



**habarisoft**  
Enterprise Messaging Software for Delphi®

# Getting started with Habari® Client for HornetQ

*Version 1.7*

---

## Trademarks

Habari is a registered trademark of Michael Justin and is protected by the laws of Germany and other countries. JBoss and the JBoss logo are registered trademarks of Red Hat, Inc. Oracle and Java are registered trademarks of Oracle and/or its affiliates. All Embarcadero brands and product names are trademarks or registered trademarks of Embarcadero. Microsoft, Windows, Windows NT, and/or other Microsoft products referenced herein are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other brands and their products are trademarks of their respective holders.

## Contents

<b>Introduction</b> .....	<b>4</b>
<b>About Habari Client for HornetQ</b> .....	<b>4</b>
<b>About HornetQ</b> .....	<b>6</b>
<b>Installation</b> .....	<b>7</b>
<b>Requirements</b> .....	<b>7</b>
<b>TCP/IP Communication Libraries</b> .....	<b>8</b>
<b>Starting HornetQ</b> .....	<b>9</b>
<b>Pre-Installation Requirements</b> .....	<b>9</b>
<b>Stomp Configuration</b> .....	<b>9</b>
<b>Running the Broker</b> .....	<b>11</b>
<b>Monitoring HornetQ</b> .....	<b>11</b>
<b>Stopping HornetQ</b> .....	<b>12</b>
<b>Communication Adapter Configuration</b> .....	<b>13</b>
<b>Introduction</b> .....	<b>13</b>
<b>The JMS API Programming Model</b> .....	<b>14</b>
<b>Connections and Sessions</b> .....	<b>15</b>
<b>Step by Step Example</b> .....	<b>15</b>
<b>Transacted Sessions</b> .....	<b>17</b>
<b>Failover Support</b> .....	<b>17</b>
<b>Destinations</b> .....	<b>19</b>
<b>Introduction</b> .....	<b>19</b>
<b>Create a new Destination</b> .....	<b>19</b>
<b>Producer and Consumer</b> .....	<b>21</b>
<b>Message Producer</b> .....	<b>21</b>
<b>Message Consumer</b> .....	<b>21</b>
<b>Text Messages</b> .....	<b>23</b>
<b>Sending a TextMessage</b> .....	<b>23</b>
<b>Receive Text Messages</b> .....	<b>24</b>
<b>Binary Messages</b> .....	<b>26</b>
<b>Send Binary Messages</b> .....	<b>26</b>
<b>Object Messages</b> .....	<b>27</b>
<b>Introduction</b> .....	<b>27</b>
<b>Message Transformer Implementations</b> .....	<b>27</b>
<b>Map Messages</b> .....	<b>30</b>
<b>Introduction</b> .....	<b>30</b>
<b>Durable Subscriptions</b> .....	<b>32</b>
<b>Description</b> .....	<b>32</b>
<b>Example Applications</b> .....	<b>33</b>
<b>Example Application Index</b> .....	<b>33</b>
<b>ConsumerTool</b> .....	<b>35</b>
<b>ProducerTool</b> .....	<b>37</b>
<b>Performance Test</b> .....	<b>39</b>
<b>JBoss AS 5 Integration</b> .....	<b>41</b>
<b>Tutorial</b> .....	<b>41</b>
<b>Creating AS 5 Profiles</b> .....	<b>41</b>
<b>Broker Configuration</b> .....	<b>42</b>
<b>Start JBoss</b> .....	<b>42</b>
<b>Running the Delphi Test</b> .....	<b>42</b>

