicationConfig</code> , which is only deprecated at this release.	Instead navigate down from the root configuration item.
Java	Methods that navigate up from a configuration item to its parent configuration item.	Instead navigate down from the root configuration item.
.NET Unified API	The <code>IRecordContentReader.HasMore</code> property.	The <code>HasMoreFields()</code> or <code>HasMoreRecords()</code> methods.
Publisher API	Topic fetch handlers.	Applications can provide fetch results for stateless topics using the <code>Publisher.fetchForClient()</code> API.

The following table lists API classes and methods that have been deprecated. If your application code uses these classes or methods, consider rewriting your application code to not include these features.

Table 82: API features deprecated in version 5.6

API affected	Deprecated feature	Suggested alternative
Unified API	The <code>setSessionDetailsListener</code> , <code>setSessionDetails</code> , and <code>getSessionDetails</code> in <code>ClientControl</code>	Use a session properties listener instead.
Java	The <code>getServerConfig</code> method in <code>ReplicationConfig</code>	Accessors for the parent configuration object are no longer used. Instead navigate down from the parent configuration item.
Java Publisher API	The <code>Management</code> class, which provided JMX Management utilities.	Instead use methods available through the JVM runtime.
Java Publisher API	All topic locking methods and associated properties. <code>TopicProperty.LOCKABLE</code> , <code>TopicProperty.LOCK_TIMEOUT</code> , <code>TopicData.lock()</code> , <code>TopicData.unlock()</code> , <code>TopicData.isLockedByCurrentThread()</code> , <code>Topic.isLockable()</code> , <code>Topic.lock()</code> , <code>Topic.unlock()</code> , <code>Topic.isLockedByCurrentThread()</code> , <code>Topic.setLockTimeout()</code> , and <code>Topic.getLockTimeout()</code>	Topic locking is handled implicitly for stateless topics. Stateful topics can be updated transactionally. This transactional mechanism manages the topic locks.
Java Publisher API	<code>Publisher.publishMessage()</code> methods and <code>Publisher.publishExclusiveMessage()</code>	For stateless topics, use <code>Topic.publishMessage()</code> . For stateful topics (those with topic data), use <code>PublishingTopicData.publishMessage()</code> .
Java Publisher API	<code>TopicStatistics.getInboundMessageStatistics()</code>	<code>TopicStatistics()</code>
Java Publisher API	<code>Client.getFetchReply()</code> , <code>TopicClient.sendFetchReply()</code> This change is part of deprecating <code>fetch</code> for stateless topics.	<code>Publisher.fetchForClient()</code>

The following list includes behavior that has changed in the API. If your application code relies on the previous behavior, rewrite your application code to take into account the new behavior.

- In the C Unified API, the `session_create_async()` no longer returns a session object. To prevent the creation of an invalid pointer the method now returns NULL regardless of success or failure.

Instead retrieve the session object on `_connected` callback passed to `session_create_async()`.
- On Java Unified API clients, new tasks that cannot be added to the inbound thread pool queue because the queue is full block until there is space for them on the queue. This is a change in behavior, as previously the calling thread ran the task instead of blocking, resulting in out of order processing.

Java clients that run more than one client session within the same JVM must increase their inbound thread pool queue size to at least 3 times the number of sessions.

- In the Publisher API, the `TopicClient.subscribe()` and `TopicClient.unsubscribe()` methods now return `false` or an empty list, depending on their return type. In future releases, the return type of the `TopicClient.unsubscribe()` methods will be `void`.
- In the Publisher API, the `Topic.publishMessage()` methods now enforce an order to topic updates made using stateless topics. In previous releases, it was possible for clients to receive updates through stateless topics in different orders.
- In the Publisher API, the `Topic.publishMessage()` methods now check whether the topic is deleted before publishing to it. If the topic is deleted, the method throws an exception.
- In the Publisher API, the `Topic.publishMessage()` methods print a warning to the log file if they are used to update stateful topics, which are topics with topic data. In future releases, using `Topic.publishMessage()` methods this way will cause an exception.
- In the Publisher API, the `PublishingTopicData.publishMessage()` methods print a warning to the log file if they are called outside of an update block. In future releases, using `PublishingTopicData.publishMessage()` methods this way will cause an exception.
- All inbound message statistics now have a value of -1. Topic inbound message statistics are disabled and all related methods deprecated.
- Changes to how the number of subscribers to a topic are counted. Each subscription by another Diffusion server using either fan-out replication or high-availability replication is counted in the total number of subscribers to a topic.

Upgrading your server installation

Note:

At release 5.6, the Diffusion server is tested and supported on Java HotSpot Development Kit 8 (latest update).

The Diffusion server also runs on Java HotSpot Development Kit 7. However, Oracle withdrew support for Java 7 in April 2015. We recommend that you move to the latest update of Java 8 as soon as possible.

To upgrade your Diffusion server installation, complete the following steps:

1. Use the graphical or headless installer to install the new version of Diffusion.

For more information, see [Installing the Diffusion server](#) on page 535.

2. You can copy your existing license file from your previous installation to the `etc` directory of your new installation.
3. You can copy most of your existing configuration files from the `etc` directory of your previous installation to the `etc` directory of your new installation. When you do, consider making the following changes:

- In the `Server.xml` configuration file, replace the `<multiplexers>` element and its child elements with the `<multiplexer>` element and its child elements.

The `<multiplexers>` element is deprecated and will be removed in a future release. For more information, see [Server.xml](#) on page 563.

Behavior changes at the Diffusion server

The following list includes behavior that has changed at the server. If your solution relies on the previous behavior, adjust your solution to take into account the new behavior.

- The `DiffusionServer` class now provides a lifecycle listener. When starting your Diffusion server from within a Java application, you can register the lifecycle listener callback that is notified when the server changes state. For more information, see [Running from within a Java application](#) on page 637.
- Diffusion JMX MBean ObjectNames now follow Oracle best practices. For more information, see <http://www.oracle.com/technetwork/java/javase/tech/best-practices-jsp-136021.html>

Related concepts

[Upgrading from version 4.x to version 5.1](#) on page 1079

Consider the following information when upgrading from Diffusion version 4.x to version 5.1.

[Upgrading from version 5.1 to version 5.5](#) on page 1085

Consider the following information when upgrading from Diffusion version 5.1 to version 5.5.

[Upgrading from version 5.6 to version 5.7](#) on page 1095

Consider the following information when upgrading from Diffusion version 5.6 to version 5.7.

[Upgrading from version 5.7 to version 5.8](#) on page 1098

Consider the following information when upgrading from Diffusion version 5.7 to version 5.8.

[Upgrading from version 5.8 to version 5.9](#) on page 1102

Consider the following information when upgrading from Diffusion version 5.8 to version 5.9.

[Upgrading to a new patch release](#) on page 1105

When upgrading to a new patch release there are typically no changes to the configuration values or the APIs. All that is required is to copy your existing files from the old installation to the new installation.

[Known issues in Diffusion 5.9](#) on page 1106

Be aware of the following issues when using Diffusion 5.9.

Related reference

[Interoperability](#) on page 1076

If you plan to use different versions of Diffusion servers and clients together, review the following information that summarizes support between versions.

Upgrading from version 5.6 to version 5.7

Consider the following information when upgrading from Diffusion version 5.6 to version 5.7.

Upgrading your applications

Server-side components

Recompile all Java application components that are deployed to the Diffusion server, such as publishers and authorization handlers, against the new version `diffusion.jar` file. This file is located in the `lib` directory of your new Diffusion server installation.

Some features that your Java application components might use have been removed or deprecated. Pay attention to new deprecation warnings and compilation failures that occur during recompilation and review the API changes information in the following section to see if these changes affect your applications.

Clients

You can choose not to recompile your client applications and continue to use client libraries from a previous release. If you choose to use client libraries from a previous release, ensure that the libraries are compatible with the new server. For more information, see [Interoperability](#) on page 1076.

You can choose to upgrade your client applications to use the new client libraries. To do this, recompile the client applications against the client libraries located in the `clients` directory of your new Diffusion server installation and repackage your client application with the new library.

Alternatively, the Java library is available in the [Push Public Maven Repository](#) and the JavaScript library from [NPM](#).

Some features that your client applications might use have been removed or deprecated. Review the API changes information in the following section to see if these changes affect your applications.

API changes

Further information about removed or deprecated features is available in following locations:

- The release notes provided online at <http://docs.pushtechology.com/docs/5.9.24/ReleaseNotice.html>
- The API documentation located at <http://docs.pushtechology.com/5.7>

The following table lists API classes and methods that have been removed. If you attempt to recompile application code that uses these classes or methods against the version 5.7 APIs, it fails. Rewrite your application code to not include these features.

Table 83: API features removed in version 5.7

API affected	Removed feature	Suggested alternative
Unified API	<code>Topics.Listener</code> and <code>Messages.Listener</code>	

The following table lists API classes and methods that have been deprecated. If your application code uses these classes or methods, consider rewriting your application code to not include these features.

Table 84: API features deprecated in version 5.7

API affected	Deprecated feature	Suggested alternative
All clients	The HTTP Full Duplex transport	Use WebSocket connections instead.
The C Classic API	All use of this API	Use the C Unified API instead.
All APIs	Service topics and their associated classes and methods.	Use the Messaging and MessagingControl features of the Unified API to send point-to-point requests.
All APIs	Protocol buffer topics and their associated classes and methods.	Use binary topics to send any binary data, including protocol buffers. Handle the serialization and deserialization of those protocol buffers in the clients.

