FioranoMQ® 9

Native CSharp RTL Sample Applications Guide

Contents

Native CSharp-Runtime Examples

Sample programs illustrating the use of Native CSharp-RTL for PubSub and PTP operations

- PTP
  - AdminAPI
  - AdvisoryMessage
  - DeadMessageQueue
  - DurableConnections
    - Basic
    - Serverlessmode
  - LMS
  - MessageSelector
  - MultiPartMessage
  - ObjectMessage
  - PriorityQueues
  - QueueBrowser
  - RequestReply
    - TimedRequestReply
  - RevalidateConnections
  - SendReceive
  - Transactions
- PubSub
  - AdminAPI
  - AdvisoryMessage
  - DurableConnections
    - Basic
    - Serverlessmode
  - DurableSubscriber
  - LMS
  - MessageBrowser
  - MessageSelector
  - ObjectMessage
  - PubSub
  - RequestReply
    - TimedRequestReply

- RevalidateConnections
- Transactions

These samples are available in %FMQ_DIR%\clients\nativecsharp\samples directory.

Compiling the Sample Programs:

To compile the sample programs shipped with FioranoMQ installer, run the script build_samples.bat which is present under %FMQ_DIR%\clients\nativecsharp\scripts. To run the script file make sure %DOT_NET_FRAMEWORK_HOME% should be set which should point to csc.exe and %DOT_NET_HOME% which should point to the DOT NET installation DIR.

Note: Use [DOS 8.3] Naming format for setting DOT_NET_FRAMEWORK_HOME and DOT_NET_HOME.

Running the Sample Programs:

To run any of the sample program please make sure that the dll “fmq-csharp-native.dll”, which is present under %FMQ_DIR%\clients\nativecsharp\assembly, should be present locally or should be present in the Global Assembly Cache.

PTP Samples

AdminAPI

Following example illustrates the use of FioranoMQ Administration APIs for creation of JMS Administered objects such as Destinations, ConnectionFactories and Users.

- **AdminTest.cs** - The program "AdminTest" uses Administrator APIs to log in as administrator with a password and create and lookup Queue, ConnectionFactories and Users using the appropriate API calls.

The administrator uses the password "passwd" to login as the administrator. If you have previously set the administrator password to something other than "passwd", then this test case will fail.

To run these samples using FioranoMQ, do the following:

1. Compile the source file. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the CSharp program.

This program creates two Queues namely - Queuetest1 and Queuetest2; and two ConnectionFactories viz. myconnectionFactory1 and myconnectionFactory2.

AdvisoryMessages

FioranoMQ provides advisory messages to inform its client applications about the situations that can affect them. The advisory messages provide errors and informational messages.

Following examples illustrates the use of Advisory Messages in PTP messaging model using the FioranoMQ Native CSharp Runtime Library.

- **QSender.cs** The program implements an advisory message listener which listens to all type of advisory messages. It provides one possible handling of various advisory messages. Also it reads strings from standard input and sends them on the Queue "primaryQueue" in persistent mode
- **QReceiver.cs** - The program implements an advisory message listener which listens for all type of advisory messages. It provides one possible handling of various advisory messages. Also It implements a synchronous listener to listen for messages on queue named "primaryQueue"

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the Receiver by executing the Receiver.exe executable file.
3. Run the Sender by executing the Sender.exe file.

DeadMessagesQueue

FioranoMQ has support for handling expired messages on the FioranoMQ server for the PTP sub system of JMS. The Dead Message Queue (DMQ) is created on the server to handle expired messages, which were earlier deleted from the FioranoMQ server.

These programs illustrate the basic functionality of FioranoDMQ using the FioranoMQ Native CSharp Runtime Library. This directory contains the following three programs.

- **Sender.cs -** It implements a client application sending some data on "primaryQueue". Few messages sent have infinite time to live, while remaining have very small TTL.
- **Receiver.cs** - It receives messages synchronously from "primaryQueue".
- **ReceiverDMQ.cs** - It receives messages synchronously from dead message queue.

To run these samples using FioranoMQ, do the following:

1. Before running these samples, make sure that you have set the properties named Fiorano->mq->ptp->Queues->PrimaryQueue->enableDMQ, fiorano->mq->ptp->queues->primaryQueue->isDbCleanupEnabled & fiorano->mq->ptp->queuingSubSystem->EnableNotificationOnDeadMessage to true.
2. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
3. Run the ReceiverDMQ by executing ReceiverDMQ.exe executable file.
4. Run the Sender by executing the Sender.exe executable file.
5. Run the receiver by executing the Receiver.exe file.

Receiver will receive messages that had infinite time to live and remaining messages will be received by ReceiverDMQ.

DurableConnections

FioranoMQ introduces the concept of a Durable Connection. A Durable Connection maintains connectivity with the FioranoMQ server at all times. The applications do not have to take care of storing, re-connecting and then forwarding the stored messages to the server. This frees up the application from the complex task of building the store and forward mechanism at its end. In addition, it improves the reliability of the underlying JMS transport.

