Download



# 6 Embedding a Message Queue Broker in a Java Client

Message Queue supports running a broker from within a Java client. Such a broker, called an *embedded broker*, runs in the same JVM as the Java client that creates and starts it.

Beyond operating like a normal standalone broker, an embedded broker offers the application in which it is embedded access to a special kind of connection called a *direct mode* connection. Direct mode connections are used just like ordinary connections, but they are much higher performing because they use in-memory transport instead of TCP. To specify a direct mode connection, the client specifies `mq://localhost/direct` as the broker address in the connection factory from which it subsequently creates the connection.

The following sections provide more information about creating and managing embedded brokers:



## Creating, Initializing and Starting an Embedded Broker

To create, initialize, and start an embedded broker, you:

1. Create a broker instance in the client runtime.
2. Create a broker event listener.
3. Define properties to use when initializing the broker instance.
4. Initialize the broker instance.
5. Start the broker instance.

The following listing shows an example of creating, initializing, and starting an Embedded Broker. In this example, *args* represents the string of arguments to pass as properties when initializing the broker instance, and *EmbeddedBrokerEventListener* is an existing class that implements the `BrokerEventListener` interface.

```
import com.sun.messaging.jmq.jmsclient.runtime.BrokerInstance;
import com.sun.messaging.jmq.jmsclient.runtime.ClientRuntime;
import com.sun.messaging.jmq.jmsservice.BrokerEvent;
import com.sun.messaging.jmq.jmsservice.BrokerEventListener;

// Obtain the ClientRuntime singleton object
ClientRuntime clientRuntime = ClientRuntime.getRuntime();

// Create a broker instance
BrokerInstance brokerInstance = clientRuntime.createBrokerInstance();

// Create a broker event listener
BrokerEventListener listener = new EmbeddedBrokerEventListener();

// Convert the broker arguments into Properties. Note that parseArgs is
// a utility method that does not change the broker instance.
Properties props = brokerInstance.parseArgs(args);

// Initialize the broker instance using the specified properties and
// broker event listener
brokerInstance.init(props, listener);

// now start the embedded broker
brokerInstance.start();
```

### Creating a Broker Event Listener

When initializing an embedded broker, you must provide a broker event listener. This listener is an instance of a class that implements the `BrokerEventListener` interface. This interface specifies two methods:

- `public void brokerEvent(BrokerEvent brokerEvent)`, which is called when the broker starts and stops. This method is not required to perform any specific actions, so you can implement an empty method.
- `public boolean exitRequested(BrokerEvent event, Throwable thr)`, which is called when the embedded broker is about to shut down, either because of a user command or because of a fatal error. This method is not required to perform any specific actions, so you can implement an empty method. The return value is ignored.

The following listing shows an example class that implements the `BrokerEventListener` interface.

```
class EmbeddedBrokerEventListener implements BrokerEventListener {

    public void brokerEvent(BrokerEvent brokerEvt) {
        System.out.println ("Received broker event:"+brokerEvent);
    }

    public boolean exitRequested(BrokerEvent event, Throwable thr) {
        System.out.println ("Broker is about to shut down because of:"+event+" with "+thr);
        // return value will be ignored
        return true;
    }
}
```

### Arguments to Specify When Initializing an Embedded Broker

When initializing an embedded broker, you can provide a list of arguments as properties.

Because a Java client runtime (not the `imqbroker` utility) is initializing the broker, you should specify these arguments:

`-imqhome` *path*
The home directory of the Message Queue installation (see "Directory Variable Conventions").

`-libhome` *path*
The directory in which Message Queue libraries are stored, `IMQ_HOME/lib`.

`-varhome` *path*
The directory in which Message Queue temporary or dynamically created configuration and data files are stored installation (see "Directory Variable Conventions").

You can also specify `imqbrokerd` options as arguments. Two useful options to specify as arguments are:

`-name` *instanceName*
The instance name of the broker.

`-port` *portNumber*
The port number for the broker's Port Mapper. This is port number on which the broker listens for client connections.