API affected	Deprecated feature	Suggested alternative
All Unified API clients	Content encoding	Use a third-party library to encrypt or compress all or part of the data used to create your message content.
All Unified API clients	<code>getStreamsForTopic()</code> method in the <code>PTDiffusionTopicStreamDelegate</code> class	None
Apple Unified API	<code>diffusionTopicStream:didUnsubscribeFromTopicPa</code> method in the <code>Topics</code> class	<code>diffusionTopicStream:didUnsubscribeFromTopicPa</code>
Configuration API	<code>ClientServiceConfig.isClosingCallbackRequests</code> and <code>ClientServiceConfig.setCloseCallbackRequests</code>	None

The following list includes behavior that has changed in the API. If your application code relies on the previous behavior, rewrite your application code to take into account the new behavior.

- If the connection or write timeouts are configured to be greater than one hour, a warning message is output and a timeout of one hour is used.

Upgrading your server installation

Note:

At release 5.7, the Diffusion server is tested and supported on Java HotSpot Development Kit 8 (latest update).

Java 7 is not supported.

To upgrade your Diffusion server installation, complete the following steps:

1. Use the graphical or headless installer to install the new version of Diffusion.
For more information, see [Installing the Diffusion server](#) on page 535.
2. You can copy your existing license file from your previous installation to the `etc` directory of your new installation.
3. You can copy most of your existing configuration files from the `etc` directory of your previous installation to the `etc` directory of your new installation. When you do, consider making the following changes:
 - In the `Management.xml` file remove the `register-topics` element. This element was deprecated and a no-op, but has now been removed from the schema.
 - In the `Server.xml` file ensure that the connection timeout and write timeout are set to less than one hour. If these values are greater than one hour, a value of one hour is used and a message is written to the log. In future releases, timeouts greater than one hour will not be valid configuration and the server will not start.
 - In the `Security.store`, to maintain the previous security configuration, grant the `select_topic` permission to all sessions that already have `read_topic` permission.

For example, in the default `Security.store` file, change the following lines:

```
set "CLIENT" default topic permissions [READ_TOPIC,
SEND_TO_MESSAGE_HANDLER]
...
set "OPERATOR" topic "Diffusion" permissions [READ_TOPIC,
SEND_TO_MESSAGE_HANDLER]
```

Add the `select_topic` permission to the CLIENT and OPERATOR roles:

```
set "CLIENT" default topic permissions [SELECT_TOPIC,  
    READ_TOPIC, SEND_TO_MESSAGE_HANDLER]  
...  
set "OPERATOR" topic "Diffusion" permissions [SELECT_TOPIC,  
    READ_TOPIC, SEND_TO_MESSAGE_HANDLER]
```

As this configuration grants the CLIENT role to all sessions, the session inherit the `select_topic` permission.

Behavior changes at the Diffusion server

The following list includes behavior that has changed at the server. If your solution relies on the previous behavior, adjust your solution to take into account the new behavior.

- If the connection or write timeouts are configured to be greater than one hour, a warning message is output and a timeout of one hour is used.

Related concepts

[Upgrading from version 4.x to version 5.1](#) on page 1079

Consider the following information when upgrading from Diffusion version 4.x to version 5.1.

[Upgrading from version 5.1 to version 5.5](#) on page 1085

Consider the following information when upgrading from Diffusion version 5.1 to version 5.5.

[Upgrading from version 5.5 to version 5.6](#) on page 1091

Consider the following information when upgrading from Diffusion version 5.5 to version 5.6.

[Upgrading from version 5.7 to version 5.8](#) on page 1098

Consider the following information when upgrading from Diffusion version 5.7 to version 5.8.

[Upgrading from version 5.8 to version 5.9](#) on page 1102

Consider the following information when upgrading from Diffusion version 5.8 to version 5.9.

[Upgrading to a new patch release](#) on page 1105

When upgrading to a new patch release there are typically no changes to the configuration values or the APIs. All that is required is to copy your existing files from the old installation to the new installation.

[Known issues in Diffusion 5.9](#) on page 1106

Be aware of the following issues when using Diffusion 5.9.

Related reference

[Interoperability](#) on page 1076

If you plan to use different versions of Diffusion servers and clients together, review the following information that summarizes support between versions.

Upgrading from version 5.7 to version 5.8

Consider the following information when upgrading from Diffusion version 5.7 to version 5.8.

Upgrading your applications

Server-side components

Recompile all Java application components that are deployed to the Diffusion server, such as publishers and authorization handlers, against the new version `diffusion.jar` file. This file is located in the `lib` directory of your new Diffusion server installation.

Some features that your Java application components might use have been removed or deprecated. Pay attention to new deprecation warnings and compilation failures that occur during recompilation and review the API changes information in the following section to see if these changes affect your applications.

Clients

You can choose not to recompile your client applications and continue to use client libraries from a previous release. If you choose to use client libraries from a previous release, ensure that the libraries are compatible with the new server. For more information, see [Interoperability](#) on page 1076.

You can choose to upgrade your client applications to use the new client libraries. To do this, recompile the client applications against the client libraries located in the `clients` directory of your new Diffusion server installation and repackage your client application with the new library.

Alternatively, the Java library is available in the [Push Public Maven Repository](#) and the JavaScript library from [NPM](#).

Some features that your client applications might use have been removed or deprecated. Review the API changes information in the following section to see if these changes affect your applications.

API changes

Further information about removed or deprecated features is available in following locations:

- The release notes provided online at <http://docs.pushtechology.com/docs/5.9.24/ReleaseNotice.html>
- The API documentation located at <http://docs.pushtechology.com/5.8>

The following table lists API classes and methods that have been removed. If you attempt to recompile application code that uses these classes or methods against the version 5.8 APIs, it fails. Rewrite your application code to not include these features.

Table 85: API features removed in version 5.8

API affected	Removed feature	Suggested alternative
Java Classic API	<code>QueuesConfig</code>	This configuration can now only be done on the Diffusion server
MBeans	<code>MultiplexerLatencyNotification</code> in the <code>Multiplexer</code> MBean and <code>PermissionNotification</code> in the <code>AuthorisationManager</code> MBean	None

The following table lists API classes and methods that have been deprecated. If your application code uses these classes or methods, consider rewriting your application code to not include these features.

Table 86: API features deprecated in version 5.8

API affected	Deprecated feature	Suggested alternative
Configuration API	<code>setPriority</code> and <code>getPriority</code> methods in <code>ThreadPoolConfig</code>	None.
Configuration API	<code>setThreadPriority</code> and <code>getThreadPriority</code> methods in <code>MultiplexerConfig</code>	None
Configuration API	<code>ThreadPoolListenerConfig</code> class	None
Configuration API	<code>setThreadPoolListener</code> , <code>getThreadPoolListener</code> , and <code>removeThreadPoolListener</code> methods in <code>ThreadPoolConfig</code>	None. These methods are now no-ops.
Java Classic API	<code>getInboundThreadPool</code> method in the <code>ThreadService</code> class	None

The following list includes behavior that has changed in the API. If your application code relies on the previous behavior, rewrite your application code to take into account the new behavior.

- For JavaScript, Android, and Java clients that register session properties listeners, the Diffusion server can aggregate the initial batch of notifications into a single message. This decreases the risk of overflow in the message queue that is used to queue messages to be sent to the client. The client session registering the session properties listener might need to use a larger maximum message size to accommodate this message.
- In Android, Java, and .NET, a session properties listener now receives a notification when a client becomes disconnected from the Diffusion server.
- In Android, Java, and .NET, a session properties listener now receives a notification when a client fails over connection to another Diffusion server.
- Clients can subscribe to a routing topic before a routing subscription handler is added. This subscription is now evaluated when a handler is added.
- Publishers no longer receive not acknowledged notifications for messages sent to Unified API clients.

Upgrading your server installation

Note:

At release 5.8, the Diffusion server is tested and supported on Java HotSpot Development Kit 8 (latest update).

Java 7 is not supported.

To upgrade your Diffusion server installation, complete the following steps:

1. Use the graphical or headless installer to install the new version of Diffusion.
For more information, see [Installing the Diffusion server](#) on page 535.
2. You can copy your existing license file from your previous installation to the `etc` directory of your new installation.
3. You can copy most of your existing configuration files from the `etc` directory of your previous installation to the `etc` directory of your new installation. When you do, consider making the following changes:

- In the `Server.xml` file remove the `thread-priority` elements for all types of thread. This element is deprecated and a no-op.
- In the `Server.xml` file remove the `keep-alive` and `priority` elements for thread pool definitions. These elements are deprecated and a no-op.
- In the `WebServer.xml` file ensure that the value of `message-sequence-timeout` is less than 1 hour. Values greater than 1 hour (3600000ms) cause a warning to be logged and the timeout is set to one hour.

This parameter is used to re-order out-of-order messages received over separate HTTP connections opened by client browsers. It is rarely necessary to set this to more than a few tens of seconds.

Behavior changes at the Diffusion server

The following list includes behavior that has changed at the server. If your solution relies on the previous behavior, adjust your solution to take into account the new behavior.

- You can no longer set a thread priority for threads.
- You can no longer set a priority or keep-alive time for a thread pool.
- Configuring max-size for a thread pool is now optional. If no value is defined, the max-size defaults to the core-size.
- You can no longer set the message sequence timeout to more than 1 hour.

Related concepts

[Upgrading from version 4.x to version 5.1](#) on page 1079

Consider the following information when upgrading from Diffusion version 4.x to version 5.1.

[Upgrading from version 5.1 to version 5.5](#) on page 1085

Consider the following information when upgrading from Diffusion version 5.1 to version 5.5.

[Upgrading from version 5.5 to version 5.6](#) on page 1091

Consider the following information when upgrading from Diffusion version 5.5 to version 5.6.

[Upgrading from version 5.6 to version 5.7](#) on page 1095

Consider the following information when upgrading from Diffusion version 5.6 to version 5.7.

[Upgrading from version 5.8 to version 5.9](#) on page 1102

Consider the following information when upgrading from Diffusion version 5.8 to version 5.9.

[Upgrading to a new patch release](#) on page 1105

When upgrading to a new patch release there are typically no changes to the configuration values or the APIs. All that is required is to copy your existing files from the old installation to the new installation.

