MigratoryData Client API for C++

Developer's Guide and Reference Manual

July 12, 2018

Contents

Chapter 1

Developer's Guide

This guide includes the following sections:

1.1 Overview

This Application Programming Interface (API) contains all the necessary operations for connecting to a cluster of one or more MigratoryData servers, subscribing to subjects, getting real-time messages for the subscribed subjects, and publishing real-time messages.

Before reading this manual, it is recommended to read *MigratoryData Architecture Guide* (PDF, HTML).

1.2 Creating C++ clients for MigratoryData Server

A typical API usage is as follows:

1.2.1 Step 1 - Include the library.

Include the headers of the API located in the folder `include` of this API package.

Add the `include` folder to the *Include Directories* of your C++ application.

The API library is available in the folder `lib` of this API package. Add the `lib` folder to the *Library Directories* of your C++ application. Also, add the API library itself to the list of *Library Dependencies* of your C++ application.

1.2.2 Step 2 - Define the log listener class for processing the log messages.

The log listener class should implement the MigratoryDataLogListener interface.

Use the API call MigratoryDataClient.setLogListener() to assign an instance of the log listener class for getting the logs of the API.

1.2.3 Step 3 - Define the listener class for processing the real-time messages and status notifications.

The listener class should implement the MigratoryDataListener interface.

Use the API call MigratoryDataClient.setListener() to attach your listener implementation.

1.2.4 Step 4 - Specify the list of MigratoryData servers where to connect to.

Use the API method MigratoryDataClient.setServers() to specify one or more MigratoryData servers to which your C++ client will connect to. In fact, the C++ client will connect to only one of the MigratoryData servers in this list. But, defining two or more MigratoryData servers is recommended to achieve fail-over (and horizontal scalling / load balancing). Supposing the MigratoryData server - to which the C++ client connected - goes down, then the API will automatically reconnect that client to another MigratoryData server in the list.

1.2.5 Step 5 Subscribe to subjects and publish messages

Use the API method MigratoryDataClient.subscribe() to subscribe to subjects and use the API method MigratoryDataClient.publish() to publish messages.

1.2.6 Step 6 - Handle the real-time messages and status notifications.

Handle the messages received for the subscribed subjects as well as the status notifications in your listener implementation defined at Step 3 above.

1.3 Examples

Examples built with this API are available in the folder `examples` of this API package; start with the README file which explains how to compile and run them.

Chapter 2

Deprecated List

Member MigratoryDataClient::NOTIFY_PUBLISH_NO_SUBSCRIBER

no more is use.

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 MigratoryDataClient Class Reference

This class implements all the necessary operations for connecting to a cluster of one or more MigratoryData servers, subscribing to subjects, getting real-time messages for the subscribed subjects, and publishing real-time messages.

Public Member Functions

- MigratoryDataClient ()

  Create a MigratoryDataClient object.
- void setLogListener (MigratoryDataLogListener *logListener)

  Define a log listener object.
- void setLogLevel (MigratoryDataLogLevel logLevel)

  Configure the logging level.
- void setEncryption (bool encryption)

  Configure whether to use SSL/TLS encryption when connecting to a MigratoryData server.
- void setListener (MigratoryDataListener *listener)

  Attach a MigratoryDataListener for handling real-time messages and status notifications.
- void setServers (std::vector< std::string > &servers)

  Specify a cluster of one or more MigratoryData servers to which the client will connect to.
- void subscribe (std::vector< std::string > &subjects)

  Subscribe to one or more subjects.
- void unsubscribe (std::vector< std::string > &subjects)

  Unsubscribe from one or more subjects.
- void setEntitlementToken (std::string &token)

  Assign an authorization token to the client.
- void getSubjects (std::vector< std::string > &subjects)

  Return the list of subscribed subjects.
- void notifyBeforeReconnectRetries (int n)

  Define the number of failed attempts to connect to one or more MigratoryData servers before triggering a status notification NOTIFY_SERVER_DOWN.
- void notifyWhenReconnectRateExceedsThreshold (int n)

  Define the number of reconnect attempts to one or more MigratoryData servers per 3-minute window before triggering a status notification NOTIFY_RECONNECT_RATE_EXCEEDED.
- void setQuickReconnectInitialDelay (int seconds)

Define the number of seconds to wait before attempting to reconnect to the cluster after a connection failure is detected.