DurableConnections -Basic

The examples illustrate the behaviour of PTP Sender/Receiver clients when durable connections are enabled on the server using the FioranoMQ Native CSharp Runtime Library.

- **Sender.cs** - Reads strings from standard input and sends them to the Queue "primaryQueue". When the FioranoMQ Server is down, the sender stores the messages in client side cache in the directory specified in the env.put (FioranoJNDIContext.DURABLE_CONNECTIONS_BASE_DIR, ."+File.separator+"myCache").

  This sample revalidates connections when the server comes up and then all the messages stored in the client side cache are sent to the server. Revalidate, re-establishes connection with FioranoMQ Server and restores the validity of created sessions, senders and receivers on this connection. Optional arguments can be provided for transacted sessions, and the use of sendPending- Messages API where you can first check for sending and previously pending messages for a particular connection when the connection comes up for the first time.

- **Receiver.cs** - Implements a synchronous listener to listen for messages sent to the queue - "primaryQueue". This sample revalidates connections with the FioranoMQ Server when the connection is lost with FioranoMQ Server. Revalidate, reestablishes connection with FioranoMQ Server and also restores the validity of created sessions, senders and receivers on this connection.

- **CSPQueueBrowser.cs -** Browses messages stored in the local client side cache. When the connection with the server is down, all the messages are stored in client side cache. The browser provides APIs to browse messages stored in the local client side cache. Options can be provided in the API CSPBrowser. browseMessagesOnQueue(); to browse messages for a particular queue in transacted sessions, non-transacted sessions or all messages.

Note: Before running these samples, make sure that you have set the properties named AllowDurableConnections & EnableAutoRevalidation to true

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the Receiver by executing Receiver.exe executable file.
3. Run the Sender by executing the Sender.exe executable file.
4. Send some messages (Receiver will be receiving these messages).
5. Shut down FioranoMQ server.

6. Send some more messages (at this point receiver will not receive any messages).
7. Run the CSPQueueBrowser by executing CSPQueueBrowser.exe executable file to browse the messages in local client side cache.
8. Restart FioranoMQ server.
9. The sender/receiver should now revalidate connection and all client side cached messages will be sent to the receiver. (The receiver should now receive all messages). You can run the CSPQueueBrowser again to check whether any messages are left in client side cache. (There should not be any messages left now).

DurableConnections – Serverlessmode

The examples illustrate the behaviour of PTP Sender/Receiver clients in Serverless mode when durable connections are enabled on the server using the FioranoMQ Native CSharp Runtime Library. These samples illustrate the support provided in FioranoMQ to send and receive messages even in case when the MQ Server is not up and running. Please stop the MQ Server before running these samples.

- **Sender.cs** - Reads strings from standard input and publishes them on the Queue "primaryQueue". If the server has not started, the sender stores the messages in client side cache in the default directory, ".\cspCache". The directory used to store message in controlled by the environment variable FioranoJNDIContext.DURABLE_CONNECTIONS_BASE_DIR. When the server comes up, all the messages are sent to it.
- **Receiver.cs** - Implements an asynchronous listener to listen for messages published on the Queue - "primaryQueue". In case, server has not started, this receiver doesnot receive any message. But when server comesup, it receives all the messages published on primaryQueue.

Note: Before running these samples, make sure that you have set the properties named AllowDurableConnections to true

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the Receiver by executing Receiver.exe executable file.
3. Run the Sender by executing the Sender.exe executable file.
4. Send some messages (Receiver will not be receiving these messages).
5. Start FioranoMQ server (At this time all the published messages will be sent to FioranoMQ server transparently).
6. Receiver will receive all the published messages.

LMS

With the introduction of Large Message Support (LMS) in FioranoMQ, clients can now transfer large messages in the form of large files with theoretically no limit on the message size. Large messages can be attached with any JMS message and the client can be sure of a reliable and secure transfer of the message through FioranoMQ server.

Following are the highlights of the FioranoMQ LMS implementation

- Reliable transfer of large messages.
- No increase in cache/JVM heap size required.
- The Large message transfer is not restricted to any queue or topic.
- Ability to resume message transfer at both sender and receiver end.
- Minimal changes in the application code.

This folder contains samples that demonstrates the use of sending large messages (files) using Fiorano JMS server using the FioranoMQ Native CSharp Runtime Library.

- **LMSender.cs** - This program illustrates how an application can send a large message in the point- to- point messaging model. This program performs the following steps.
  1. It creates a JMS connection with the FioranoMQ server and send a large message (JMS message with a property which specifies the path of the large file) on primaryQueue.
  2. The published large message notifies the consumers listening on primary-Queue about the availability of a file called send.zip that can be received.
  3. When a consumer responds to the notification, then both the parties (sender and consumer) performs the initial handshaking operation.
  4. When the hand-shake is complete, the sender creates a status listener and registers it on the large message so that it can be notified about the transfer status asynchronously.
  5. Now sender starts sending the various fragments of the file, send.zip, to the consumer and starts receiving notification when a complete window of fragments is transferred.
  6. When the message (file) transfer is complete, the application is notified about the completion of the message transfer. In case of a failure in transferring the message, the program receives the notification through the same status listener.
- **LMReceiver.cs** - This program illustrates how an application can receive and store the large message in the point-to-point messaging model. This program performs the following steps.
  1. It creates a JMS connection with the FioranoMQ server and starts listening for availability of messages on primaryQueue.
  2. On receiving the availability notification, it creates a status listener so that it can receive notifications about the message transfer status asynchronously.