<b>JBoss AS 7.1 integration</b> .....	<b>43</b>
<b>Management API</b> .....	<b>44</b>
<b>Management Notifications</b> .....	<b>44</b>
<b>Message Options</b> .....	<b>45</b>
<b>JMS Standard Properties</b> .....	<b>45</b>
<b>Conditional Symbols</b> .....	<b>46</b>
<b>HABARI_LOGGING</b> .....	<b>46</b>
<b>HABARI_RAW_TRACE</b> .....	<b>46</b>
<b>HABARI_STOMP_11</b> .....	<b>46</b>
<b>HABARI_USE_RTTI</b> .....	<b>46</b>
<b>Known Limitations</b> .....	<b>47</b>
<b>Broker Limitations</b> .....	<b>47</b>
<b>Sessions</b> .....	<b>47</b>
<b>Messages</b> .....	<b>47</b>
<b>Destinations</b> .....	<b>48</b>
<b>JMSXGroupID</b> .....	<b>48</b>
<b>References</b> .....	<b>49</b>
<b>Habari Client for HornetQ License</b> .....	<b>50</b>
<b>Third Party Library Licenses</b> .....	<b>52</b>
<b>Synapse</b> .....	<b>52</b>
<b>Indy BSD License</b> .....	<b>53</b>
<b>SuperObject</b> .....	<b>53</b>
<b>Log4D</b> .....	<b>54</b>
<b>NativeXml</b> .....	<b>54</b>
<b>Release Notes</b> .....	<b>55</b>
<b>Version 1.7</b> .....	<b>55</b>
<b>Version 1.6</b> .....	<b>56</b>
<b>Version 1.5</b> .....	<b>56</b>
<b>Version 1.4</b> .....	<b>57</b>
<b>Version 1.3</b> .....	<b>58</b>
<b>Version 1.2</b> .....	<b>58</b>
<b>Version 1.1</b> .....	<b>59</b>
<b>Version 1.0</b> .....	<b>60</b>
<b>Index</b> .....	<b>61</b>

# Introduction

---

## About Habari Client for HornetQ

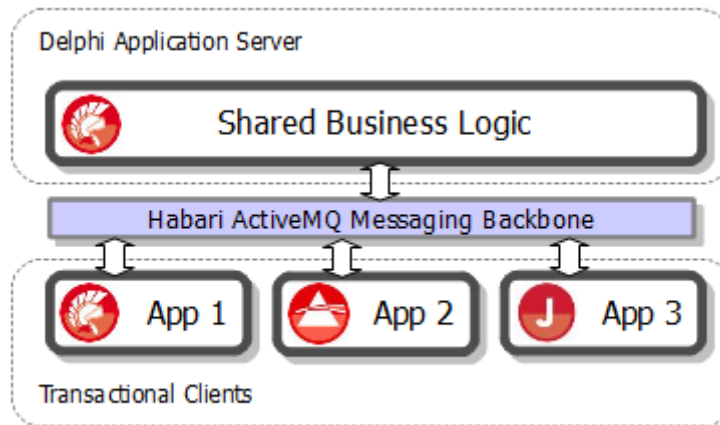
Habari Client for HornetQ is a library for Delphi® and Free Pascal which provides easy access to HornetQ. Habari Client for HornetQ uses a plug-in style architecture for communication libraries. It supports HornetQ 2.1 and newer, Delphi 2009 to XE2 and Free Pascal, and follows the specification of the JMS API for Message Oriented Middleware.

## How Can I Use It?

Here are some examples for software solutions built on top of a Message Broker like HornetQ:

- **Application Server Integration:** HornetQ is integrated into JBoss and Red Hat application server products.
- **Intranet News Ticker Application:** using the publish and subscribe communication model, news can be delivered to all registered client applications. The message sender works like a broadcast station, and does not care if clients don't listen.
- **Load Balancing:** using the point-to-point or queuing model, many 'worker' applications can be installed on different computers. Every new message sent to the queue will be delivered only to one client. The server will keep messages until they are expired or delivered to a client.
- **Persistent Storage:** messages and objects can be stored in the Object Broker and retrieved even after a restart.
- **Interprocess Communication:** applications can use point-to-point messages to exchange information between each other even if the receiver currently is not running.

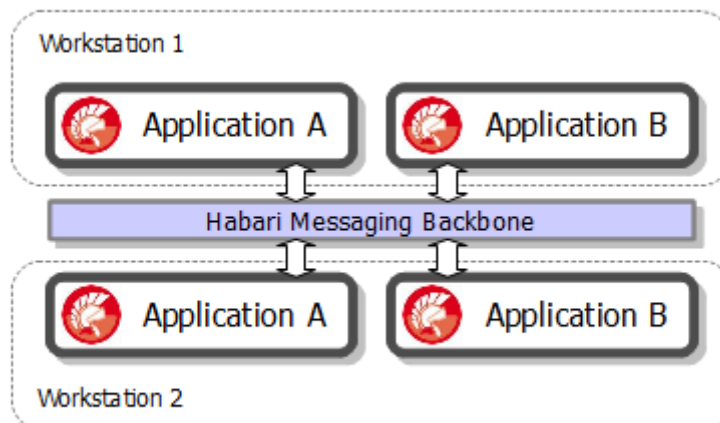
## Example Illustrations



### Habari for shared business logic

Similar to SOAP or REST servers, Delphi software systems can use Habari to provide business logic to other processes.