- void setQuickReconnectMaxRetries (int retries)

  Define the maximum number of retries for the Quick Reconnect phase.
- void setReconnectPolicy (std::string policy)

  Define the reconnect policy to be used after the Quick Reconnect phase.
- void setReconnectTimeInterval (int seconds)

  Define the time interval used for the reconnect schedule after the Quick Reconnect phase.
- void setReconnectMaxDelay (int seconds)

  Define the maximum reconnect delay for the TRUNCATED_EXPONENTIAL_BACKOFF policy.
- void publish (MigratoryDataMessage &message)

  Publish a message.
- void disconnect ()

  Disconnect from the connected MigratoryData server and dispose the resources used by the connection.
- virtual ~MigratoryDataClient ()

  Destructor.

Public Attributes

- const std::string NOTIFY_SERVER_UP

  Indicate that the client successfully connected to a MigratoryData server.
- const std::string NOTIFY_SERVER_DOWN

  Indicate that the client failed to connect to a MigratoryData server.
- const std::string NOTIFY_DATA_SYNC

  After a failover reconnection, the client synchronized a subscribed subject with the latest message available for that subject, as well as with all messages published during the failover for that subject.
- const std::string NOTIFY_DATA_RESYNC

  After a failover reconnection, the client synchronized a subscribed subject with the latest message available for that subject, but not with the potential messages published during the failover, therefore behaving as a new client.
- const std::string NOTIFY_SUBSCRIBE_ALLOW

  Indicate that the client was authorized to subscribe to a subject.
- const std::string NOTIFY_SUBSCRIBE_DENY

  Indicate that the client was not authorized to subscribe to a subject.
- const std::string **NOTIFY_SUBSCRIBE_TIMEOUT**
- const std::string NOTIFY_PUBLISH_DENIED

  Indicate that the client was unable to publish a message because it is not allowed by your entitlement rules.
- const std::string NOTIFY_PUBLISH_NO_SUBSCRIBER

  Indicate that the client was unable to publish a message because there is no client subscribed to the subject of the message.
- const std::string NOTIFY_PUBLISH_OK

  Indicate that the client successfully published a message.
- const std::string NOTIFY_PUBLISH_FAILED

  Indicate that the client was unable to publish a message.
- const std::string NOTIFY_RECONNECT_RATE_EXCEEDED

  Indicate that the reconnect rate threshold per 3-minute window has been reached.
- const std::string CONSTANT_WINDOW_BACKOFF

  A constant used to define the reconnect policy.
- const std::string TRUNCATED_EXPONENTIAL_BACKOFF

  A constant used to define the reconnect policy.

5.1.1 Detailed Description

This class implements all the necessary operations for connecting to a cluster of one or more MigratoryData servers, subscribing to subjects, getting real-time messages for the subscribed subjects, and publishing real-time messages.

5.1.2 Member Function Documentation

5.1.2.1 setLogListener()

```
void MigratoryDataClient::setLogListener (
            MigratoryDataLogListener * logListener )
```

Define a log listener object.

The *log listener* object should belong to a class which implements the MigratoryDataLogListener interface.

It is advisable to configure this API call first if you want to log as much as possible.

Parameters

logListener	An instance of a class which implements the MigratoryDataLogListener interface

5.1.2.2 setLogLevel()

```
void MigratoryDataClient::setLogLevel (
            MigratoryDataLogLevel logLevel )
```

Configure the logging level.

Parameters

logLevel	The logging verbosity (MigratoryDataLogLevel.LOG_ERROR, MigratoryDataLogLevel.LOG_INFO, MigratoryDataLogLevel.LOG_DEBUG or MigratoryDataLogLevel.LOG_TRACE); by default MigratoryDataLogLevel.LOG_INFO is configured.

5.1.2.3 setEncryption()

```
void MigratoryDataClient::setEncryption (
            bool encryption )
```

Configure whether to use SSL/TLS encryption when connecting to a MigratoryData server.

When using encryption you have to connect to the ports of the MigratoryData servers that are configured to listen for encrypted connections. See the parameter ListenEncrypted in the MigratoryData Configuration Guide

Parameters

encryption	Determine whether the client connects to the MigratoryData server using an encrypted SSL/TLS connection

5.1.2.4 setListener()

```
void MigratoryDataClient::setListener (
            MigratoryDataListener * listener )
```

Attach a MigratoryDataListener for handling real-time messages and status notifications.