3. Now the program specifies the complete path of the location (.\received.zip) on which the received fragments (of the sent file) should be saved.
4. The consumer receives a transfer status notification when it receives a complete window successfully.
5. Also when the message (file) transfer is complete, then the consumer is notified about the completion and in case of a failure in receiving the message, the program receives the exception through the same status listener.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the LMSender by executing LMSender.exe executable file.
3. Run the LMReceiver by executing LMReceiver.exe file.

Note: You would notice that when the file, send.zip is successfully transferred and received by the consumer, then a new file called received.zip will get created in the same directory.

MessageSelectors

A message selector allows a client to specify, by message header, the messages it's interested in. Only messages whose headers and properties match the selector are delivered. The semantics of not delivered differ a bit depending on the MessageConsumer being used. Message selectors cannot reference message body values.

Consumer's message selectors cannot be modified dynamically. However it is not necessary to close the connection or reconnect for this purpose. It is sufficient if the consumer is closed (subscriber.close() API) and a new consumer is created with the new selector.

This directory contains two sample programs which illustrate the use of Message Selector using the FioranoMQ Native CSharp Runtime Library.

- **SelectorReceive.cs** - This example illustrates the use of Message Selectors. The receiver receives Messages that match the criteria specified in the getSelectedMessages() method.
- **SelectorSend.cs** - SelectorSend publishes messages, which contain details such as make and model about a particular car as JMS Message properties.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the SelectorRecv by executing the SelectorRecv.exe executable file.
3. Run the SelectorSend by executing the SelectorSend.exe file.

   SelectorSend.exe –filename [Fully qualified name of the file]

Note: Run the "SelectorRecv" program first, and then run the "SelectorSend" program. When the SelectorSend program is run, the data file ch05.dat must be supplied as input.

MultiPartMessage

This directory contains two sample programs which illustrate the use of Multi Part Message using the FioranoMQ Native CSharp Runtime Library.

- **MultiPartMessage.cs**- This sample provides a definition of the multipart message and provides APIs to add a new part and to retrieve the already added parts from the message.
- **SampleClass.cs -** This sample provides a stepped down version of a object part that can be added to the multipart message.
- **MultiPartMessageSender.cs -** This sample demonstrates how an application can create a multipart message, adds different parts to it and finally sends it on a JMS destination using the JMS semantics completely.
- **MultiPartMessageReceiver.cs**- This sample demonstrates how an application can receive a multipart message and then retrieve different parts that were added by the sender sample.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the MultiPartMessageSender by executing the MultiPartMessageSender.exe executable file.
3. Run the MultiPartMessageReceiver by executing the MultiPartMessageReceiver.exe file.

ObjectMessage

Any serializable Java object can be packaged into an Object message and published on a Topic or Queue. If a collection of Java objects needs to be published, one of the collection classes provided in Java 2 can be used.

This directory contains a program which demonstrates how to send Object messages using JMS in a PTP messaging model using the FioranoMQ Native CSharp Runtime Library.

- **ObjectSender.cs** - Creates an Object Message and send it on a queue. By default the queue on which message is published is primaryQueue, which can be modified using the command line argument.
- **ObjectReceiver.cs** - Implements a synchronous listener which listens for object messages published on a queue. The default queue on which listener is made is primaryQueue, which can be modified using the command line arguments.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the Receiver by executing the Receiver.exe executable file.
3. Start up one or more named clients using the command(s) by executing the ObjectSender.exe file.

PriorityQueues

Priority queues refer to sending messages with priorities on Queues.

- **PriorityQueue.cs** - This program demonstrates the process to send messages with priorities on Queues.

The administrator uses the password "passwd" to login as the administrator. If you have previously set the administrator password to something other than "passwd", then this test case will fail.

To run these samples using FioranoMQ, do the following:

1. Compile the source file. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the CSharp program.

Browser

This directory contains two sample programs which illustrate basic JMS Browser functionality using the FioranoMQ Native CSharp Runtime Library.

- **Q_Sender.cs** - Reads strings from standard input and sends them on the queue "primaryQueue".
- **QBrowser.cs** - Implements a browser, which is used to browse the messages on the queue "primaryQueue", and prints out the received messages.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program
2. Start the Sender program. When the program comes up, type in a few strings, pressing the Enter key after each string.
3. Start the QueueBrowser program. The Queue Browser will enumerate all the messages sent.
4. Repeat steps (2) and (3) over. Each time, you will find that all messages published during are immediately browsed by the QueueBrowser.