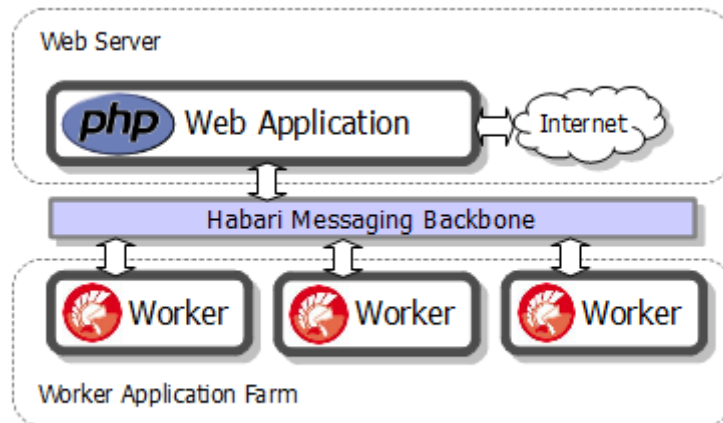
Documents and messages (including objects, serialized using JSON or XML) can be exchanged and secured by **client-side acknowledgment** and **transactional sessions**.



### Habari in a network of Delphi applications

This illustration shows different Delphi applications running in a local network, using Habari client libraries to implement **Interprocess communication**: applications use point-to-point messages to exchange information between each other even if the receiver currently is not running.

Using the **publish/subscribe** communication model, news can be delivered to all registered client applications. The message sender works like a broadcast station, and does not care if clients don't listen.



### Habari in a load balancing solution

In this example, a PHP web application sends data to the message queue. The Habari communication layer in the Delphi worker applications takes care of receiving and acknowledging incoming messages.

Using the point-to-point or queuing model, many 'worker' applications can be installed on different computers. Every new message sent to the **message queue** will be delivered only to one client. The message broker will keep messages until they are expired or delivered to a client.

---

## About HornetQ

HornetQ is an open source project to build a multi-protocol, embeddable, very high performance, clustered, asynchronous messaging system.

### HornetQ Features

<http://community.jboss.org/wiki/HornetQFeatures>

# Installation

---

## Requirements

### Development Environment

- Delphi 2009 or higher
- Free Pascal 2.6.0

### Message Broker

- HornetQ 2.2.2.Final
- HornetQ 2.2.5.Final
- HornetQ 2.2.12.Final (included in JBoss AS 7.1.1.Final)
- HornetQ 2.2.14.Final

### TCP/IP Communication Library

See the next chapter for a discussion of all communication libraries and a feature matrix.

### Internet Direct (Indy)

Subversion repository access:

<https://svn.atozed.com:444/svn/Indy10/trunk>

Unofficial snapshots:

<http://indy.fulgan.com/ZIP>

### Synapse

Subversion repository access:

<https://synalist.svn.sourceforge.net/svnroot/synalist/trunk/>

## TCP/IP Communication Libraries

### Supported libraries

#### Internet Direct (Indy) 10

The communication adapter for Indy supports both GUI-based and console mode applications, and works with Delphi 2009 to XE2 and Free Pascal.

The library has been tested with these versions of Internet Direct:

- Indy 10.5.8

#### Synapse

The communication adapter for Synapse supports both GUI-based and console mode applications, and works with Delphi 2009 to XE2 and Free Pascal.

The library has been tested with these versions of Synapse:

- Release 144

# Starting HornetQ

---

## Pre-Installation Requirements

For installation requirements, please check these documents:

- <http://docs.jboss.org/hornetq/2.2.5.Final/quickstart-guide/en/html/installation.html>  
and
- <http://community.jboss.org/wiki/HornetQTechnicalFAQ>

---

## Stomp Configuration

HornetQ provides native support for Stomp. To be able to send and receive Stomp messages, you must configure a `NettyAcceptor` with a `protocol` parameter set to `stomp`.