Parameters

listener	An instance of a class which implements the MigratoryDataListener interface

5.1.2.5 setServers()

```
void MigratoryDataClient::setServers (
            std::vector< std::string > & servers )
```

Specify a cluster of one or more MigratoryData servers to which the client will connect to.

If you specify two or more MigratoryData servers, then all these MigratoryData servers should provide the same level of data redundancy, by making available for subscription the same set of subjects. This is required for achieving (weighted) load balancing, failover, and guaranteed message delivery of the system. In this way, the MigratoryData servers of the `servers` list form a *cluster*.

For example, to connect to a cluster formed of two MigratoryData servers installed at the addresses `p1.example.com` and `p2.example.com`, and configured to accept clients on the standard HTTP port `80`, the following code can be used:

```
vector<string> servers;
servers.push_back("p1.example.com:80");
servers.push_back("p2.example.com:80");
client->setServers(servers);
```

To achieve load-balancing, the API connects the client to a MigratoryData server chosen randomly from the `servers` list. In this way, the load is balanced among all the members of the cluster.

> Moreover, the API supports weighted load-balancing. This feature is especially useful if the MigratoryData servers in the cluster are installed on machines with different capacities. You can assign to each member of the cluster a *weight* ranging from `0` to `100`. This weight assignment is a hint provided to the API to select with a higher probability a MigratoryData server with a higher weight either initially when the client connects to the cluster or later during a failover reconnection.
>
> Supposing the address `p1.example.com` corresponds to a machine that is twice more powerful than the machine having the address `p2.example.com`, then you can assign to `p1.example.com` a weight `100` and to `p2.example.com` a weight `50` by prefixing each address with the assigned weight as follows:

```
vector<string> servers;
 servers.push_back("100 p1.example.com:80");
 servers.push_back("50 p2.example.com:80");
 client->setServers(servers);
```

The API assigns a default weight `100` to the addresses not prefixed with a specific weight.

To achieve failover, if the connection between the client and a MigratoryData server is broken, then the API will automatically detect the failure and will select another MigratoryData server from the `servers` list. If the client fails to connect to the new selected server, a status notification NOTIFY_SERVER_DOWN will be triggered (unless you modify the number of failed attempts with MigratoryDataClient.notifyBeforeReconnectRetries()), and a new MigratoryData server in the cluster will be selected again and again until the client will be able to connect to one of the MigratoryData servers in the cluster. When successfully connected, the API will trigger a status notification NOTIFY_SERVER_UP.

Furthermore, if guaranteed message delivery is enabled, then the potential messages published for a subscribed subject during the failover period, will be automatically retrieved from the cache of the MigratoryData server to which the client reconnects to and a status notification NOTIFY_DATA_SYNC will be triggered for that subject.

If, for example, the failover period is abnormally long, and the client is not able to retrieve, after a failover reconnection, the messages published during the failover period for one of its subscribed subjects, then the API will retrieve only the most recent message available for that subject and will trigger a NOTIFY_DATA_RESYNC status notification for that subject, the client behaving as a new client which connects to the cluster at the moment of the failover reconnection.

For a complete discussion related to load balancing, failover, and guaranteed message delivery features see the *MigratoryData Architecture Guide* (PDF, HTML).

Parameters

servers	An array of strings where each string represents the network address (IP address or DNS domain name and its corresponding port) of a MigratoryData server, optionally prefixed by a weight ranging from `0` to `100`. If the weight prefix is not provided to an address, then the API will automatically assign to that address a default weight `100`.

5.1.2.6 subscribe()

```
void MigratoryDataClient::subscribe (
            std::vector< std::string > & subjects )
```

Subscribe to one or more subjects.

Subscribe for real-time messages having as subjects the strings provided in the `subjects` parameter.

As an example, supposing messages are market data updates having as subjects stock names. Then, to subscribe for the messages having as subjects `/stocks/NYSE/IBM` and `/stocks/Nasdaq/MSFT` the following code will be used:

```
vector<string> subjects;
subjects.push_back("/stocks/NYSE/IBM");
subjects.push_back("/stocks/Nasdaq/MSFT");
client->subscribe(subjects);
```

Parameters

subjects	An array of strings representing subjects.