[Known issues in Diffusion 5.9](#) on page 1106

Be aware of the following issues when using Diffusion 5.9.

Related reference

[Interoperability](#) on page 1076

If you plan to use different versions of Diffusion servers and clients together, review the following information that summarizes support between versions.

Upgrading from version 5.8 to version 5.9

Consider the following information when upgrading from Diffusion version 5.8 to version 5.9.

Upgrading your applications

Server-side components

Recompile all Java application components that are deployed to the Diffusion server, such as publishers and authorization handlers, against the new version `diffusion.jar` file. This file is located in the `lib` directory of your new Diffusion server installation.

Some features that your Java application components might use have been removed or deprecated. Pay attention to new deprecation warnings and compilation failures that occur during recompilation and review the API changes information in the following section to see if these changes affect your applications.

Clients

You can choose not to recompile your client applications and continue to use client libraries from a previous release. If you choose to use client libraries from a previous release, ensure that the libraries are compatible with the new server. For more information, see [Interoperability](#) on page 1076.

You can choose to upgrade your client applications to use the new client libraries. To do this, recompile the client applications against the client libraries located in the `clients` directory of your new Diffusion server installation and repackage your client application with the new library.

Alternatively, the Java library is available in the [Push Public Maven Repository](#) and the JavaScript library from [NPM](#).

Some features that your client applications might use have been removed or deprecated. Review the API changes information in the following section to see if these changes affect your applications.

The Classic API has been deprecated at this release and will be removed in a future release. Consider rewriting all Classic API clients using the Unified API.

API changes

Further information about removed or deprecated features is available in following locations:

- The release notes provided online at <http://docs.pushtechology.com/docs/5.9.24/ReleaseNotice.html>
- The API documentation located at <http://docs.pushtechology.com/docs/5.9>

The following table lists API classes and methods that have been removed. If you attempt to recompile application code that uses these classes or methods against the version 5.9 APIs, it fails. Rewrite your application code to not include these features.

Table 87: API features removed in version 5.9

API affected	Removed feature	Suggested alternative
All APIs	Delegated topics and all related classes and methods, including state providers.	
Java Unified API	<code>Topics.getStreamsForTopic</code>	None
Java Unified API	Headers	None
.NET Unified API	<code>Topics.GetStreamsForTopic</code>	None

The following table lists API classes and methods that have been deprecated. If your application code uses these classes or methods, consider rewriting your application code to not include these features.

Table 88: API features deprecated in version 5.9

API affected	Deprecated feature	Suggested alternative
Classic API	Every Classic client API. The Java Publisher API is not deprecated.	Use the Unified API instead
Java Unified API	In <code>TopicControl</code> : <code>removeTopics</code> , <code>RemoveCallback</code> , and <code>RemoveContextCallback</code>	<code>remove</code> , <code>RemovalCallback</code> , and <code>RemovalContextCallback</code> These changes correspond to a change in behavior that allows topics to be deleted without also deleting their child topics in the topic tree.
Apple Unified API	<code>PTDiffusionSessionErrorHandler</code> and <code>PTDiffusionSessionDefaultErrorHandler</code>	
.NET Unified API	All session details methods in <code>IClientControl</code>	Use session properties instead.
Authorization handler API	<code>canSubscribe()</code>	Use declarative role-based security instead.

The following list includes behavior that has changed in the API. If your application code relies on the previous behavior, rewrite your application code to take into account the new behavior.

- There are additional reasons why client sessions can become unsubscribed from a topic:
 - **AUTHORIZATION.** A client session is unsubscribed from a topic with this reason when the session principal changes and the roles assigned to that session no longer contain permissions for that topic.

SUBSCRIPTION_REFRESH. A client session is unsubscribed from a topic with this reason when the topic type or attributes have changed. For example, if the session connection fails over to another Diffusion server.

- When creating a topic at a multi-part topic path, for example A/B/C, the intermediate paths, A and A/B, remain empty and topics can be created at these paths at a later point.

In previous releases, stateless topics were created at these intermediate paths. If your application relies on these stateless topics being present, you must now explicitly create them.

Upgrading your server installation

Note:

At release 5.9, the Diffusion server is tested and supported on Java HotSpot Development Kit 8 (latest update).

Java 7 is not supported.

To upgrade your Diffusion server installation, complete the following steps:

1. Use the graphical or headless installer to install the new version of Diffusion.

For more information, see [Installing the Diffusion server](#) on page 535.

2. You can copy your existing license file from your previous installation to the `etc` directory of your new installation.
3. You can copy most of your existing configuration files from the `etc` directory of your previous installation to the `etc` directory of your new installation. When you do, consider making the following changes:
 - In `Connectors.xml` the default value has changed from `classic` to `all`. If you did not specify an `api-type` element and relied upon the default behavior, you must now specify a value of `classic` to restrict Unified API connections through a connector.
 - In the `Management.xml` file, consider removing the `users` and `assigned-roles` elements, which are now deprecated. Instead use roles and permissions defined in the `Security.store`.

Behavior changes at the Diffusion server

The following list includes behavior that has changed at the server. If your solution relies on the previous behavior, adjust your solution to take into account the new behavior.

- By default, connectors now accept connections from both Classic API clients and Unified API clients. Previously the default was to only accept connections from Classic API clients. If you relied upon this behavior to restrict connections from Unified API client, you must now explicitly define an `api-type` element with a value `classic`.

Related concepts

[Upgrading from version 4.x to version 5.1](#) on page 1079

Consider the following information when upgrading from Diffusion version 4.x to version 5.1.

[Upgrading from version 5.1 to version 5.5](#) on page 1085

Consider the following information when upgrading from Diffusion version 5.1 to version 5.5.

[Upgrading from version 5.5 to version 5.6](#) on page 1091

Consider the following information when upgrading from Diffusion version 5.5 to version 5.6.

[Upgrading from version 5.6 to version 5.7](#) on page 1095

Consider the following information when upgrading from Diffusion version 5.6 to version 5.7.

[Upgrading from version 5.7 to version 5.8](#) on page 1098

Consider the following information when upgrading from Diffusion version 5.7 to version 5.8.

[Upgrading to a new patch release](#) on page 1105

When upgrading to a new patch release there are typically no changes to the configuration values or the APIs. All that is required is to copy your existing files from the old installation to the new installation.

[Known issues in Diffusion 5.9](#) on page 1106

Be aware of the following issues when using Diffusion 5.9.

Related reference

[Interoperability](#) on page 1076

If you plan to use different versions of Diffusion servers and clients together, review the following information that summarizes support between versions.

Upgrading to a new patch release

When upgrading to a new patch release there are typically no changes to the configuration values or the APIs. All that is required is to copy your existing files from the old installation to the new installation.

To upgrade to a new patch release, complete the following steps:

1. Use the graphical or headless installer to install the new version of Diffusion.

For more information, see [Installing the Diffusion server](#) on page 535.

2. Copy your existing license file from your previous installation to the `etc` directory of your new installation.
 3. Copy your existing configuration files from the `etc` directory of your previous installation to the `etc` directory of your new installation.
 4. Copy any publishers located in the `ext` directory of the previous installation into the `ext` directory of the new installation.
-

Related concepts

[Upgrading from version 4.x to version 5.1](#) on page 1079

Consider the following information when upgrading from Diffusion version 4.x to version 5.1.

[Upgrading from version 5.1 to version 5.5](#) on page 1085

Consider the following information when upgrading from Diffusion version 5.1 to version 5.5.

[Upgrading from version 5.5 to version 5.6](#) on page 1091

Consider the following information when upgrading from Diffusion version 5.5 to version 5.6.

[Upgrading from version 5.6 to version 5.7](#) on page 1095

Consider the following information when upgrading from Diffusion version 5.6 to version 5.7.

[Upgrading from version 5.7 to version 5.8](#) on page 1098

Consider the following information when upgrading from Diffusion version 5.7 to version 5.8.

[Upgrading from version 5.8 to version 5.9](#) on page 1102

Consider the following information when upgrading from Diffusion version 5.8 to version 5.9.

[Known issues in Diffusion 5.9](#) on page 1106

Be aware of the following issues when using Diffusion 5.9.

Related reference

[Interoperability](#) on page 1076

If you plan to use different versions of Diffusion servers and clients together, review the following information that summarizes support between versions.

Known issues in Diffusion 5.9

Be aware of the following issues when using Diffusion 5.9.

Publishers cannot send to Unified API clients via Messaging

When a publisher sends a message to a Unified API client, the messages are not routed to the client's Messaging feature as expected but instead are delivered as if they were updates via topic streams.

Messages sent in this way do not have headers. If the messages are set as requiring acknowledgment, they are not acknowledged and cause a nack callback even though they have been delivered to the correct client, though through the wrong mechanism.

JMS adapter can duplicate messages received from Diffusion at configuration change

During configuration change there is a window during which both old and new JMS connections are in place and can deliver messaging from Diffusion clients to JMS destinations. A message sent from a Diffusion client to a topic associated with a JMS destination that is unaffected by a configuration change (i.e. it is neither removed or added) during the configuration change window is duplicated.

Latest value not notified for Record topics when new TopicStream added

If the Topics feature is used to add a new TopicStream that covers Record topics that are already subscribed to then the stream will be notified of subscription to the topic but will not be given the latest value for the topic and therefore would be unable to process subsequent deltas. For this reason, when using Record topics, it is important that the stream that will process the topics is added before the topics are subscribed to.

Issues with message loss on reconnect for Classic API clients and any clients using DPT

After reconnecting a C Unified API client to the same server it is possible that there could be some message loss. When using single value topics this may not be a problem as a full value is always delivered but for topics that deliver deltas this could mean that deltas are missed and therefore the current state is unknown. For this reason it is not recommended to use reconnection with C Unified API clients.

This issue will not be resolved. We recommend you do not use the DPT transport if you have reconnection configured. We recommend you use Unified API clients instead of Classic API clients.