RequestReply

JMS provides the JMSReplyTo message header field for specifying the Destination where a reply to a message should be sent. The JMSCorrelationID header field of the reply can be used to reference the original request. In addition, JMS provides a facility for creating temporary queues and topics that can be used as a unique destination for replies.

Enterprise messaging products support many styles of request/reply, from the simple “one message request yields a one message reply” to “one message request yields streams of messages from multiple respondents.” Rather than architect a specific JMS request/reply abstraction, JMS provides the basic facilities on which many can be built.

For convenience, JMS defines request/reply helper classes (classes written using JMS) for both the PTP and Pub/Sub domains that implement a basic form of request/reply. JMS providers and clients may provide more specialized implementations

Basic

This directory contains the examples illustrating use of the built-in JMS request/reply API (the request() and replyTo() APIs) using the FioranoMQ Native CSharp Runtime Library.

- **Requestor.cs** - Implements a client application making requests.
- **Replier.cs** Implements a server application that listens for requests on a given Queue and replies to each request.

FioranoMQ provides a wrapper over the JMS specific Topic and QueueRequestors, namely, FioranoTopicRequestor and FioranoQueueRequestors. These wrappers provide additional APIs to set timeout for requests.

FioranoTopic and FioranoQueue Requestor, now make it mandatory for the replier to set the "correlation-ID" received from the requested message. FioranoRequestor sends data with a "correlation-ID" set, the replier has to send back a reply with the "correlation-ID" set by the requestor.

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the Requestor by executing the Requestor.exe executable file.
3. Run the Replier by executing the Replier.exe file.

Timeout

This directory contains two sample programs which illustrate the timed request/ reply abstraction provided by FioranoMQ.

- **TimedQueueRequestor.cs** - Reads strings from standard input and use it to send a request on the queue "primaryqueue". It waits for the reply, for a specific time interval.

- **TimedQueueReplier.cs** - Implements an asynchronous listener, which listens on the queue "primaryqueue", and replies to each request.

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the replier by executing the TimedQueueReplier.exe executable file.
3. Run the timed requestor by executing the TimedQueueRequestor.exe file.

RevalidateConnections

FioranoMQ server on detecting a loss of connection, notifies the application by invoking the onException ( ) of ExceptionListener object, which is associated with the Connection instance. The loss of connection might be due to number of possible reasons, such as termination of server process and network failure. When working over an unreliable network, the FioranoMQ administrator must enable pinger in the server.

On detecting a failure, an application automatically reconnects through a proprietary API - revalidate ( ), provided by FioranoMQ. In addition, when the application automatically reconnects, its does not lose its state, active Senders and Receivers. This API can be found in FioranoQueueConnection for PTP.

This directory contains two sample programs which illustrate revalidate connection property using the FioranoMQ Native CSharp Runtime Library. These samples revalidate connections when the connection is lost with the FioranoMQ Server. In addition, it restores the validity of created sessions, senders and receivers on this connection.

- **Sender.cs** - Reads strings from standard input and sends them on the queue "primaryQueue".
- **Receiver.cs** - Implements an asynchronous listener, which listens on the queue "primaryQueue", and prints out the received messages.

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the receiver by executing the receiver.exe executable file.
3. Run the sender by executing the Sender.exe file.
4. Shut down FioranoMQ server
5. Restart FioranoMQ server.
6. The Sender and Receiver will re-establish the connection with the server after sometime and messages will start getting exchanged again.

SendReceive

This directory contains four sample programs which illustrate basic JMS Send/ Receive functionality using the FioranoMQ Native CSharp Runtime Library.

- **QSender.cs** - Reads strings from standard input and sends them on the queue "primaryqueue".
- **QReceiver.cs** - Implements an asynchronous listener, which listens on the queue "primaryqueue", and prints out the received messages.
- **NPQSender.cs** - Reads strings from standard input and sends them on the Queue "primaryQueue" in non-persistent mode.
- **ListenerReceiver.cs -** Implements an listener, which listens on the queue "primaryqueue", and prints out the received messages

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the Receiver by executing the Receiver.exe executable file in case you want to run asynchronous receiver otherwise run ListenerReceiver.exe to receive message synchronously.
3. Run the Sender by executing the Sender.exe file in case you want to send Persistent Messages otherwise run NPQSender.exe which will send Non-Persistent Messages.

Note: To run any of the Native CSharp samples, ensure that environment variable FMQ_DIR points to Fiorano installation directory (this defaults to \Fiorano\FioranoMQ\fmq).

Transactions

A Session may optionally be specified as transacted. Each transacted session supports a single series of transactions. Each transaction groups a set of produced messages and a set of consumed messages into an atomic unit of work. In effect, transactions organize a session's input message stream and output message stream into series of atomic units. When a transaction commits, its atomic unit of input is acknowledged and its associated atomic unit of output is sent. If a transaction rollback is done, its produced messages are destroyed and its consumed messages are automatically recovered.