To add an 'Acceptor' for the Stomp protocol, it is necessary to edit the broker configuration file `hornetq-configuration.xml`

Navigate to the directory `<hornetq>\config\stand-alone\non-clustered\` and add the acceptor in the section 'acceptors':

```
<acceptors>
  ...
  <acceptor name="stomp">
    <factory-
class>org.hornetq.core.remoting.impl.netty.NettyAcceptorFactory</factory-class>
    <param key="protocol" value="stomp"/>
    <param key="port" value="61613"/>
  </acceptor>
  ...
</acceptors>
```

If the server is running on a different system (for example, a Linux virtual machine):

```
<acceptors>
  ...
  <acceptor name="stomp">
```

```
<factory-class>org.hornetq.core.remoting.impl.netty.NettyAcceptorFactory</factory-
class>
  <param key="protocol" value="stomp"/>
  <param key="host" value="{hornetq.remoting.netty.host:0.0.0.0}"/>
  <param key="port" value="61613"/>
</acceptor>
...
</acceptors>
```

## Destinations

Configure queues and topics in the `hornetq-jms.xml` file. Example:

```
<configuration ...>
...
  <queue name="ExampleQueue">
    <entry name="/queue/ExampleQueue"/>
  </queue>
...
</configuration>
```

## Permissions

Some HornetQ destination commands require user permissions which are not enabled in the default configuration. They can be added in the `<security-settings>` section of `hornetq-configuration.xml`:

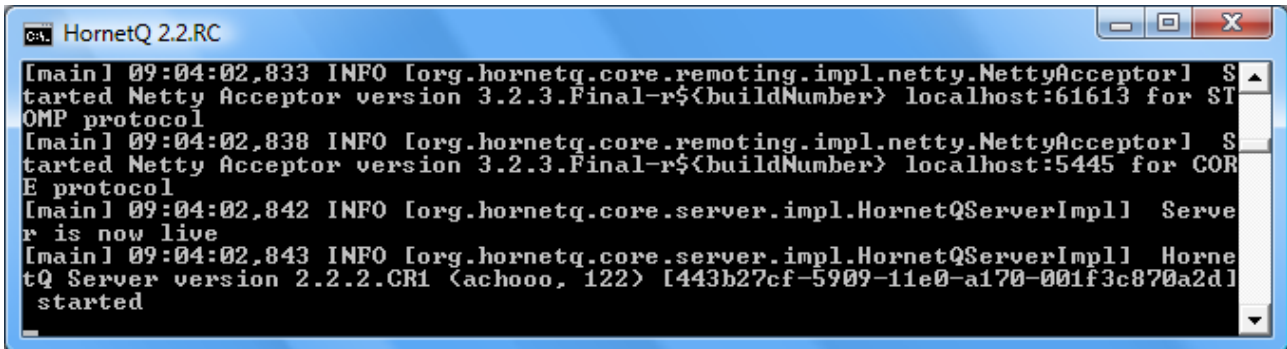
```
<configuration ...>
...
  <security-settings>
    <security-setting match="#">
      <permission type="createNonDurableQueue" roles="guest"/>
      <permission type="deleteNonDurableQueue" roles="guest"/>
      <permission type="createDurableQueue" roles="guest"/>
      <permission type="consume" roles="guest"/>
      <permission type="send" roles="guest"/>
    </security-setting>
  </security-settings>
...
</configuration>
```

Additional information is available in the HornetQ documentation.

<http://docs.jboss.org/hornetq/2.2.5.Final/user-manual/en/html/interoperability.html#stomp>