Related concepts

[Upgrading from version 4.x to version 5.1](#) on page 1079

Consider the following information when upgrading from Diffusion version 4.x to version 5.1.

[Upgrading from version 5.1 to version 5.5](#) on page 1085

Consider the following information when upgrading from Diffusion version 5.1 to version 5.5.

[Upgrading from version 5.5 to version 5.6](#) on page 1091

Consider the following information when upgrading from Diffusion version 5.5 to version 5.6.

[Upgrading from version 5.6 to version 5.7](#) on page 1095

Consider the following information when upgrading from Diffusion version 5.6 to version 5.7.

[Upgrading from version 5.7 to version 5.8](#) on page 1098

Consider the following information when upgrading from Diffusion version 5.7 to version 5.8.

[Upgrading from version 5.8 to version 5.9](#) on page 1102

Consider the following information when upgrading from Diffusion version 5.8 to version 5.9.

[Upgrading to a new patch release](#) on page 1105

When upgrading to a new patch release there are typically no changes to the configuration values or the APIs. All that is required is to copy your existing files from the old installation to the new installation.

Related reference

[Interoperability](#) on page 1076

If you plan to use different versions of Diffusion servers and clients together, review the following information that summarizes support between versions.

Appendix

Appendices

The appendices contain reference information.

In this section:

- [Document conventions](#)
- [Glossary](#)
- [Trademarks](#)
- [Copyright Notices](#)

Appendix

A

Document conventions

This user manual uses certain typographic conventions to distinguish between different types of information.

The following table describes how different types of information are represented typographically.

Table 89: Typographic conventions used in this manual

Convention	Usage
Monospace	Indicates the following items: <ul style="list-style-type: none">• Source code• Class or method names• Command names• File paths• Information input by the user
Bold	Indicates the following items: <ul style="list-style-type: none">• Interface element titles (buttons, menu items, field names)• Window or panel titles
<i>Italic</i>	Indicates the following items: <ul style="list-style-type: none">• New terms – when appearing in the text• Variable values – when appearing in code or syntax examples
Greater than sign (>)	Indicates a menu item or sequence of menu items. For example, “Choose File > Save ” means choose the Save item from the File menu.
Highlighting	Indicates an example value in descriptive text.

Appendix

B

Glossary

The glossary contains key terms associated with Diffusion and their definitions.

In this section:

- [A](#)
- [C](#)
- [D](#)
- [E](#)
- [F](#)
- [G](#)
- [H](#)
- [I](#)
- [J](#)
- [L](#)
- [M](#)
- [N](#)
- [P](#)
- [Q](#)
- [R](#)
- [S](#)
- [T](#)
- [U](#)
- [V](#)
- [W](#)
- [X](#)

A

acknowledgment

A message sent by the recipient of a message or update to inform the sender of that message or update that it was received. Not all messages or updates are acknowledged. A message or update must specify that acknowledgment is required to receive an acknowledgment.

ack

API

Application Programming Interface

A set of contracts that you can program against to interact with Diffusion.

The following APIs are available:

- Publisher API
- Unified API
- Classic API

API

API

API

APNS

Apple Push Notification Service

Apple Push Notification Service (APNS)

APNS

APNS

APR

Apache Portable Runtime

Apache Portable Runtime (APR)

APR

APR

ASCII

American Standard Code for Information Interchange

A character-encoding scheme that encodes 128 specified characters into 7-bit binary integers.

ASCII

ASCII

ASCII

C

callback

An object, specific to a single call, that is used to respond to a request.

CBOR

Concise Binary Object Representation

Concise Binary Object Representation (CBOR)

CBOR

CBOR

certify

Push Technology certifies specific hardware and software version for use with Diffusion. Certified versions have been fully functional tested and performance tested with the Diffusion server. In addition, Push Technology supports some hardware and software that has not been certified.

client

An entity that connects to the Diffusion server and subscribes to topics.

Typically a client is a user-written application communicating with the Diffusion server through a client API or the Diffusion protocol. A publisher can be a client of another publisher in a distributed environment.

client library

A library that is included in a client application to enable interaction with the Diffusion server.

conflation

The merging or replacing of a queued update with a newer update to reduce network traffic. Conflation removes outdated information from the queue of content to be sent and either replaces the outdated information with the conflated information or appends the conflated information to the end of the queue.

connector

A configured point of connection to a server.

There can be one or more connectors (each listening on a different port). Each connector can accept single or multiple types of connection.

consume

When a message or update is received by a topic listener and that listener chooses not to pass on the message or update to subsequent topic listeners. Topic streams cannot choose to consume messages or updates they receive.

CORS

Cross-Origin Resource Sharing
cross-origin resource sharing (CORS)

CORS

CORS

CPU

Central Processing Unit
CPU

CPU

CPU

credentials

A piece of information that is used to authenticate a principal.

CSR

Certificate Signing Request
certificate signing request (CSR)

CSR

CSR

CSS

Cascading Style Sheet
CSS

CSS

CSS

D

DAR file

A Diffusion archive file. This file contains a publisher and can be deployed on the Diffusion server.
DAR

DAR

DAR

delimiter

A byte value that is used as a separator in messages or updates.

Depending on the type of message or update, it can contain field delimiters or record delimiters.

delta

Data that is sent to a client subscribed to a topic. The delta contains only information that has been updated on the topic since the last data was sent to the client. Topics that contain only a single item of data cannot use delta messages.

Diffusion

The Diffusion product comprising the Diffusion server and client libraries.

dirty

The state of a page in a paged topic at the client if its content has been changed on the Diffusion server, but not updated on the client.

DLL

Dynamic Link Library

DLL

DLL

DLL

DOM

Document Object Model

DOM

DOM

DOM

DMZ

De-militarized Zone

de-militarized zone (DMZ)

DMZ

DMZ

DPT

Diffusion Protocol for TCP DEPRECATED: A Diffusion proprietary streaming protocol that clients can use to communicate with the Diffusion server.

DPT

Diffusion Protocol for TCP

DPT

E

EULA

End User License Agreement

EULA

EULA

EULA

F

feature

An API module that contains a conceptual set of facilities.

fetch

A request from a client for the current state of all data on a topic. A client can fetch a topic's state without being subscribed to the topic. This request-response mechanism of getting data from a topic is separate from the pub-sub mechanism.

flow control

A mechanism within a Diffusion client that limits the rate that client sends messages as the load level from that client increases. A client application rapidly making thousands of calls to the Diffusion server might overflow the internal queues, which results in the client session being closed. Flow control protects against these queues overflowing by progressively delaying messages from the client to the Diffusion server.

field

A section of content that contains data of a specific type. Fields are nested inside records. A record can contain one or many fields.

functional topic

A topic to which data cannot be published. These topics provide other capabilities to subscribing clients, for example, notifications.

The following types of topic are functional topics:

- [Routing](#)
- [Child list](#)
- [Service](#)
- [Topic notify](#)

G

GBE

Gigabit Ethernet

GBE

GBE

GBE

GCM

Google Cloud Messaging

Google Cloud Messaging (GCM)

GCM

GCM

global-scoped permission

Permissions at global scope apply to actions on the Diffusion server.

GUI

Graphical User Interface

GUI

GUI

GUI

H

handler

A handler is an object responsible for responding to one or more instances of a single type of request.

HDD

Hard Disk Drive

HDD

HDD

HDD

HTML

Hypertext Markup Language

HTML

HTML

HTML

HTTP

Hypertext Transfer Protocol

HTTP

HTTP

HTTP

I

IDE

Integrated Development Environment

integrated development environment (IDE)

IDE

IDE

ISAPI

Internet Server Application Programming Interface

ISAPI

initial topic load

The data sent to a client when it first subscribes to a topic. This data contains the value of the current state of the topic.

initial topic load (ITL)

ITL

ITL

J

JAR

Java Archive

JAR

JAR

JAR

JDK

Java Development Kit

Java Development Kit (JDK)

JDK

JDK

JMS

Java Message Service

Java Message Service (JMS)

JMS

JMS

JMX

Java Management Extensions

Java Management Extensions (JMX)

JMX

JMX

JRE

Java Runtime Environment

Java Runtime Environment (JRE)

JRE

JRE

JSON

JavaScript Object Notation

JavaScript Object Notation (JSON)

JSON

JSON

JVM

Java Virtual Machine

Java Virtual Machine (JVM)

JVM

JVM

L

LDAP

Lightweight Directory Access Protocol

Lightweight Directory Access Protocol (LDAP)

LDAP

LDAP

line

An entry in a page of a paged topic. The number of lines on a page is specified by the client that views the paged topic.

listener

In the Unified API, a listener is an object that is always called when a particular event occurs.

M

message

A message is a series of bytes of information formatted according to the Diffusion protocol which can be sent between components utilizing Diffusion.

message queue

A queue of messages. Each client connection to Diffusion has such a queue on the Diffusion server upon which messages are put for sending to the client.

queue

metadata

Data about data. In Diffusion metadata is used to define the structure of messages.

message metadata

multicast

To send data to several recipients at the same time.

The datagrid uses multicasting to locate other datagrid nodes.

N

NAT

Network Address Translation
network address translation (NAT)

NAT

NAT

NIC

Network Interface Controller
NIC

NIC

NIC

NIO

New Input-Output
NIO

NIO

Non-blocking Input/Output

NIO

notification

...

P

paged topic

A topic whose data is formatted in lines and viewed as pages containing one or more lines.

The following types of topic are paged topics:

- [Paged string](#)
- [Paged record](#)

PNG

Portable Network Graphics
PNG

PNG

PNG

permission

A permission represents the right to perform an action on the Diffusion server or on data hosted by the Diffusion server. Permissions can be global- or topic-scoped.

PID

Process ID

PID

PID

PID

ping

A query sent by a publisher, client or by the Diffusion server to a connected component to check that the connection exists and the latency of the connection.