This directory contains a sample programs which illustrate JMS Transaction functionality using the FioranoMQ Native CSharp Runtime Library.

- **Transactions.cs -** Implements the sender and receiver, and uses the commit/rollback functionality to demonstrate JMS Transactions

To run this sample using FioranoMQ, do the following:

1. Compile the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.

2. Run the sample by executing the Transactions.exe executable file. For proper results from the sample, ensure that there are no messages in the primary-Queue.

PubSub samples

AdminAPI

Following example illustrates the use of FioranoMQ Administration APIs for creation of JMS Administered objects such as Destinations, ConnectionFactories and Users.

- **AdminTest.cs** - The program "AdminTest" uses Administrator APIs to log in as administrator with a password and create and lookup Topics, ConnectionFactories and Users using the appropriate API calls.

The administrator uses the password "passwd" to login as the administrator. If you have previously set the administrator password to something other than "passwd", then this test case will fail.

To run these samples using FioranoMQ, do the following:

1. Compile the source file. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the CSharp program.

This program creates two Topics namely - MyTopictest1 and MyTopictest2;and two ConnectionFactories viz. MyTconnectionFactory1 and MyTconnectionFactory2.

AdvisoryMessages

FioranoMQ provides advisory messages to inform its client applications about the situations that can affect them. The advisory messages provide errors and informational messages.

Following examples illustrates the use of Advisory Messages in PTP messaging model using the FioranoMQ Native CSharp Runtime Library.

- **Publisher.cs** The program implements an advisory message listener which listens to all type of advisory messages. It provides one possible handling of various advisory messages. Also it reads strings from standard input and sends them on the topic "primaryTopic" in persistent mode.
- **Subscriber.cs** - The program implements an advisory message listener which listens for all type of advisory messages. It provides one possible handling of various advisory messages. Also it implements a synchronous listener to listen for messages on topic named "primaryTopic".

Note: To see the CONNECTION_STATE_CHANGED advisory message being printed on the colsole of clients, stop the server after running the clients. Before that make sure durable connection is enabled on the server (i.e. in MQConfigLoader paramaters AllowDurableConnections & EnableAutoRevalidation are true)

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the Subscriber by executing the Subscriber.exe executable file.
3. Run the Publisher by executing the Publisher.exe file.

DurableConnections

FioranoMQ introduces the concept of a Durable Connection. A Durable Connection maintains connectivity with the FioranoMQ server at all times. The applications do not have to take care of storing, re-connecting and then forwarding the stored messages to the server. This frees up the application from the complex task of building the store and forward mechanism at its end. In addition, it improves the reliability of the underlying JMS transport.

DurableConnections -Basic

The examples illustrate the behaviour of PubSub Publisher/Subscriber when durable connections are enabled on the server using the FioranoMQ Native CSharp Runtime Library.

- **Publisher.cs** - Reads strings from standard input and sends them to the Topic "primaryTopic". When the FioranoMQ Server is down, the sender stores the messages in client side cache in the directory specified in the env.put (FioranoJNDIContext.DURABLE_CONNECTIONS_BASE_DIR, ."d:\\myCache").

  If the server, specified in backup url of connection factory, is up all the messages stored in the client side cache are sent to this server. Otherwise when the server comes up again all the messages are sent to it. Revalidate, re-establishes connection with FioranoMQ Server and also restores the validity of created sessions, senders and receivers on this connection. Optional arguments can be provided for transacted sessions.
- **Subscriber.cs** - Implements a asynchronous listener to listen for messages published on the Topic - "primaryTopic". When the server goes down the receiver blocks and either wait for backup server to be connected or for original server to be up again. Revalidate, re-establishes connection with FioranoMQ Server and also restores the validity of created sessions, senders and receivers on this connection.
- **CSPTopicBrowser.cs -** Browses messages stored in the local client side cache. When the connection with the server is down, all the messages are stored in client side cache. The browser provides APIs to browse messages stored in the local client side cache. Options can be provided in the API CSPBrowser. browseMessagesOnTopic(); to browse messages for a particular topic in transacted sessions, non-transacted sessions or all messages.

Note: Before running these samples, make sure that you have set the properties named AllowDurableConnections & EnableAutoRevalidation to true

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.

2. Run the Subscriber by executing Subscriber.exe executable file.
3. Run the Publisher by executing the Publisher.exe executable file.
4. Send some messages (Subscriber will be receiving these messages).
5. Shut down FioranoMQ server.
6. Send some more messages (at this point Subscriber will not receive any messages).
7. Run the CSPTopicBrowser by executing CSPTopicBrowser.exe executable file to browse the messages in local client side cache.
8. Restart FioranoMQ server.
9. The Publisher/Subscriber should now revalidate connection and all client side cached messages will be sent to the Subscriber. (The Subscriber should now receive all messages). You can run the CSPTopicBrowser again to check whether any messages are left in client side cache. (There should not be any messages left now)..