5.1.2.7 unsubscribe()

```
void MigratoryDataClient::unsubscribe (
            std::vector< std::string > & subjects )
```

Unsubscribe from one or more subjects.

Unsubscribe from the subscribed subjects provided in the `subjects` parameter.

Parameters

subjects	An array of strings representing subjects.

5.1.2.8 setEntitlementToken()

```
void MigratoryDataClient::setEntitlementToken (
            std::string & token )
```

Assign an authorization token to the client.

For more details about authorization see the parameter `Authorization.Type` in the *MigratoryData Configuration Guide* (PDF, HTML).

Parameters

token	A string representing an authorization token.

5.1.2.9 getSubjects()

```
void MigratoryDataClient::getSubjects (
            std::vector< std::string > & subjects )
```

Return the list of subscribed subjects.

Parameters

out	*subjects*	A vector of strings representing the subscribed subjects.

5.1.2.10 notifyBeforeReconnectRetries()

```
void MigratoryDataClient::notifyBeforeReconnectRetries (
            int n )
```

Define the number of failed attempts to connect to one or more MigratoryData servers before triggering a status notification NOTIFY_SERVER_DOWN.

Parameters

n	The number of the failed attempts to connect to one or more MigratoryData servers before triggering a status notification NOTIFY_SERVER_DOWN; default value is `1`.

5.1.2.11 notifyWhenReconnectRateExceedsThreshold()

```
void MigratoryDataClient::notifyWhenReconnectRateExceedsThreshold (
            int n )
```

Define the number of reconnect attempts to one or more MigratoryData servers per 3-minute window before triggering a status notification NOTIFY_RECONNECT_RATE_EXCEEDED.

Parameters

n	The number of reconnect attempts to one or more MigratoryData servers per 3-minute window before triggering a status notification NOTIFY_RECONNECT_RATE_EXCEEDED; default value is `15`.

5.1.2.12 setQuickReconnectInitialDelay()

```
void MigratoryDataClient::setQuickReconnectInitialDelay (
            int seconds )
```

Define the number of seconds to wait before attempting to reconnect to the cluster after a connection failure is detected.

Connection Failure Detection

Connection failure is detected immediately for almost all users running modern browsers. For a few users running modern browsers (and being subject to temporary, atypical network conditions) as well as for all users running older browsers without WebSocket support, connection failure is detected after 30-40 seconds.

Reconnection Phases and Policies

When a connection failure is detected, the API will attempt to reconnect to the servers of the MigratoryData cluster as follows: First, it will attempt to reconnect (up to a number of times as defined by MigratoryDataClient.setQuick↩ ReconnectMaxRetries()) using small delays between retries (the Quick Reconnection phase). If the connection cannot be established after the Quick Reconnection phase, then the API will attempt to reconnect less frequently according to the policy defined by MigratoryDataClient.setReconnectPolicy().

The delays between retries are computed acoording to the following algoritm where the values of the variables involved are defined by the API methods having substantially the same names:

```
Quick Reconnect Phase (retries <= quickReconnectMaxRetries)
-----------------------------------------------------------

   (retries starts with 1 and increment by 1 at each quick reconnect)

   reconnectDelay = quickReconnectInitialDelay * retries - random(0, quickReconnectInitialDelay

After Quick Reconnect Phase (retries > quickReconnectMaxRetries)
----------------------------------------------------------------

   (reset retries to start with 1 and increment by 1 at each reconnect)

   If reconnectPolicy is CONSTANT_WINDOW_BACKOFF, then

      reconnectDelay = reconnectTimeInterval

   else if reconnectPolicy is TRUNCATED_EXPONENTIAL_BACKOFF, then

      reconnectDelay = min(reconnectTimeInterval * (2 ^ retries) - random(0, reconnectTimeInter
```

For a few users running modern browsers (and being subject to temporary, atypical network conditions) as well as for all users running older browsers without WebSocket support, if `reconnectDelay` computed with the algorithm above is less than 10 seconds, then it is rounded to 10 seconds.

Parameters

seconds	The number of seconds to wait before attempting to reconnect to the cluster; default value is `5` seconds.

5.1.2.13 setQuickReconnectMaxRetries()

```
void MigratoryDataClient::setQuickReconnectMaxRetries (
            int retries )
```

Define the maximum number of retries for the Quick Reconnect phase.