The following types of ping are available:

server ping

A client pings the Diffusion server.

client ping

A publisher pings a specific client.

system ping

Diffusion pings all clients at a regular interval.

PDF

Portable Document Format

PDF

PDF

PDF

primary server

In a fan-out solution, the server from which updates are fanned out to replica servers.

In previous releases, this server was called the master server. This terminology is no longer used.

master server

principal

An identity that can be authenticated by the Diffusion server or by a client.

A principal can be a user or client. After a principal has been authenticated, it can be assigned roles that enable it to access actions or resources.

protocol

A protocol defines the exact format of data passed between the Diffusion server and a client.

publisher

The component which publishes messages relating to one or more topics.

A server can host one or more publishers. Messages sent by clients on particular topics are routed to the Publisher that owns the topic. Publisher functionality is provided by users by writing a Java publisher class.

publishing topic

A topic where data is published and from which the data is distributed to subscribing clients.

The following types of topic are publishing topics:

- [Single value](#)
- [Record](#)
- [Stateless](#)
- [Protocol buffer](#) (Deprecated)
- [Custom](#)

push notification destination

An endpoint, described by either an APNS device token or a GCM registration ID, where push notifications are received.

Q

message queue

A queue of messages. Each client connection to Diffusion has such a queue on the Diffusion server upon which messages are put for sending to the client.

queue

R

RAID

Redundant Array of Independent Disks

RAID

RAID

RAID

RAM

Random Access Memory

RAM

RAM

RAM

record

A section of content that acts as a container for a set of fields. Inside the content of a message or update you can have one or many records. A record can contain one or many fields.

regular expression

A string that uses special characters to describe a search pattern.

Diffusion uses Java-style regular expressions.

regex

replica server

In a fan-out solution, a server to which updates are fanned out from the primary server.

In previous releases, this server was called the slave server. This terminology is no longer used.

slave server

RMI

Remote Method Invocation

remote method invocation (RMI)

RMI

RMI

role

A role is a named set of permissions and other roles. Principals and sessions can both be assigned roles.

role hierarchy

Roles are hierarchical. A role can include other roles and, by doing so, have the permissions assigned to the included roles. A role cannot include itself, either directly or indirectly – through a number of included roles.

RPM

Redhat Package Manager

Redhat Package Manager (RPM)

RPM

RPM

S

SAS

Serial Attached SCSI

SAS

SAS

SAS

SDK

Software Development Kit

software development kit (SDK)

SDK

SDK

server

The component that hosts topics and publishers. A server broadcasts topic updates to all subscribed clients.

Clients can connect to servers through the Unified API or Classic API (deprecated).

Diffusion server

session

An ongoing dialog between a client and the Diffusion server.

Typically, a session represents a single client connection to a single server. However, in the event of connection failure the session can automatically reconnect to the same server or even fail over to another server and still retain its context.

session will

A set of actions to be completed after a session closes.

A client session can specify actions that are completed by the Diffusion server that the session connects to after the session has closed. A session will can be used to close or tidy up topics managed or updated by the client session.

will

SLF4J

Simple Logging Facade for Java

SLF4J

SLF4J

SLF4J

SSH

Secure Shell

SSH

SSH

SSH

SSL

Secure Sockets Layer

Secure Sockets Layer (SSL)

SSL

SSL

state

The latest published values of all data items on the topic. The state of a topic is stored on the Diffusion server.

stateful topic

A topic that stores a current value as topic data on the Diffusion server.

stateless topic

A topic that does not store a current value on the Diffusion server.

structural conflation

A form of conflation that enables you to define the operations performed on outdated content. You can merge, aggregate, reverse or combine the effects of multiple changes into a single consistent and current notification to the client.

stream

In the Unified API, a stream is a sequence of responses to a single request.

subscribe

A client registers interest in a topic such that the client receives messages sent to that topic.

support

Push Technology supports a number of hardware and software versions, these versions have not necessarily been tested. Those hardware and software versions that we have tested are listed as 'certified'.

T

TCP

Transmission Control Protocol

TCP

TCP

TCP

throttling

Limiting the volume of messages that the Diffusion server transmits to a client within a specified period of time.

Throttling can be used to limit bandwidth usage or to prevent more messages being sent to a client than the client can handle.

TLS

Transport Layer Security

Transport Layer Security (TLS)

TLS

TLS

topic

A logical channel through which messages are distributed.

Topics provide a logical link between publishers and subscribers. Clients or publishers publish messages to topics. Clients subscribe to topics to receive messages published to that topic.

topic path

A string representation of a location in the topic tree.

A topic path consists of parts separated by a slash character (/).

Topic paths describe a location where a topic can be bound and used for pub-sub distribution of data.

Topic paths can also be used for bi-directional messaging: a client can send a message to a topic path and the Diffusion server routes the message to the topic's publisher or publishers.

topic name

hierarchic topic name

full topic name

topic path prefix

The root part of a topic selector.

A concrete topic path to the most specific part of the topic tree that contains all topics that the selector can specify. For example, for the topic selector `?foo/bar/baz/.*/bing`, the topic path prefix is `foo/bar/baz`.

path prefix

topic selector

An object that retrieves one or more topics based on their topic paths.

A topic selector uses a pattern expression, which can include one or more regular expressions, to match to the path of one or more topics.

selector

topic-scoped permission

Permissions at topic scope apply to actions on a topic.

Topic-scoped permissions are defined against topic branches. The permissions that apply to a topic are the set of permissions defined at the most specific branch of the topic tree.

topic tree

The organization structure of topics on the Diffusion server.

A topic can have subtopics and can itself be a subtopic of another topic. All topics created on the Diffusion server by a publisher or client are in the topic tree.

topic hierarchy

transport

An implementation of a network protocol. The mechanism by which clients communicate with the Diffusion server.

U

update

Data published to a topic by a client or publisher that is applied to the topic to change the topic state. The updated data is then pushed out to all subscribing clients.

URL

Uniform Resource Locator

URL

URL

URL

UTF-8

Universal Character Set Transformation Format 8-bit

A character encoding capable of encoding all possible characters in Unicode.

UTF-8

UTF-8

UTF-8

V

VCPU

Virtual Central Processing Unit

VCPU

VCPU

VCPU

W

WAR

Web Application Archive

WAR

WAR

WAR

X

XHR

XmlHttpRequest

XHR

XHR

XHR

XML

Extensible Markup Language

XML

XML

XML

XSD

XML Schema Definition

XSD

XSD

XSD

Appendix

C

Trademarks

The following trademarked terms are included in this manual.

Diffusion is trademark of Push Technology Ltd.

ActionScript, Adobe, Flash, and Flexare registered trademarks of Adobe Systems Incorporated.

AIX™, Bluemix®, Cast Iron®, and WebSphere® are trademarks of IBM.

Amazon and Amazon EC2 are trademarks of Amazon.

Android and Chrome are trademarks of Google Inc.

Ant, Apache, Apache Derby™, Apache Tomcat™, and Maven are trademarks of The Apache Software Foundation.

Apple, Mac®, macOS, Safari, and Siri® are registered trademarks of Apple Inc.

BlackBerry® is a registered trademark of RIM.

CentOS and Red Hat are trademarks or registered trademarks of Red Hat, Inc.

Dell™ is trademark of Dell, Inc.

Docker is trademarks or registered trademarks of Docker, Inc. in the United States and/or other countries.

Eclipse is a trademark of the Eclipse Foundation, Inc.

F5 is a registered trademark of F5 Networks, Inc.

Firefox is a registered trademark of Mozilla Foundation.

Hazelcast is a trademark of Hazelcast Inc.

Intel and Xeon are trademarks of Intel Corporation.

Internet Explorer, Microsoft, Silverlight, and Windows are trademarks or registered trademarks of Microsoft Corporation.

iOS is a registered trademark of Cisco.

Java, JavaScript, Oracle, and Solaris™ are trademarks or registered trademarks of Oracle Corporation.

Joyent is a trademarks of Joyent.

Linux is a trademark of Linus Torvalds.

Nagios[®] is a registered trademark of Nagios Enterprises.

Node.js is a trademark of Joyent, Inc.

Opera is a registered trademark of Opera Software ASA.

Splunk is a trademark of Splunk, Inc.

SUSE[®] is a registered trademark of SUSE LLC.

TIBCO Enterprise Message Service is a trademark of TIBCO Software Inc.

Ubuntu is a registered trademark of Canonical Ltd.

UNIX is a registered trademark of The Open Group.

VeriSign[®] is a registered trademark of VeriSign, Inc.

VMware[®] and VMware vSphere are registered trademarks of VMware, Inc.

Appendix

D

Copyright Notices

Diffusion uses third party, open source software. The rights to this software are not owned by Push Technology and the software is distributed under different licensing agreements. The distribution and use of third-party software is subject to the applicable terms.

The following sections list the software used, their licenses, copyright notices and disclaimers.

In this section:

- [ANTLR](#)
- [apns](#)
- [Apache Commons Codec](#)
- [Apache Portable Runtime](#)
- [Bootstrap](#)
- [CocoaAsyncSocket](#)
- [concurrent-trees](#)
- [CQEngine](#)
- [cron4j](#)
- [d3](#)
- [disruptor](#)
- [FastColoredTextBox](#)
- [Fluent validation](#)
- [Fluidbox](#)
- [gcm-server](#)
- [GeoIP API](#)
- [GeoLite City Database](#)
- [geronimo-jms_1.1_spec](#)
- [Google code prettify](#)
- [hashmap](#)
- [Hazelcast](#)
- [HPPC](#)
- [htmlcompressor](#)

- inherits
- jackson-core
- jackson-dataformat-cbor
- JCIP Annotations
- JCTools
- jQuery
- json-simple
- JZlib
- Knockout
- libwebsockets
- log4j2
- loglevel
- long
- Metrics
- Minimal JSON
- Modernizr
- NLog
- openscv
- OpenSSL
- PCRE
- Picocontainer
- Protocol Buffers
- Rickshaw
- Servlet API
- SLF4J
- slf4j-android-logger
- SocketRocket
- Tabber
- Tapestry (Plastic)
- TrueLicense
- when
- ws
- Licenses

ANTLR

Version 4.3

<http://www.antlr.org>

ANTLR is distributed under the [BSD 3-clause License](#).