## Running the Broker

- Navigate to the directory `<hornetq>\bin\`
- Execute `run.bat`

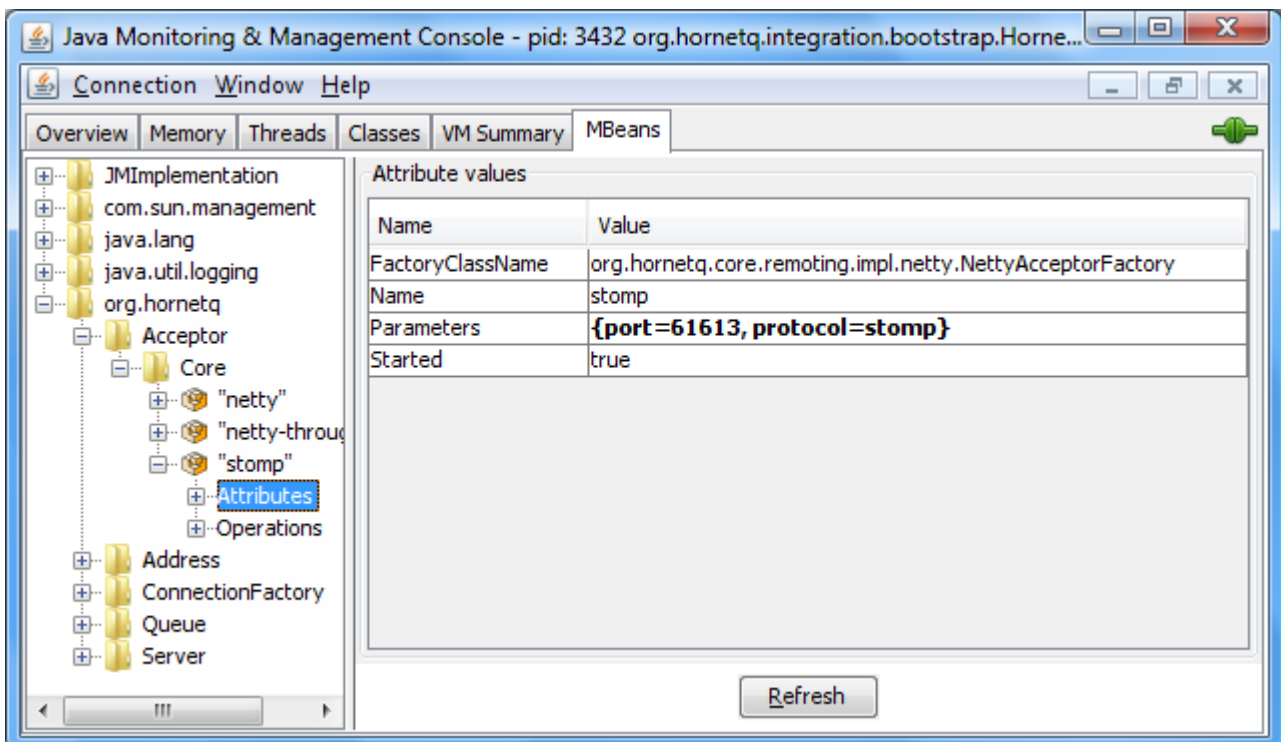


```
[main] 09:04:02,833 INFO [org.hornetq.core.remoting.impl.netty.NettyAcceptor] S
tarted Netty Acceptor version 3.2.3.Final-r${buildNumber} localhost:61613 for ST
OMP protocol
[main] 09:04:02,838 INFO [org.hornetq.core.remoting.impl.netty.NettyAcceptor] S
tarted Netty Acceptor version 3.2.3.Final-r${buildNumber} localhost:5445 for COR
E protocol
[main] 09:04:02,842 INFO [org.hornetq.core.server.impl.HornetQServerImpl] Serve
r is now live
[main] 09:04:02,843 INFO [org.hornetq.core.server.impl.HornetQServerImpl] Horne
tQ Server version 2.2.2.CR1 (achoo, 122) [443b27cf-5909-11e0-a170-001f3c870a2d]
started
```

Illustration 1: HornetQ log

The log shows that the STOMP Acceptor is listening for connections on port 61613.

## Monitoring HornetQ



Java Monitoring & Management Console - pid: 3432 org.hornetq.integration.bootstrap.Horne...

Connection Window Help

Overview Memory Threads Classes VM Summary MBeans

Attributes

Name	Value
FactoryClassName	org.hornetq.core.remoting.impl.netty.NettyAcceptorFactory
Name	stomp
Parameters	{port=61613, protocol=stomp}
Started	true

Refresh

Illustration 2: JMX console

You can use the JMX support to view the state of HornetQ.

## Stopping HornetQ

- Navigate to the directory <hornetq>\config\bin\
- Execute stop.bat

# Communication Adapter Configuration

---

## Introduction

Habari uses communication adapters as an abstraction layer between the internal library and the TCP/IP library. These adapters are implemented using a common API, which allows to exchange them easily, even at runtime.

## Installation of Communication Adapter classes

A communication adapter implementation can be prepared for usage by simply adding its Delphi unit to the project. Behind the scenes, the communication adapter will add itself to the communication adapter list in the BTAdapterRegistry unit. If more than one communication adapter is in the project, the first adapter class in the list will be the default adapter. (The methods of the adapter registry performs some checks, for example to prevent duplicate entries in the adapter list, and raise exceptions in case of errors)

No additional setup of communication adapters is required. At run time, the JMS connection class will pick the default adapter from this list.