Parameters

retries	The maximum number of quick reconnect retries; default value is `3`.

5.1.2.14 setReconnectPolicy()

```
void MigratoryDataClient::setReconnectPolicy (
            std::string policy )
```

Define the reconnect policy to be used after the Quick Reconnect phase.

See MigratoryDataClient.setQuickReconnectInitialDelay() to learn about the Quick Reconnect phase and the reconnect schedule for the policy defined by this method.

Parameters

policy	The reconnect policy to be used after the Quick Reconnect phase. The possible values are CONSTANT_WINDOW_BACKOFF and TRUNCATED_EXPONENTIAL_BACKOFF; the default value is TRUNCATED_EXPONENTIAL_BACKOFF.

5.1.2.15 setReconnectTimeInterval()

```
void MigratoryDataClient::setReconnectTimeInterval (
            int seconds )
```

Define the time interval used for the reconnect schedule after the Quick Reconnect phase.

See MigratoryDataClient.setQuickReconnectInitialDelay() to learn about the Quick Reconnect phase and how the value defined by this API method is used.

Parameters

seconds	A time interval expressed in seconds; default is `20` seconds.

5.1.2.16 setReconnectMaxDelay()

```
void MigratoryDataClient::setReconnectMaxDelay (
            int seconds )
```

Define the maximum reconnect delay for the TRUNCATED_EXPONENTIAL_BACKOFF policy.

See MigratoryDataClient.setQuickReconnectInitialDelay() to learn how the value defined by this API method is used.

Parameters

seconds	The maximum reconnect delay when the policy TRUNCATED_EXPONENTIAL_BACKOFF is used; default value is 360 seconds.

5.1.2.17 publish()

```
void MigratoryDataClient::publish (
            MigratoryDataMessage & message )
```

Publish a message.

If the message includes a closure data, then a status notification will be provided via MigratoryDataListener.on↩ Status() to inform whether the message publication has been successful or not.

Parameters

message	A MigratoryDataMessage message

5.1.2.18 disconnect()

```
void MigratoryDataClient::disconnect ( )
```

Disconnect from the connected MigratoryData server and dispose the resources used by the connection.

This method should be called when the connection is no longer necessary.

5.1.3 Member Data Documentation

5.1.3.1 NOTIFY_SERVER_UP

```
const std::string MigratoryDataClient::NOTIFY_SERVER_UP
```

Indicate that the client successfully connected to a MigratoryData server.

This constant indicates that the client successfully connected to one of the MigratoryData servers defined with the API method MigratoryDataClient.setServers().

5.1.3.2 NOTIFY_SERVER_DOWN

```
const std::string MigratoryDataClient::NOTIFY_SERVER_DOWN
```

Indicate that the client failed to connect to a MigratoryData server.

This constant indicates that the client failed to connect to one of the MigratoryData servers defined with the API method MigratoryDataClient.setServers().

5.1.3.3 NOTIFY_DATA_SYNC

```
const std::string MigratoryDataClient::NOTIFY_DATA_SYNC
```

After a failover reconnection, the client synchronized a subscribed subject with the latest message available for that subject, as well as with all messages published during the failover for that subject.

This constant indicates that the client successfully synchronized the subject provided in the detail information of the status notification. Also, the potential messages published for that subject during the failover period have been successfully retrieved at the moment of the reconnection.

5.1.3.4 NOTIFY_DATA_RESYNC

```
const std::string MigratoryDataClient::NOTIFY_DATA_RESYNC
```

After a failover reconnection, the client synchronized a subscribed subject with the latest message available for that subject, but not with the potential messages published during the failover, therefore behaving as a new client.

This constant indicates that the client successfully synchronized the subject provided in the detail information of the status notification. However, the client was unable to get the messages published during the failover. Therefore, it behaves like a new client which connects to the MigratoryData server at the moment of the failover reconnection.

5.1.3.5 NOTIFY_SUBSCRIBE_ALLOW

```
const std::string MigratoryDataClient::NOTIFY_SUBSCRIBE_ALLOW
```

Indicate that the client was authorized to subscribe to a subject.

This constant indicates that the client – identified with the token defined with the API method MigratoryData↩Client.setEntitlementToken() – is allowed to subscribe to the subject provided in the detail information of the status notification.