Copyright (c) 2014 Terence Parr, Sam Harwell

apns

Version 1.0.0.Beta6

<https://github.com/notnoop/java-apns/>

apns is distributed under the [BSD 3-clause License](#).

Copyright (c) 2009 Mahmood Ali

Apache Commons Codec

Version 1.8

<http://commons.apache.org/codec/>

Apache Commons Codec is distributed under the [Apache License 2.0](#).

Copyright 2002-2011 The Apache Software Foundation. All Rights Reserved

Additional notices

The following information is included in the `NOTICE.txt` file that accompanies the source:

```
Apache Commons Codec
Copyright 2002-2011 The Apache Software Foundation
```

```
This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).
```

```
-----
src/test/org/apache/commons/codec/language/DoubleMetaphoneTest.java
contains
test data from http://aspell.sourceforge.net/test/batch0.tab.
```

```
Copyright (C) 2002 Kevin Atkinson (kevina@gnu.org). Verbatim copying
and distribution of this entire article is permitted in any medium,
provided this notice is preserved.
-----
```

Apache Portable Runtime

Version 8.3.3

<http://apr.apache.org>

APR is distributed under the [Apache 2.0 License](#).

Copyright (c) 2015 The Apache Software Foundation

Bootstrap

Version: 3.2.0

<https://github.com/twbs/bootstrap/>

Bootstrap is distributed under the [MIT License](#).

Copyright (c) 2011-2014 Twitter, Inc

Additional notes

We also use [Glyphicons](#), which are included as part of Bootstrap.

CocoaAsyncSocket

Version 7.3.4

<https://github.com/robbiehanson/CocoaAsyncSocket>

The CocoaAsyncSocket project is in the public domain.

The original TCP version (AsyncSocket) was created by Dustin Voss in January 2003.

Updated and maintained by Deusty LLC and the Apple development community.

concurrent-trees

Version 2.4.0

<https://github.com/npgall/concurrent-trees>

concurrent-trees is distributed under the [Apache 2.0 License](#).

Copyright 2012-2013 Niall Gallagher

CQEngine

Version 1.2.6

<https://github.com/npgall/cqengine>

CQEngine is distributed under the [Apache 2.0 License](#).

Copyright 2012-2015 Niall Gallagher

cron4j

Version 2.2.5

<http://www.sauronsoftware.it/projects/cron4j/>

Cron4j is distributed under the [LGPL 2.1](#).

Copyright (C) 2007-2010 Carlo Pelliccia (www.sauronsoftware.it)

Source code is available from the following location: <http://sourceforge.net/projects/cron4j/files/cron4j/2.2.5/cron4j-2.2.5.zip/download/>

For a fee, Push Technology can also provide this source on a CD. To request a copy, contact support@pushtechonology.com.

d3

Version 3.2.1

<http://d3js.org/>

d3 is distributed under the [Apache License 2.0](#).

Copyright (c) 2010-2014, Michael Bostock

disruptor

Version 3.3.5

<https://github.com/LMAX-Exchange/disruptor>

disruptor is distributed under the [Apache License 2.0](#).

Copyright 2011 LMAX Ltd.

FastColoredTextBox

<https://github.com/PavelTorgashov/FastColoredTextBox>

FastColoredTextBox distributed under the [LGPL 3.0](#) or later.

Copyright (C) Pavel Torgashov, 2011-2014.

Source code is available at the following location: <https://github.com/PavelTorgashov/FastColoredTextBox>

For a fee, Push Technology can also provide this source on a CD. To request a copy, contact support@pushtechonology.com.

Fluent validation

Version 3.3.1.0

<http://fluentvalidation.codeplex.com/>

Fluent validation is distributed under the [Apache License 2.0](#).

Copyright Jeremy Skinner

Fluidbox

<https://github.com/terrymun/Fluidbox>

Fluidbox is distributed under the [MIT License](#).

Copyright (c) 2014 [Terry Mun](#)

[gcm-server](#)

Version 1.0.0

<https://github.com/google/gcm/>

gcm-server is distributed under the [Apache License 2.0](#).

Copyright 2012 Google Inc. All rights reserved.

[GeoIP API](#)

Version 1.2.13

<http://www.maxmind.com/en/opensource>

The GeoIP API is distributed under the [LGPL 2.1](#) or later.

Copyright (C) 2003 MaxMind LLC. All Rights Reserved

[GeoLite City Database](#)

<http://www.maxmind.com/en/opensource>

The GeoLite City Database is distributed under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#).

Copyright MaxMind LLC

[geronimo-jms_1.1_spec](#)

Version 1.1

<http://geronimo.apache.org/>

geronimo-jms_1.1_spec is distributed under the [Apache License 2.0](#)

Copyright 2003-2006 The Apache Software Foundation

Additional notices

The following information is included in the NOTICE.txt file that accompanies the source:

```
Apache Geronimo  
Copyright 2003-2006 The Apache Software Foundation
```

```
This product includes software developed by  
The Apache Software Foundation (http://www.apache.org/).
```

Google code prettify

<https://github.com/google/code-prettify>

Prettify is distributed under the [Apache 2.0 License](#).

Copyright (C) 2006 Google Inc.

hashmap

Version: 2.0.3

<https://github.com/flesler/hashmap>

hashmap is distributed under the [MIT License](#).

Copyright (c) 2012-2013 Ariel Flesler aflesler@gmail.com

Hazelcast

Version 3.6.4

<http://www.hazelcast.org/>

Hazelcast is distributed under the [Apache License 2.0](#)

Copyright (c) 2008-2016, Hazelcast, Inc. All Rights Reserved.

Additional notices

The following information is included in the NOTICE.txt file that accompanies the source:

```
**
** NOTICE file corresponding to the section 4 (d) of the Apache
** License,
** Version 2.0, in this case for the Hazelcast distribution.
**
```

The end-user documentation included with a redistribution, if any, must include the following acknowledgement:

```
"This product includes software developed by the Hazelcast
Project (http://www.hazelcast.com)."
```

Alternately, this acknowledgement may appear in the software itself, if and wherever such third-party acknowledgements normally appear.

The name "Hazelcast" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact talip@hazelcast.com or fuad@hazelcast.com

Copyright (c) 2008-2014 Hazelcast Inc. All rights reserved.

HPPC

Version 0.7.1

<http://labs.carrotsearch.com/hppc.html>

HPPC is distributed under the [Apache License 2.0](#).

Copyright 2010-2013, Carrot Search s.c., Boznicza 11/56, Poznan, Poland

htmlcompressor

Version 1.5.2

<http://code.google.com/p/htmlcompressor/>

The htmlcompressor is distributed under the [Apache License 2.0](#).

Copyright 2009-2011 Sergiy Kovalchuk

Additional notices: [Apache License 2.0 Notice](#)

inherits

Version 2.0.1

<https://github.com/isaacs/inherits>

Inherits is distributed under the [ISC License](#).

Copyright (c) Isaac Z. Schlueter.

jackson-core

Version 2.7.1

<https://github.com/fasterxml/jackson-core>

jackson-core is distributed under the [Apache License 2.0](#)

Copyright Tatu Saloranta

jackson-dataformat-cbor

Version 2.7.1

<https://github.com/fasterxml/jackson-dataformat-cbor>

jackson-dataformat-cbor is distributed under the [Apache License 2.0](#)

Copyright Tatu Saloranta

JCIP Annotations

Version 1

<https://github.com/stephenc/jcip-annotations>

jcip-annotations is distributed under the [Apache License 2.0](#)

Copyright 2013 Stephen Connolly.

JCTools

Version 1.1

<https://github.com/JCTools/JCTools>

JCTools is distributed under the [Apache License 2.0](#)

Copyright 2015 Nitsan Wakart.

jQuery

Version: 1.7.1

<https://jquery.org/>

jQuery is distributed under the [MIT License](#).

Copyright 2014 jQuery Foundation and other contributors <http://jquery.com/>

json-simple

Version 1.1.1

<https://github.com/fangyidong/json-simple>

json-simple is distributed under the [Apache License 2.0](#).

Copyright (c) Yidong Fang, Chris Nokleberg

JZlib

Version 1.02

<http://www.jcraft.com/jzlib/>

JZlib is distributed under the [BSD 3-clause License](#). This has not been modified, it has been compiled from the source code for distribution.

Copyright 2000-2011 ymnk, JCraft, Inc. All rights reserved.

Knockout

Version 2.1.0

<http://knockoutjs.com/>

Knockout is distributed under the [MIT License](#).

Copyright (c) Steven Sanderson, the Knockout.js team, and other contributors

libwebsockets

Version 1.7.7

<https://libwebsockets.org/index.html>

libwebsockets is distributed under the [LGPL 2.1](#).

Copyright (C) 2010-2015 Andy Green <andy@warmcat.com>

Source code is available from the following location: <https://github.com/warmcat/libwebsockets>

For a fee, Push Technology can also provide this source on a CD. To request a copy, contact support@pushtech.com.

log4j2

Version 2.4.1

<http://logging.apache.org/log4j/2.x/>

log4j2 is distributed under the [Apache License 2.0](#).

Copyright The Apache Software Foundation. All Rights Reserved.

loglevel

Version: 1.4.0

<https://github.com/pimterry/loglevel>

loglevel is distributed under the [MIT License](#).

Copyright (c) 2013 Tim Perry

long

Version: 2.2.5

<https://github.com/dcodeIO/Long.js>

long is distributed under the [Apache License 2.0](#).