The default adapter can be changed at runtime by setting the adapter class (either by its name or by its type).

## Available Communication Adapters

The Habari Client for HornetQ libraries includes two adapters for TCP/IP libraries, one for Indy (Internet Direct) and one for Synapse.

### Indy (Internet Direct)

The Indy adapter requires Indy 10.5.8.

### Synapse

The Synapse adapter requires Synapse revision 144.

# The JMS API Programming Model

The Java online documentation contains a description of the JMS API Programming model<sup>1</sup>:

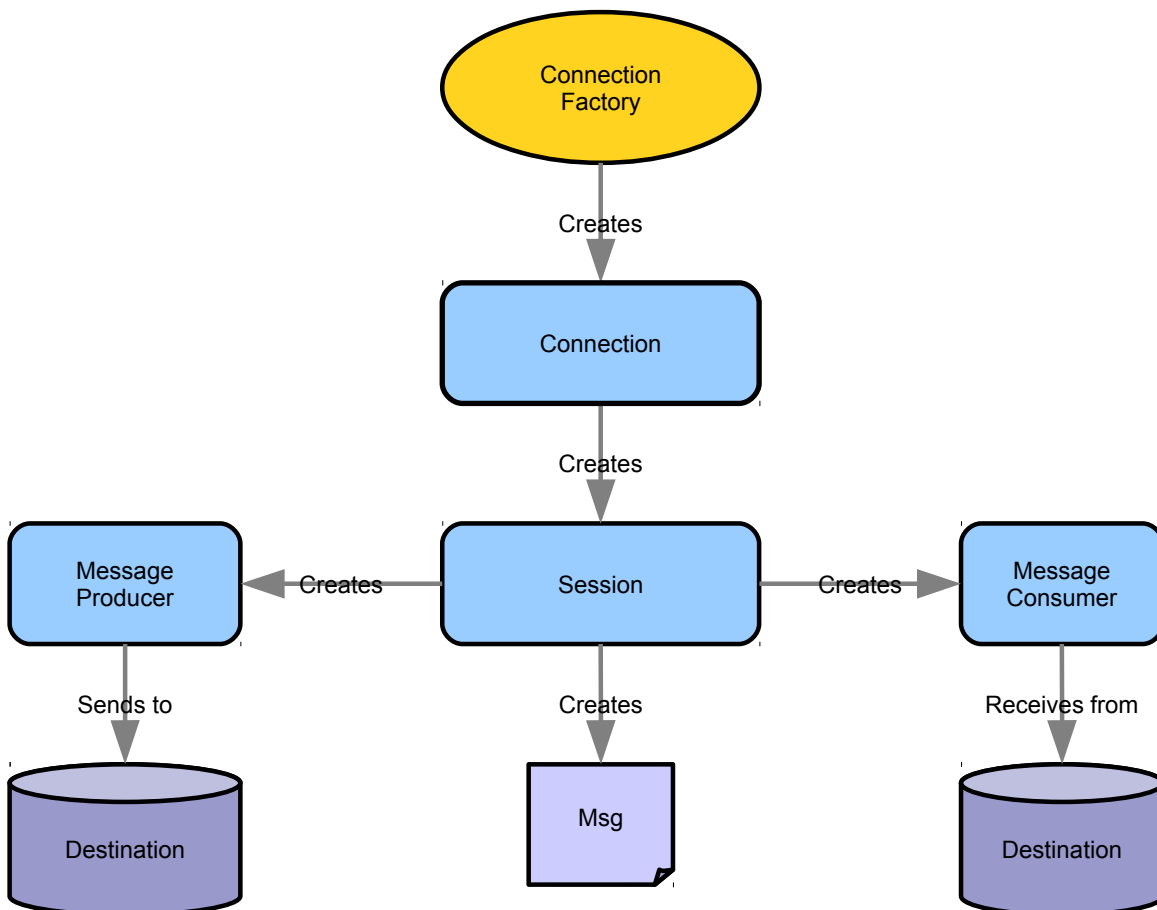


Illustration 1: The JMS API Programming Model: Overview

1 <http://download.oracle.com/javaee/5/tutorial/doc/bnceh.html>

# Connections and Sessions

---

## Step by Step Example

### Add required units

Three units are required for this example

- a communication adapter unit (e. g. BTCommAdapterIndy)
- a connection factory unit (BTJMSConnectionFactory or BTJMSConnection)
- the unit containing the interface declarations (BTJMSInterfaces)