DurableConnections – Serverlessmode

The examples illustrate the behaviour of PubSub Publisher/Subscriber in Serverless mode when durable connections are enabled on the server using the FioranoMQ Native CSharp Runtime Library. These samples illustrate the support provided in FioranoMQ to send and receive messages even in case when the MQ Server is not up and running. Please stop the MQ Server before running these samples.

- **Publisher.cs** - Reads strings from standard input and publishes them on the Topic "primaryTopic". If the server has not started, the sender stores the messages in client side cache in the default directory, ".\cspCache". The directory used to store message in controlled by the environment variable FioranoJNDIContext.DURABLE_CONNECTIONS_BASE_DIR. When the server comes up, all the messages are sent to it.
- **Subscriber.cs** - Implements an asynchronous listener to listen for messages published on the Topic - "primaryTopic". In case, server has not started, this subscriber does not receive any message. But when server comes up, it receives all the messages published on primaryTopic.

Note: Before running these samples, make sure that you have set the properties named AllowDurableConnections to true

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the Subscriber by executing Subscriber.exe executable file.
3. Run the Publisher by executing the Publisher.exe executable file.
4. Send some messages (Subscriber will not be receiving these messages).
5. Start FioranoMQ server (At this time all the published messages will be sent to FioranoMQ server transparently).

6. Subscriber will receive all the published messages.

DurableSubscribers

A subscriber can be either Durable or Non-Durable. If a topic T has one or more registered durable subscribers, the JMS server retains all the messages published on that topic in an off-line database, for delivery to those subscribers that might currently be disconnected from the server. Messages are only deleted from the off-line storage once they are delivered to all the registered durable subscribers.

A non-durable subscriber only receives “on-line” messages that are published to the JMS Server while the subscriber is connected. The JMS server does not store messages for delivery to non-durable subscribers. Thus, non-durable subscribers will lose messages that are published to the server while the subscriber is disconnected.

This directory contains two sample programs which illustrate basic JMS Durable-Subscriber functionality using the FioranoMQ Native CSharp Runtime Library.

- **Publisher_d.cs** - Reads strings from standard input and publishes PERSISTENT messages on the topic "primarytopic".
- **DurableSubscriber.cs** - Implements a durable subscriber using the client ID "DS_Client_1", and durable subscriber name "Sample_Durable_Subscriber", listening on the topic "primarytopic".

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Start the DurableSubscriber program first, so that the subscriber can register with the FioranoMQ Server.
3. Next, start the Publisher_d program. When the program comes up, type in a few strings, pressing the Enter key after each string. The string is published and is received by the Durable Subscriber started in step (a) above.
4. Now, shut down the Durable Subscriber, but keep typing in messages into the Publisher program. These messages are automatically stored by the FioranoMQ Server, since a Durable Subscriber was previously registered on the topic to which the messages are being published.
5. After a while, restart the DurableSubscriber program. On restart, you will find that all messages that were published during the time that the durable subscriber was down are now made available to the subscriber.
6. Repeat steps 4 and 5 over. Each time, you will find that all messages published during the time that the Subscriber is down are immediately made available to the Subscriber when it restarts.

LMS

With the introduction of Large Message Support (LMS) in FioranoMQ, clients can now transfer large messages in the form of large files with theoretically no limit on the message size. Large messages can be attached with any JMS message and the client can be sure of a reliable and secure transfer of the message through FioranoMQ server.

Following are the highlights of the FioranoMQ LMS implementation

- Reliable transfer of large messages.
- No increase in cache/JVM heap size required.
- The Large message transfer is not restricted to any queue or topic.
- Ability to resume message transfer at both sender and receiver end.
- Minimal changes in the application code.

This folder contains samples that demonstrates the use of sending large messages (files) using Fiorano JMS server using the FioranoMQ Native CSharp Runtime Library.

- **LMPublisher.cs** - This program illustrates how an application can send a large message in the publish- subscribe messaging model. This program performs the following steps.
  1. It creates a JMS connection with the FioranoMQ server and send a large message (JMS message with a property which specifies the path of the large file) on primaryQueue.
  2. The published large message notifies the consumers listening on primary-Queue about the availability of a file called send.zip that can be received.
  3. When a consumer responds to the notification, then both the parties (sender and consumer) performs the initial handshaking operation.
  4. When the hand-shake is complete, the sender creates a status listener and registers it on the large message so that it can be notified about the transfer status asynchronously.
  5. Now sender starts sending the various fragments of the file, send.zip, to the consumer and starts receiving notification when a complete window of fragments is transferred.
  6. When the message (file) transfer is complete, the application is notified about the completion of the message transfer. In case of a failure in transferring the message, the program receives the notification through the same status listener.
- **LMSubscriber.cs** - This program illustrates how an application can receive and store the large message in the point-to-point messaging model. This program performs the following steps.
  1. It creates a JMS connection with the FioranoMQ server and starts listening for availability of messages on primaryTopic.
  2. On receiving the availability notification, it creates a status listener so that it can receive notifications about the message transfer status asynchronously.