Copyright 2013 Daniel Wirtz dcode@dcode.io

Copyright 2009 The Closure Library Authors. All Rights Reserved.

Metrics

Version 3.0.0-BETA

<http://metrics.codahale.com/>

Metrics is distributed under the [Apache License 2.0](#).

Copyright (c) 2010-2013 Coda Hale, Yammer.com

Additional notices

The following information is included in the NOTICE.txt file that accompanies the source:

```
Metrics
```

```
Copyright 2010-2013 Coda Hale and Yammer, Inc.
```

```
This product includes software developed by Coda Hale and Yammer, Inc.
```

```
This product includes code derived from the JSR-166 project  
(ThreadLocalRandom, Striped64,  
LongAdder), which was released with the following comments:
```

```
Written by Doug Lea with assistance from members of JCP JSR-166  
Expert Group and released to the public domain, as explained at  
http://creativecommons.org/publicdomain/zero/1.0/
```

Minimal JSON

<https://github.com/ralfstx/minimal-json>

Minimal JSON is distributed under the [MIT License](#).

Copyright (c) 2014, 2015 EclipseSource

Modernizr

Version: 2.8.3

<http://modernizr.com/>

Modernizr is distributed under the [MIT License](#) and [BSD 3-clause License](#).

NLog

Version 3.1.0

<https://github.com/NLog/NLog/>

NLog is distributed under the [BSD 3-clause License](#).

Copyright (c) 2004-2011 Jaroslaw Kowalski <jaak@jkowalski.net>

opencsv

Version 2.3

<http://opencsv.sourceforge.net/>

opencsv is distributed under the [Apache License 2.0](#).

Copyright 2005 Bytecode Pty Ltd.

OpenSSL

Version 1.0.2a

<https://www.openssl.org/>

OpenSSL is distributed under the [OpenSSL and SSLeay Licenses](#).

PCRE

Version 1.5.2

<http://www.pcre.org/>

PCRE is distributed under the [BSD 3-clause License](#).

THE BASIC LIBRARY FUNCTIONS

Written by: Philip Hazel

Email local part: ph10

Email domain: cam.ac.uk

University of Cambridge Computing Service,
Cambridge, England.

Copyright (c) 1997-2015 University of Cambridge
All rights reserved.

PCRE2 JUST-IN-TIME COMPILATION SUPPORT

Written by: Zoltan Herczeg

Email local part: hzmester

Email domain: freemail.hu

Copyright(c) 2010-2015 Zoltan Herczeg
All rights reserved.

STACK-LESS JUST-IN-TIME COMPILER

Written by: Zoltan Herczeg

Email local part: hzmester

Email domain: freemail.hu

Copyright(c) 2009-2015 Zoltan Herczeg
All rights reserved.

Picocontainer

Version 2.15

<http://picocontainer.codehaus.org/>

Picocontainer is distributed under the [BSD 3-clause License](#).

Copyright (c) 2003-2008 PicoContainer Organization. All rights reserved.

Protocol Buffers

Version 2.6.1

<https://github.com/google/protobuf>

Google's Protocol Buffers are distributed under the [BSD 3-clause License](#).

Copyright 2008 Google Inc. All rights reserved.

Rickshaw

<http://code.shutterstock.com/rickshaw/>

Rickshaw is distributed under the [MIT License](#).

Copyright (C) 2011-2013 by Shutterstock Images, LLC

Servlet API

<http://jetty.mortbay.org/project/modules/servlet-api-2.5>

Servlet API is distributed under the [CDDL v1.0 License](#).

SLF4J

Version 1.7.7

<http://www.slf4j.org/>

SLF4J is distributed under the [MIT License](#).

Copyright (c) 2004-2013 QOS.ch All rights reserved.

slf4j-android-logger

Version 1.0.4

<https://github.com/PSDev/slf4j-android-logger>

slf4j-android-logger is distributed under the [Apache 2.0 License](#).

Copyright 2013 Philip Schiffer

SocketRocket

Version 0.3.1-beta2

<https://github.com/square/SocketRocket>

SocketRocket is distributed under the [Apache License 2.0](#).

Copyright 2012 Square Inc.

Tabber

Version: 1.9

<http://www.barelyfitz.com/projects/tabber/>

Tabber is distributed under the [MIT License](#).

Copyright (c) 2006 Patrick Fitzgerald pat@barelyfitz.com

Tapestry (Plastic)

Version 5.3.7

<http://tapestry.apache.org/>

Tapestry is distributed under the [Apache License 2.0](#).

Copyright 2011, 2012 The Apache Software Foundation

Additional notices

The following information is included in the NOTICE.txt file that accompanies the source:

```
This product includes software developed by  
The Apache Software Foundation (http://www.apache.org/).
```

```
Please refer to the NOTICE.txt in each sub-module to  
identify further dependencies.
```

```
The Maven central repository is the preferred method to download  
Tapestry  
and its dependencies. The binary archive includes just basic  
dependencies for tapestry-core; using other modules (such as  
tapestry-hibernate or any of the others) requires downloading  
additional dependencies. Please refer to the Maven POM for each module  
to identify its dependencies.
```

TrueLicense

Version 1.33

<http://truelicense.java.net/>

This version of TrueLicense was distributed under the [EPL](#).

Copyright 2005-2012 Schlichtherle IT Services

when

Version 3.7.3

<https://github.com/cujojs/when>

<http://cujojs.com/>

When is distributed under the [MIT License](#).

Copyright (c) 2011 Brian Cavalier

WS

Version 0.8.0

<https://github.com/websockets/ws>

WS is distributed under the [MIT License](#).

Copyright (c) 2011 Einar Otto Stangvik

Licenses

The following licenses are used by the third party, open source software that is distributed with Diffusion.

Apache License 2.0

Apache License

Version 2.0, January 2004

<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

Related concepts

[Apache Commons Codec](#) on page 1134

[Apache Portable Runtime](#) on page 1134

[CQEngine](#) on page 1135

[concurrent-trees](#) on page 1135

[d3](#) on page 1136

[disruptor](#) on page 1136
[Fluent validation](#) on page 1136
[geronimo-jms_1.1_spec](#) on page 1137
[gcm-server](#) on page 1137
[Hazelcast](#) on page 1138
[HPPC](#) on page 1139
[htmlcompressor](#) on page 1139
[jackson-core](#) on page 1139
[jackson-dataformat-cbor](#) on page 1139
[JCIP Annotations](#) on page 1140
[JCTools](#) on page 1140
[json-simple](#) on page 1140
[log4j2](#) on page 1141
[long](#) on page 1141
[Metrics](#) on page 1142
[Google code prettify](#) on page 1138
[opencsv](#) on page 1143
[slf4j-android-logger](#) on page 1144
[SocketRocket](#) on page 1145
[Tapestry \(Plastic\)](#) on page 1145

BSD 3-clause License

Copyright (c) <YEAR>, <OWNER>

All rights reserved.

Note: The copyright statement above is included in its completed form in the sections of this document specific to the individual products covered by this license.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The names of the authors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL JCRAFT, INC. OR ANY CONTRIBUTORS TO THIS SOFTWARE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER

CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Related concepts

[ANTLR](#) on page 1134

[apns](#) on page 1134

[Protocol Buffers](#) on page 1144

[JZlib](#) on page 1140

[NLog](#) on page 1142

[Modernizr](#) on page 1142

[PCRE](#) on page 1143

[Picocontainer](#) on page 1144

Common Development and Distribution License

Sun Microsystems, Inc. ("Sun") ENTITLEMENT for SOFTWARE

Licensee/Company: Entity receiving Software.

Effective Date: Date of delivery of the Software to You.

Software: JavaMail 1.4.

License Term: Perpetual (subject to termination under the SLA).

Licensed Unit: Software Copy.

Licensed unit Count: Unlimited.

Permitted Uses:

1. You may reproduce and use the Software for Individual, Commercial, or Research and Instructional Use for the purposes of designing, developing, testing, and running Your applets and application("Programs").

2. Subject to the terms and conditions of this Agreement and restrictions and exceptions set forth in the Software's documentation, You may reproduce and distribute portions of Software identified as a redistributable in the documentation ("Redistributable"), provided that:

(a) you distribute Redistributable complete and unmodified and only bundled as part of Your Programs,

(b) your Programs add significant and primary functionality to the Redistributable,

(c) you distribute Redistributable for the sole purpose of running your Programs,

(d) you do not distribute additional software intended to replace any component(s) of the Redistributable,

(e) you do not remove or alter any proprietary legends or notices contained in or on the Redistributable.

(f) you only distribute the Redistributable subject to a license agreement that protects Sun's interests consistent with the terms contained in this Agreement, and

(g) you agree to defend and indemnify Sun and its licensors from and against any damages, costs, liabilities, settlement amounts and/or expenses (including attorneys' fees) incurred in connection with any claim, lawsuit or action by any third party that arises or results from the use or distribution of any and all Programs and/or Redistributable.

3. Java Technology Restrictions. You may not create, modify, or change the behavior of, or authorize your licensees to create, modify, or change the behavior of, classes, interfaces, or subpackages that are in any way identified as "java", "javax", "sun" or similar convention as specified by Sun in any naming convention designation.

B. Sun Microsystems, Inc. ("Sun") SOFTWARE LICENSE AGREEMENT

READ THE TERMS OF THIS AGREEMENT ("AGREEMENT") CAREFULLY BEFORE OPENING SOFTWARE MEDIA PACKAGE. BY OPENING SOFTWARE MEDIA PACKAGE, YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCESSING SOFTWARE ELECTRONICALLY, INDICATE YOUR ACCEPTANCE OF THESE TERMS BY SELECTING THE "ACCEPT" BUTTON AT THE END OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL OF THE TERMS, PROMPTLY RETURN THE UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR, IF SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE "DECLINE" (OR "EXIT") BUTTON AT THE END OF THIS AGREEMENT. IF YOU HAVE SEPARATELY AGREED TO LICENSE TERMS ("MASTER TERMS") FOR YOUR LICENSE TO THIS SOFTWARE, THEN SECTIONS 1-5 OF THIS AGREEMENT ("SUPPLEMENTAL LICENSE TERMS") SHALL SUPPLEMENT AND SUPERSEDE THE MASTER TERMS IN RELATION TO THIS SOFTWARE.