5.1.3.6 NOTIFY_SUBSCRIBE_DENY

```
const std::string MigratoryDataClient::NOTIFY_SUBSCRIBE_DENY
```

Indicate that the client was not authorized to subscribe to a subject.

This constant indicates that the client – identified with the token defined with the API method MigratoryData↩Client.setEntitlementToken() – is not allowed to subscribe to the subject provided in the detail information of the status notification.

5.1.3.7 NOTIFY_PUBLISH_DENIED

```
const std::string MigratoryDataClient::NOTIFY_PUBLISH_DENIED
```

Indicate that the client was unable to publish a message because it is not allowed by your entitlement rules.

This constant is used to indicate that the publication of the message, having the closure provided in the detail information of the status notification, has failed. The publication failed because the client – identified with the token defined with the API method MigratoryDataClient.setEntitlementToken() – is not allowed to publish on the subject of the message.

5.1.3.8 NOTIFY_PUBLISH_NO_SUBSCRIBER

```
const std::string MigratoryDataClient::NOTIFY_PUBLISH_NO_SUBSCRIBER
```

Indicate that the client was unable to publish a message because there is no client subscribed to the subject of the message.

This constant is used to indicate that the publication of the message, having the closure provided in the detail information of the status notification, has failed. The publication failed because there is no client then subscribed to the subject of the message.

Deprecated no more is use.

5.1.3.9 NOTIFY_PUBLISH_OK

```
const std::string MigratoryDataClient::NOTIFY_PUBLISH_OK
```

Indicate that the client successfully published a message.

This constant is used to indicate that the publication of the message, having the closure provided in the detail information of the status notification, has succeeded.

5.1.3.10 NOTIFY_PUBLISH_FAILED

```
const std::string MigratoryDataClient::NOTIFY_PUBLISH_FAILED
```

Indicate that the client was unable to publish a message.

This constant is used to indicate that the publication of the message, having the closure provided in the detail information of the status notification, has failed.

5.1.3.11 NOTIFY_RECONNECT_RATE_EXCEEDED

```
const std::string MigratoryDataClient::NOTIFY_RECONNECT_RATE_EXCEEDED
```

Indicate that the reconnect rate threshold per 3-minute window has been reached.

This constant is used to indicate that during the last 3 minutes, the reconnect rate threshold defined by MigratoryDataClient.notifyWhenReconnectRateExceedsThreshold() has been reached.

5.1.3.12 CONSTANT_WINDOW_BACKOFF

```
const std::string MigratoryDataClient::CONSTANT_WINDOW_BACKOFF
```

A constant used to define the reconnect policy.

See MigratoryDataClient.setQuickReconnectInitialDelay() for more details about this policy.

5.1.3.13 TRUNCATED_EXPONENTIAL_BACKOFF

```
const std::string MigratoryDataClient::TRUNCATED_EXPONENTIAL_BACKOFF
```

A constant used to define the reconnect policy.

See MigratoryDataClient.setQuickReconnectInitialDelay() for more details about this policy.

5.2 MigratoryDataListener Class Reference

Implementations of this interface can handle the real-time messages received for the subscribed subjects as well as various status notifications.

Public Member Functions

- virtual void onMessage (const MigratoryDataMessage &message)=0

  This method handles the real-time messages received from a MigratoryData server for the subscribed subjects.
- virtual void onStatus (const std::string &status, std::string &info)=0

  This method handles the status notifications.

5.2.1 Detailed Description

Implementations of this interface can handle the real-time messages received for the subscribed subjects as well as various status notifications.

Use the API method MigratoryDataClient.setListener() to register your listener implementation.

5.2.2 Member Function Documentation

5.2.2.1 onMessage()

```
virtual void MigratoryDataListener::onMessage (
            const MigratoryDataMessage & message ) [pure virtual]
```

This method handles the real-time messages received from a MigratoryData server for the subscribed subjects.

Parameters

message	An object of type MigratoryDataMessage .

5.2.2.2 onStatus()

```
virtual void MigratoryDataListener::onStatus (
            const std::string & status,
            std::string & info )  [pure virtual]
```

This method handles the status notifications.