## Creating a Direct Connection to an Embedded Broker

Once an embedded broker has been started, you can create direct connections to it from the client in which it is embedded. To do so, you create a connection as you would with an ordinary broker, but you specify `mq://localhost/direct` as broker address in the connection factory. For example:

```
com.sun.messaging.ConnectionFactory cf = new com.sun.messaging.ConnectionFactory();
cf.setProperty(ConnectionConfiguration.imqAddressList, "mq://localhost/direct" );
Connection connection = cf.createConnection();
```

## Creating a TCP Connection to an Embedded Broker

Once an embedded broker has been started, clients other than the one in which it is embedded can connect to it as though it were an ordinary standalone broker. For example:

```
com.sun.messaging.ConnectionFactory cf = new com.sun.messaging.ConnectionFactory();
cf.setProperty(ConnectionConfiguration.imqAddressList, "mq://myhost.example.com:7676" );
Connection connection = cf.createConnection();
```

## Stopping and Shutting Down an Embedded Broker

To stop and shut down an embedded broker, use the stop() and shutdown() methods of the broker instance. For example:

```
// Stop the embedded broker
brokerInstance.stop();
// Shut down the embedded broker
brokerInstance.shutdown();
```

## Embedded Broker Example

The following listing demonstrates how to:

- Create, initialize and start an embedded broker
- Create a direct connection
- Send and receive messages across a direct connection
- Stop and shut down an embedded broker
- Create a broker event listener

```
package test.direct;

import java.util.Properties;

import javax.jms.Connection;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.MessageProducer;
import javax.jms.Queue;
import javax.jms.Session;
import javax.jms.TextMessage;

import com.sun.messaging.ConnectionConfiguration;
import com.sun.messaging.jmq.jmsclient.runtime.BrokerInstance;
import com.sun.messaging.jmq.jmsclient.runtime.ClientRuntime;
import com.sun.messaging.jmq.jmsservice.BrokerEvent;
import com.sun.messaging.jmq.jmsservice.BrokerEventListener;

public class EmbeddedBrokerExample {

    public void run(String[] args) throws Exception{

        // obtain the ClientRuntime singleton object
        ClientRuntime clientRuntime = ClientRuntime.getRuntime();

        // create the embedded broker instance
        BrokerInstance brokerInstance = clientRuntime.createBrokerInstance();

        // convert the specified broker arguments into Properties
        // this is a utility function: it doesn't change the broker
        Properties props = brokerInstance.parseArgs(args);

        // initialise the broker instance
        // using the specified properties
        // and a BrokerEventListener
        BrokerEventListener listener = new ExampleBrokerEventListener();
        brokerInstance.init(props, listener);

        // now start the embedded broker
        brokerInstance.start();

        System.out.println ("Embedded broker started");

        // now create a direct connection to the embedded broker
        // this is identical to a normal TCP connection except that a special URL is used
        com.sun.messaging.ConnectionFactory qcf = new com.sun.messaging.ConnectionFactory();
        qcf.setProperty(ConnectionConfiguration.imqAddressList, "mq://localhost/direct");

        Connection connection = qcf.createConnection();
        System.out.println ("Created direct connection to embedded broker");

        // now create a session and a producer and consumer in the normal way
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        Queue queue = session.createQueue("exampleQueue");
        MessageConsumer consumer = session.createConsumer(queue);
        MessageProducer producer = session.createProducer(queue);

        // send a message to the queue in the normal way
        TextMessage textMessage = session.createTextMessage("This is a message");
        producer.send(textMessage);

        // receive a message from the queue in the normal way
        connection.start();
        Message receivedMessage = consumer.receive(1000);
        System.out.println ("Received message "+((TextMessage)receivedMessage).getText());

        // close the client connection
        connection.close();

        // stop the embedded broker
        brokerInstance.stop();

        // shutdown the embedded broker
        brokerInstance.shutdown();

    }

    public static void main(String[] args) throws Exception {

        EmbeddedBrokerExample ebe = new EmbeddedBrokerExample();
        ebe.run(args);

    }

    class ExampleBrokerEventListener implements BrokerEventListener {

        public void brokerEvent(BrokerEvent brokerEvt) {
            System.out.println ("Received broker event:"+brokerEvent);
        }

        public boolean exitRequested(BrokerEvent event, Throwable thr) {
            System.out.println ("Broker is about to shut down because of:"+event+" with "+thr);

            // return value will be ignored
            return true;
        }
    }
}
```