3. Now the program specifies the complete path of the location (.\received.zip) on which the received fragments (of the sent file) should be saved.
4. The consumer receives a transfer status notification when it receives a complete window successfully.
5. Also when the message (file) transfer is complete, then the consumer is notified about the completion and in case of a failure in receiving the message, the program receives the exception through the same status listener.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the LMSender by executing LMSender.exe executable file.
3. Run the LMReceiver by executing LMReceiver.exe file.

Note: You would notice that when the file, send.zip is successfully transferred and received by the consumer, then a new file called received.zip will get created in the same directory.

MessageBrowser

This directory contains a MessageBrowser along with a Publisher and a Subscriber that illustrates the message browser functionality using the FioranoMQ Native CSharp Runtime Library.

- **MessageBrowser.cs** - An application that creates a message browser on a session using a clientID and a subscriberID which denote a durable subscriber. The ids are passed as command line arguments.
- **DurablePublisher.cs** - An application that publishes persistent messages on “primarytopic"
- **DurableSubscriber.cs -** Implements a durable subscriber using the client ID "DS_Client_1", and durable subscriber name "Sample_Durable_Subscriber", listening on the topic "primarytopic".

The best way to view message browser functionality is as follows:

1. Start the DurableSubscriber program, so that the subscriber can register with FioranoMQ Server.
2. Start the DurablePublisher program. When the program is displayed, type in a few strings, pressing the Enter key after each string. The string is published and is received by the Durable Subscriber started in step (1) above.
3. Shut down the Durable Subscriber, but keep typing in messages into the Publisher program. These messages are automatically stored by the FioranoMQ Server, since a Durable Subscriber was previously registered on the topic to which the messages are being published.
4. Start the MessageBrowser program passing the clientID and the SubscriberID of the Durable subscriber as command line arguments. The Message Browser will enumerate all the messages Published after the Durable Subscriber was shutdown.

5. Restart the durable subscriber. On restart, you will find that all messages, which were published during the time durable subscriber was inactive, are now made available to the subscriber.
6. Restart the MessageBrowser. Now the MessageBrowser will not enumerate any messages, since all of them have been received by the DurableSubscriber.
7. Repeat steps (4) and (5). Each time, you will find that all messages published during the time the Subscriber is inactive are immediately made browsed by the MessageBrowser.

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script scalled csclientbuild.bat which compiles the Native CSharp program.
2. Run the DurableSubscriber by executing DurableSubscriber.exe executable file.
3. Run the DurablePublisher by executing DurablePublisher.exe executable file.
4. Send a few messages. (Receiver will be receiving these messages).
5. Kill the durable subscriber.
6. Run the Message Browser using the command

   MessageBrowser.exe DS_Client_1 Sample_Durable_Subscriber

Note: To run any of the Native CSharp samples, ensure that environment variable FMQ_DIR points to Fiorano's installation directory (default is \Fiorano\FioranoMQ).

MsgSel

A message selector allows a client to specify, by message header, the messages it's interested in. Only messages whose headers and properties match the selector are delivered. The semantics of not delivered differ a bit depending on the MessageConsumer being used.Message selectors cannot reference message body values.

This directory contains two sample programs which illustrate the use of message selectors using the FioranoMQ Native CSharp Runtime Library.

- **SelectorPublisher.cs** – SelectorPublisher publishes messages which contain details such as make, model etc. about a particular car as JMS Message properties.
- **SelectorSubscriber.cs** - Implements an asynchronous listener, which listens on the topic "primarytopic" for the messages which match the criteria specified in the getSelectedMessages() method.

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script scalled csclientbuild.bat which compiles the Native CSharp program.
2. Run the SelectorSubscriber by executing the SelectorSubscriber.exe executable file.

3. Run the SelectorPublisher by executing the SelectorPublisher.exe file using the following command.

   SelectorPublisher -filename

Note: Run the "SelectorSubscriber" program first, and then run the "SelectorPublisher" program. When the SelectorPublisher program is run, the data file ch05.dat must be supplied as input.

ObjectMessage

Any serializable Java object can be packaged into an Object message and published on a Topic or Queue. If a collection of Java objects needs to be published, one of the collection classes provided in Java 2 can be used.

This directory contains a program which demonstrates how to send Object messages using JMS in a PubSub messaging model using the FioranoMQ Native CSharp Runtime Library.

- **ObjectPublisher.cs** - Creates an Object Message and send it on a topic. By default the topic on which message is published is primaryTopic, which can be modified using the command line argument.
- **ObjectSubscriber.cs** - Implements a synchronous listener which listens for object messages published on a topic. The default topic on which listener is made is primaryTopic, which can be modified using the command line arguments.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the Subscriber by executing the Subscriber.exe executable file.
3. Start up one or more named clients using the command(s) by executing the ObjectPublisher.exe file.

PubSub

This directory contains two sample programs which illustrate basic JMS Publisher/ Subscriber functionality using the FioranoMQ Native CSharp Runtime Library.

- **Publisher.cs** - Reads strings from standard input and sends them on the topic "primarytopic".
- **Subscriber.cs** - Implements an asynchronous listener, which listens on the topic "primarytopic", and prints out the received messages.