The possible values of the `status` parameter are:

- `MigratoryDataClient.NOTIFY_SERVER_UP` indicates that the client successfully connected to the MigratoryData server provided in the detail information of the status notification
- `MigratoryDataClient.NOTIFY_SERVER_DOWN` indicates that the client was not able to connect to the MigratoryData server provided in the detail information of the status notification
- `MigratoryDataClient.NOTIFY_DATA_SYNC` indicates that, after a failover reconnection, the client successfully synchronized the subject given in the detail information of the status notification. Moreover, the client received the messages published during the failover period for this subject.
- `MigratoryDataClient.NOTIFY_DATA_RESYNC` indicates that, after a failover reconnection, the client successfully synchronized the subject given in the detail information of the status notification. However, the client have not received the potential messages published during the failover period for this subject, the client behaving like a new client which just connected to the MigratoryData server.
- `MigratoryDataClient.NOTIFY_SUBSCRIBE_ALLOW` indicates that the client – identified with the token given in the argument of `MigratoryDataClient.setEntitlementToken()` – is allowed to subscribe to the subject provided in the detail information of the status notification
- `MigratoryDataClient.NOTIFY_SUBSCRIBE_DENY` indicates that the client – identified with the token given in the argument of `MigratoryDataClient.setEntitlementToken()` – is not allowed to subscribe to the subject provided in the detail information of the status notification
- `MigratoryDataClient.NOTIFY_PUBLISH_OK` indicates that the client successfully published the message having the closure data provided in the detail information of the status notification
- `MigratoryDataClient.NOTIFY_PUBLISH_FAILED` indicates that the client was unable to publish the message having the closure data provided in the detail information of the status notification
- `MigratoryDataClient.NOTIFY_PUBLISH_DENIED` indicates that the client was unable to publish the message having the closure data provided in the detail information of the status notification because the client – identified with the token given in the argument of `MigratoryDataClient.setEntitlementToken()` – is not allowed to publish on the subject of the message
- `MigratoryDataClient.NOTIFY_PUBLISH_NO_SUBSCRIBER` indicates that the client was unable to publish the message having the closure data provided in the detail information of the status notification because there is no client subscribed to the subject of the message

Parameters

status	The type of the status notification (see the possible values above).
info	The detail information of the status notification.

5.3 MigratoryDataLogListener Class Reference

The listener interface that you should implement in order to get the logs of the API.

Public Member Functions

- virtual void onLog (std::string &log)=0

  This method handles the logs received from the API.

5.3.1 Detailed Description

The listener interface that you should implement in order to get the logs of the API.

Use the API method MigratoryDataClient.setLogListener() to register your log listener implementation.

5.3.2 Member Function Documentation

5.3.2.1 onLog()

```
virtual void MigratoryDataLogListener::onLog (
            std::string & log )  [pure virtual]
```

This method handles the logs received from the API.

Parameters

log	A string representing a log message.

5.4 MigratoryDataMessage Class Reference

Represent a message.

Public Member Functions

- MigratoryDataMessage ()

Default constructor.

- MigratoryDataMessage (const MigratoryDataMessage &message)

  Copy constructor.
- MigratoryDataMessage (const std::string &subject, const std::string &content)

  Create a MigratoryDataMessage object.
- MigratoryDataMessage (const std::string &subject, const std::string &content, const std::string &closure)

  Create a MigratoryDataMessage object.
- std::string getSubject () const

  Get the subject of the message.
- std::string getContent () const

  Get the content of the message.
- std::string getClosure () const

  Get the closure of the message.
- bool isSnapshot () const

  Test whether the message is a snapshot message or not.
- void setReplyTosubject (std::string &replyToSubject)
- std::string getReplyToSubject () const
- virtual ~MigratoryDataMessage ()

  Destructor.

Protected Attributes

- bool **snapshot**
- bool **recovery**
- int **seq**
- int **epoch**

5.4.1 Detailed Description

Represent a message.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 MigratoryDataMessage() [1/3]

```
MigratoryDataMessage::MigratoryDataMessage (
            const MigratoryDataMessage & message )
```

Copy constructor.

Parameters

message	A MigratoryDataMessage object

5.4.2.2 MigratoryDataMessage() [2/3]

```
MigratoryDataMessage::MigratoryDataMessage (
            const std::string & subject,
            const std::string & content )
```

Create a MigratoryDataMessage object.

Parameters

subject	The subject of the message
content	The content of the message

5.4.2.3 MigratoryDataMessage() [3/3]

```
MigratoryDataMessage::MigratoryDataMessage (
            const std::string & subject,
            const std::string & content,
            const std::string & closure )
```

Create a MigratoryDataMessage object.