1. Definitions.

(a) "Entitlement" means the collective set of applicable documents authorized by Sun evidencing your obligation to pay associated fees (if any) for the license, associated Services, and the authorized scope of use of Software under this Agreement.

(b) "Licensed Unit" means the unit of measure by which your use of Software and/or Service is licensed, as described in your Entitlement.

(c) "Permitted Use" means the licensed Software use(s) authorized in this Agreement as specified in your Entitlement. The Permitted Use for any bundled Sun software not specified in your Entitlement will be evaluation use as provided in Section 3.

(d) "Service" means the service(s) that Sun or its delegate will provide, if any, as selected in your Entitlement and as further described in the applicable service listings at www.sun.com/service/servicelist.

(e) "Software" means the Sun software described in your Entitlement. Also, certain software may be included for evaluation use under Section 3.

(f) "You" and "Your" means the individual or legal entity specified in the Entitlement, or for evaluation purposes, the entity performing the evaluation.

2. License Grant and Entitlement.

Subject to the terms of your Entitlement, Sun grants you a nonexclusive, nontransferable limited license to use Software for its Permitted Use for the license term. Your Entitlement will specify (a) Software licensed, (b) the Permitted Use, (c) the license term, and (d) the Licensed Units.

Additionally, if your Entitlement includes Services, then it will also specify the (e) Service and (f) service term.

If your rights to Software or Services are limited in duration and the date such rights begin is other than the purchase date, your Entitlement will provide that beginning date(s).

The Entitlement may be delivered to you in various ways depending on the manner in which you obtain Software and Services, for example, the Entitlement may be provided in your receipt, invoice or your contract with Sun or authorized Sun reseller. It may also be in electronic format if you download Software.

3. Permitted Use.

As selected in your Entitlement, one or more of the following Permitted Uses will apply to your use of Software. Unless you have an Entitlement that expressly permits it, you may not use Software for any of the other Permitted Uses. If you don't have an Entitlement, or if your Entitlement doesn't cover additional software delivered to you, then such software is for your Evaluation Use.

- (a) Evaluation Use. You may evaluate Software internally for a period of 90 days from your first use.
- (b) Research and Instructional Use. You may use Software internally to design, develop and test, and also to provide instruction on such uses.
- (c) Individual Use. You may use Software internally for personal, individual use.
- (d) Commercial Use. You may use Software internally for your own commercial purposes.
- (e) Service Provider Use. You may make Software functionality accessible (but not by providing Software itself or through outsourcing services) to your end users in an extranet deployment, but not to your affiliated companies or to government agencies.

4. Licensed Units.

Your Permitted Use is limited to the number of Licensed Units stated in your Entitlement. If you require additional Licensed Units, you will need additional Entitlement(s).

5. Restrictions.

(a) The copies of Software provided to you under this Agreement are licensed, not sold, to you by Sun. Sun reserves all rights not expressly granted. (b) You may make a single archival copy of Software, but otherwise may not copy, modify, or distribute Software. However if the Sun documentation accompanying Software lists specific portions of Software, such as header files, class libraries, reference source code, and/or redistributable files, that may be handled differently, you may do so only as provided in the Sun documentation. (c) You may not rent, lease, lend or encumber Software. (d) Unless enforcement is prohibited by applicable law, you may not decompile, or reverse engineer Software. (e) The terms and conditions of this Agreement will apply to any Software updates, provided to you at Sun's discretion, that replace and/or supplement the original Software, unless such update contains a separate license. (f) You may not publish or provide the results of any benchmark or comparison tests run on Software to any third party without the prior written consent of Sun. (g) Software is confidential and copyrighted. (h) Unless otherwise specified, if Software is delivered with embedded or bundled software that enables functionality of Software, you may not use such software on a stand-alone basis or use any portion of such software to interoperate with any program(s) other than Software. (i) Software may contain programs that perform automated collection of system data and/or automated software updating services. System data collected through such programs may be used by Sun, its subcontractors, and its service delivery partners for the purpose of providing you with remote system services and/or improving Sun's software and systems. (j) Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility and Sun and its licensors disclaim any express or implied warranty of fitness for such uses. (k) No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this Agreement.

6. Term and Termination.

The license and service term are set forth in your Entitlement(s). Your rights under this Agreement will terminate immediately without notice from Sun if you materially breach it or take any action in derogation of Sun's and/or its licensors' rights to Software. Sun may terminate this Agreement should any Software become, or in Sun's reasonable opinion likely to become, the subject of a claim of intellectual property infringement or trade secret misappropriation. Upon termination, you will cease use of, and destroy, Software and confirm compliance in writing to Sun. Sections 1, 5, 6, 7, and 9-15 will survive termination of the Agreement.

7. Java Compatibility and Open Source.

Software may contain Java technology. You may not create additional classes to, or modifications of, the Java technology, except under compatibility requirements available under a separate agreement available at www.java.net.

Sun supports and benefits from the global community of open source developers, and thanks the community for its important contributions and open standards-based technology, which Sun has adopted into many of its products.

Please note that portions of Software may be provided with notices and open source licenses from such communities and third parties that govern the use of those portions, and any licenses granted hereunder do not alter any rights and obligations you may have under such open source licenses, however, the disclaimer of warranty and limitation of liability provisions in this Agreement will apply to all Software in this distribution.

8. Limited Warranty.

Sun warrants to you that for a period of 90 days from the date of purchase, as evidenced by a copy of the receipt, the media on which Software is furnished (if any) will be free of defects in materials and workmanship under normal use. Except for the foregoing, Software is provided "AS IS". Your exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace Software media or refund the fee paid for Software. Some states do not allow limitations on certain implied warranties, so the above may not apply to you. This limited warranty gives you specific legal rights. You may have others, which vary from state to state.

9. Disclaimer of Warranty.

UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

10. Limitation of Liability.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. In no event will Sun's liability to you, whether in contract, tort (including negligence), or otherwise, exceed the amount paid by you for Software under this Agreement. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose. Some states do not allow the exclusion of incidental or consequential damages, so some of the terms above may not be applicable to you.

11. Export Regulations.

All Software, documents, technical data, and any other materials delivered under this Agreement are subject to U.S. export control laws and may be subject to export or import regulations in other countries. You agree to comply strictly with these laws and regulations and acknowledge that you have the responsibility to obtain any licenses to export, re-export, or import as may be required after delivery to you.

12. U.S. Government Restricted Rights.

If Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in Software and accompanying documentation will be only as set forth in this Agreement; this is in accordance with 48 CFR 227.7201 through 227.7202-4 (for Department of Defense (DOD) acquisitions) and with 48 CFR 2.101 and 12.212 (for non-DOD acquisitions).

13. Governing Law.

Any action related to this Agreement will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.

14. Severability.

If any provision of this Agreement is held to be unenforceable, this Agreement will remain in effect with the provision omitted, unless omission would frustrate the intent of the parties, in which case this Agreement will immediately terminate.

15. Integration.

This Agreement, including any terms contained in your Entitlement, is the entire agreement between you and Sun relating to its subject matter. It supersedes all prior or contemporaneous oral or written communications, proposals, representations and warranties and prevails over any conflicting or additional terms of any quote, order, acknowledgment, or other communication between the parties relating to its subject matter during the term of this Agreement. No modification of this Agreement will be binding, unless in writing and signed by an authorized representative of each party.

Please contact Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, California 95054 if you have questions.

Related concepts

[Servlet API](#) on page 1144

Eclipse Public License – v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

- a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
- b) in the case of each subsequent Contributor:
 - i) changes to the Program, and
 - ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

- a) it complies with the terms and conditions of this Agreement; and
- b) its license agreement:
 - i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
 - ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
 - iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
 - iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

- a) it must be made available under this Agreement; and
- b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

Related concepts

[TrueLicense](#) on page 1145

ISC License –

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Related concepts

[inherits](#) on page 1139

The GNU Lesser General Public License, version 2.1 (LGPL-2.1)

GNU Lesser General Public License

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software – to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages – typically libraries – of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

- a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
- b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.
- c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
- d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
- e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Related concepts

[cron4j](#) on page 1135

[GeoIP API](#) on page 1137

[libwebsockets](#) on page 1141

The GNU Lesser General Public License, version 3.0 (LGPL-3.0)

This license is a set of additional permissions added to version 3 of the GNU General Public License.

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
- b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

d) Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

Related concepts

[FastColoredTextBox](#) on page 1136

The MIT License (MIT)

Copyright (c) <year> <copyright holders>

Note: The copyright statement above is included in its completed form in the sections of this document specific to the individual products covered by this license.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Related concepts

[Bootstrap](#) on page 1135

[Fluidbox](#) on page 1136

[hashmap](#) on page 1138

[jQuery](#) on page 1140

[Knockout](#) on page 1141

[loglevel](#) on page 1141

[Modernizr](#) on page 1142

[Rickshaw](#) on page 1144

[SLF4J](#) on page 1144

[Tabber](#) on page 1145

[Minimal JSON](#) on page 1142

[when](#) on page 1146

[ws](#) on page 1146

OpenSSL and SSLeay Licenses

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit. See below for the actual license texts.

OpenSSL License

Copyright (c) 1998-2011 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)"
4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.
5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.
6. Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Original SSLeay License

Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)

All rights reserved.

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com). The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as the following conditions are aheared to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

"This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)"

The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related :-).

4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement: "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The licence and distribution terms for any publically available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

Related concepts

[OpenSSL](#) on page 1143