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the Publisher by executing the Publisher.exe executable file.
3. Run the asynchronous subscriber by executing the Subscriber.exe file.

Note: To run any of the Native CSharp samples, ensure that environment variable FMQ_DIR points to Fiorano installation directory (default is \Fiorano\FioranoMQ).

RequestReply

JMS provides the JMSReplyTo message header field for specifying the Destination where a reply to a message should be sent. The JMSCorrelationID header field of the reply can be used to reference the original request.

In addition, JMS provides a facility for creating temporary queues and topics that can be used as a unique destination for replies.

Enterprise messaging products support many styles of request/reply, from the simple “one message request yields a one message reply” to “one message request yields streams of messages from multiple respondents.” Rather than architect a specific JMS request/reply abstraction, JMS provides the basic facilities on which many can be built.

For convenience, JMS defines request/reply helper classes (classes written using JMS) for both the PTP and Pub/Sub domains that implement a basic form of request/reply. JMS providers and clients may provide more specialized implementations.

Basic

This directory contains two sample programs which illustrate the request/reply abstraction supplied by the JMS API using a Native CSharp requestor application and replying application

- **TRequestor.cs** - Reads strings from standard input and use it to send a request message on the topic "primarytopic".
- **TReplier.cs** - Implements an asynchronous listener, which listens on the topic "primarytopic", and replies to each request

FioranoMQ provides a wrapper over the JMS specific Topic and QueueRequestors, namely, FioranoTopicRequestor and FioranoQueueRequestors. These wrappers provide additional APIs to set timeout for requests.

FioranoTopic and FioranoQueue Requestor, now make it mandatory for the replier to set the "correlation-ID" received from the requested message. FioranoRequestor sends data with a "correlation-ID" set, the replier has to send back a reply with the "correlation-ID" set by the requestor.

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the TReplier by executing the TReplier.exe executable file.
3. Run the TRequestor by executing the TRequestor.exe file.

timeout

This directory contains two sample programs which illustrate the request/reply abstraction supplied by the JMS API using a Native CSharp requestor application and replying application

- **TimedTopicRequestor.cs** - Reads strings from standard input and use it to send a request message on the topic "primarytopic".
- **TimedTopicReplier.cs** - Implements an asynchronous listener, which listens on the topic "primarytopic", and replies to each request

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. For convenience, compiled version of the sources is included in this directory. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the Replier by executing the TimedTopicReplier.exe executable file.
3. Run the requestor by executing the TimedTopicRequestor.exe file.

Note: If the Timeout is not to be set, put value as -1.

RevalidateConnections

FioranoMQ server on detecting a loss of connection, notifies the application by invoking the onException ( ) of ExceptionListener object, which is associated with the Connection instance. The loss of connection might be due to number of possible reasons, such as termination of server process and network failure. When working over an unreliable network, the FioranoMQ administrator must enable pinger in the server.

On detecting a failure, an application automatically reconnects through a proprietary API - revalidate ( ), provided by FioranoMQ. In addition, when the application automatically reconnects, its does not lose its state, active Senders and Receivers. This API can be found in FioranoQueueConnection for PTP.

This directory contains two sample programs which illustrate revalidate connection property using the FioranoMQ Native CSharp Runtime Library. These samples revalidate connections when the connection is lost with the FioranoMQ Server. In addition, it restores the validity of created sessions, senders and receivers on this connection.

- **RCPublisher.cs** - Reads strings from standard input and sends them on the topic "primaryTopic".
- **RCSubscriber.cs** - Implements an asynchronous listener, which listens on the topic "primaryTopic", and prints out the received messages.

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the Subscriber by executing the RCSubscriber.exe executable file.
3. Run the Publisher by executing the RCPublisher.exe file.
4. Shut down FioranoMQ server
5. Restart FioranoMQ server.
6. The Publisher and Subscriber will re-establish the connection with the server after sometime and messages will start getting exchanged again.

Transactions

A Session may optionally be specified as transacted. Each transacted session supports a single series of transactions. Each transaction groups a set of produced messages and a set of consumed messages into an atomic unit of work. In effect, transactions organize a session's input message stream and output message stream into series of atomic units. When a transaction commits, its atomic unit of input is acknowledged and its associated atomic unit of output is sent. If a transaction rollback is done, its produced messages are destroyed and its consumed messages are automatically recovered.

This directory contains a sample programs which illustrate JMS Transaction functionality using the FioranoMQ Native CSharp Runtime Library.

- **Transactions.cs -** Implements the sender and receiver, and uses the commit/rollback functionality to demonstrate JMS Transactions

To run this sample using FioranoMQ, do the following:

1. Compile the source files. The %FMQ_DIR%\nativecsharp\scripts directory contains a script called csclientbuild.bat which compiles the Native CSharp program.
2. Run the sample by executing the Transactions.exe executable file. For proper results from the sample, ensure that there are no messages in the primary-Queue.