Parameters

subject	The subject of the message
content	The content of the message
closure	The closure of the message

5.4.3 Member Function Documentation

5.4.3.1 getSubject()

```
std::string MigratoryDataMessage::getSubject ( ) const
```

Get the subject of the message.

Returns

A string representing the subject of the message

5.4.3.2 getContent()

```
std::string MigratoryDataMessage::getContent ( ) const
```

Get the content of the message.

Returns

A string representing the content of the message

5.4.3.3 getClosure()

```
std::string MigratoryDataMessage::getClosure ( ) const
```

Get the closure of the message.

Returns

The closure data of the message

5.4.3.4 isSnapshot()

```
bool MigratoryDataMessage::isSnapshot ( ) const
```

Test whether the message is a snapshot message or not.

Returns

true if the message is a snapshot message

5.4.3.5 setReplyTosubject()

```
void MigratoryDataMessage::setReplyTosubject (
            std::string & replyToSubject )
```

Set the subject to be used to reply to this message.

If a reply subject is attached to a message with this method, the message acts as a request. The clients which receive a request message will be able to reply by sending a message having as subject the reply subject.

If the reply subject is not already subscribed, it is subscribed by the API library implicitly. It can be reused for subsequent request/reply interactions (and even for receiving multiple replies to one request). When it is not needed anymore, it should be unsubscribed explicitly.

Parameters

subject	The subject to be used to reply to this message.

5.4.3.6 getReplyToSubject()

```
std::string MigratoryDataMessage::getReplyToSubject ( ) const
```

Get the subject to be used to reply to this message.

A client which receives a message containing a reply subject should interpret the message as a request. It has the option to use the reply subject - extracted from the message with this method - to send a reply.

Returns

The subject to be used to reply to this message.

Chapter 6

File Documentation

6.1 src/MigratoryDataClient.h File Reference

Include the declaration of the MigratoryDataClient class.

Classes

- class MigratoryDataClient

  This class implements all the necessary operations for connecting to a cluster of one or more MigratoryData servers, subscribing to subjects, getting real-time messages for the subscribed subjects, and publishing real-time messages.

6.1.1 Detailed Description

Include the declaration of the MigratoryDataClient class.

6.2 src/MigratoryDataListener.h File Reference

Include the declaration of the MigratoryDataListener class.

Classes

- class MigratoryDataListener

  Implementations of this interface can handle the real-time messages received for the subscribed subjects as well as various status notifications.

6.2.1 Detailed Description

Include the declaration of the MigratoryDataListener class.

6.3 src/MigratoryDataLogLevel.h File Reference

Enumerate the MigratoryData logging levels.

Enumerations

- enum MigratoryDataLogLevel { LOG_ERROR, LOG_INFO, LOG_DEBUG, LOG_TRACE }
  This class enumerates the MigratoryData logging levels.

6.3.1 Detailed Description

Enumerate the MigratoryData logging levels.

6.3.2 Enumeration Type Documentation

6.3.2.1 MigratoryDataLogLevel

`enum MigratoryDataLogLevel`

This class enumerates the MigratoryData logging levels.

The available log levels ordered by verbosity are:

- LOG_ERROR (less verbose)
- LOG_INFO
- LOG_DEBUG
- LOG_TRACE (most verbose)

Enumerator

LOG_ERROR	The `LOG_ERROR` level turns on the error logs of the API.
LOG_INFO	The `LOG_INFO` level turns on the info, warning, and error logs of the API.
LOG_DEBUG	The `LOG_DEBUG` level turns on the debug, info, warning, and error logs of the API.
LOG_TRACE	The `LOG_TRACE` level turns on all the logs of the API.

6.4 src/MigratoryDataLogListener.h File Reference

Include the declaration of the MigratoryDataLogListener class.

Classes

- class MigratoryDataLogListener

  The listener interface that you should implement in order to get the logs of the API.

6.4.1 Detailed Description

Include the declaration of the MigratoryDataLogListener class.

6.5 src/MigratoryDataMessage.h File Reference

Include the declaration of the MigratoryDataMessage class.

Classes

- class MigratoryDataMessage

  Represent a message.

6.5.1 Detailed Description

Include the declaration of the MigratoryDataMessage class.

Index