The SysUtils unit is necessary for the exception handling.

```
program SendOneMessage;  
  
{$APPTYPE CONSOLE}  
  
uses  
    SysUtils,  
    BTCommAdapterIndy,  
    BTJMSConnection,  
    BTJMSInterfaces;  
...
```

### Creating a new Connection

To create a new connection,

- declare a variable of type IConnection
- use the helper method MakeConnection of the TBTJMSConnection class to create and configure a new connection with user name, password and the broker URL

or

- use an instance of TBTJMSConnectionFactory to create connections

Since IConnection is an interface type, the connection instance will be destroyed automatically if there are no more references to it in the program. Note that there is no call to Connection.Free in the source.

```
var
  Connection: IConnection;
  Session: ISession;
  Destination: IDestination;
  Producer: IMessageProducer;
begin
  Connection := TBTJMSConnection.MakeConnection('', '', 'stomp://localhost');
  Connection.Start;
```

## Local connection

If you just need a connection to the broker on the local computer using default port number and login credentials, you can call `MakeConnection` without parameters:

```
Connection := TBTJMSConnection.MakeConnection;
```

## Creating a Session

To create the communication session,

- declare a variable of type `ISession`
- use the helper method `CreateSession` of the connection, and specify if it is a transacted session, and the acknowledgement mode

Please check the API documentation for the different session types and acknowledgement modes.

Since `ISession` is an interface type, the session instance will be destroyed automatically if there are no more references to it in the program. Note that there is no call to `Session.Free` in the source.

```
Session := Connection.CreateSession(False, amClientAcknowledge);
```

## Using the Session

The `Session` variable is ready to use now. Destinations, producers and consumers will be covered in the next chapters.

```
Destination := Session.CreateQueue('testqueue');
Producer := Session.CreateProducer(Destination);
Producer.Send(Session.CreateTextMessage('This is a test message'));
```

## Closing a Connection

Finally, the application closes the connection. The client will disconnect from the message broker. Closing a connection also implicitly closes all open sessions.

```
finally
    Connection.Close;
end;
end.
```

---

## Transacted Sessions

A session may be specified as transacted. Each transacted session supports a single series of transactions. Each transaction groups a set of message sends and a set of message receives 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, the transaction's sent messages are destroyed and the session's input is automatically recovered.

The content of a transaction's input and output units is simply those messages that have been produced and consumed within the session's current transaction.

A transaction is completed using either its session's Commit method or its session's Rollback method. The completion of a session's current transaction automatically begins the next. The result is that a transacted session always has a current transaction within which its work is done.

---

## Failover Support

The Failover transport layers reconnect logic on top of the Stomp transport.

The Failover configuration syntax allows you to specify any number of composite URIs. The Failover transport randomly chooses one of the composite URI and attempts to establish a connection to it. If it does not succeed, a new connection is established to one of the other URIs in the list.

Example for a failover URI:

```
failover:(stomp://primary:61613,stomp://secondary:61613)
```

## Transport Options

Option Name	Default Value	Description
initialReconnectDelay	10	How long to wait before the first reconnect attempt (in ms)
maxReconnectDelay	30000	The maximum amount of time we ever wait between reconnect attempts (in ms)
backOffMultiplier	2	The exponent used in the exponential backoff attempts
maxReconnectAttempts	0	If not 0, then this is the maximum number of reconnect attempts before an error is sent back to the client
randomize	True	use a random algorithm to choose the the URI to use for reconnect from the list provided

Table 1: Failover transport options

Example URI:

```
failover:(tcp://localhost:61616,tcp://remotehost:61616)?  
initialReconnectDelay=100&maxReconnectAttempts=10
```

Example code:

```
with TBTJMSConnectionFactory.Create('failover:  
(stomp://primary:61616,stomp://localhost:61613)?maxReconnectAttempts=3') do  
try  
    Conn := CreateConnection;  
    Conn.Start;  
    Conn.Stop;  
    Conn.Close;  
finally  
    Free;  
end;
```

# Destinations

---

## Introduction

The JMS API supports two models:<sup>2</sup>

1. point-to-point or queuing model
2. publish and subscribe model

In the point-to-point or queuing model, a producer posts messages to a particular queue and a consumer reads messages from the queue. Here, the producer knows the destination of the message and posts the message directly to the consumer's queue. It is characterized by following:

- Only one consumer will get the message
- The producer does not have to be running at the time the receiver consumes the message, nor does the receiver need to be running at the time the message is sent
- Every message successfully processed is acknowledged by the receiver

The publish/subscribe model supports publishing messages to a particular message topic. Zero or more subscribers may register interest in receiving messages on a particular message topic. In this model, neither the publisher nor the subscriber know about each other. A good metaphor for it is anonymous bulletin board. The following are characteristics of this model:

- Multiple consumers can get the message
- There is a timing dependency between publishers and subscribers. The publisher has to create a subscription in order for clients to be able to subscribe. The subscriber has to remain continuously active to receive messages, unless it has established a durable subscription. In that case, messages published while the subscriber is not connected will be redistributed whenever it reconnects.

---

## Create a new Destination

### Queues

A queue can be created using the CreateQueue method of the Session. Example:

```
Queue queue = session.createQueue("queueName");
```

---

<sup>2</sup> Java Message Service. (2007, November 21). In Wikipedia, The Free Encyclopedia. [http://en.wikipedia.org/wiki/Java\\_Message\\_Service](http://en.wikipedia.org/wiki/Java_Message_Service)

```
Destination := Session.CreateQueue('ExampleQueue');  
Consumer := Session.CreateConsumer(Destination);
```

The queue can then be used to send or receive messages using implementations of the `IMessageProducer` and `IMessageConsumer` interfaces. (See next chapter for an example)

## Topics

A topic can be created using the `CreateTopic` method of the `Session`. Example:

```
Destination := Session.CreateTopic('ExampleTopic');  
Consumer := Session.CreateConsumer(Destination);
```

The topic can then be used to send or receive messages using implementations of the `IMessageProducer` and `IMessageConsumer` interfaces. (See next chapter for an example).

# Producer and Consumer

---

## Message Producer

A client uses a MessageProducer object to send messages to a destination. A MessageProducer object is created by passing a Destination object to a message-producer creation method supplied by a session.

Example:

```
...
    Destination := Session.CreateQueue('ExampleQueue');
    Producer := Session.CreateProducer(Destination);
    Producer.Send(Session.CreateTextMessage('Test message'));
...
```

A client can specify a default delivery mode, priority, and time to live for messages sent by a message producer. It can also specify the delivery mode, priority, and time to live for an individual message.

---

## Message Consumer

A client uses a MessageConsumer object to receive messages from a destination. A MessageConsumer object is created by passing a Destination object to a message-consumer creation method supplied by a session.

Example:

```
...
    Destination := Session.CreateQueue('ExampleQueue');
    Consumer := Session.CreateConsumer(Destination);
...
```

A message consumer can be created with a **message selector**. A message selector allows the client to restrict the messages delivered to the message consumer to those that match the selector.

A client may either synchronously receive a message consumer's messages or have the consumer asynchronously deliver them as they arrive.

For synchronous receipt, a client can request the next message from a message consumer using one of its receive methods. There are several variations of receive that allow a client to poll or wait for the next message.

For asynchronous delivery, a client can register a `MessageListener` object with a message consumer. As messages arrive at the message consumer, it delivers them by calling the `MessageListener`'s `OnMessage` method.

# Text Messages

## Sending a TextMessage

Source code for a simple application which sends a test message:

```
program SendOneMessage;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  BTCommAdapterIndy,
  BTJMSConnection,
  BTJMSInterfaces;

var
  Connection: IConnection;
  Session: ISession;
  Destination: IDestination;
  Producer: IMessageProducer;

begin
  Connection := TBTJMSConnection.MakeConnection('', '', 'stomp://localhost');
  Connection.Start;
  try
    Session := Connection.CreateSession(False, amAutoAcknowledge);
    WriteLn('Send a message');
    Destination := Session.CreateQueue('onemessage');
    Producer := Session.CreateProducer(Destination);
    Producer.Send(Session.CreateTextMessage('This is a test message'));
    WriteLn('Hit any key');
    ReadLn;
  finally
    Connection.Close;
  end;
end.
```

The unit BTCommAdapterIndy contains the Internet Direct (Indy) communication adapter class. By including this unit, it will register the adapter class with an internal list of all

available communication adapters. By default, the first registered communication adapter will be used.

---

## Receive Text Messages

### Asynchronous receive

To receive text messages asynchronously, the client subscribes to a destination (which can be a queue or a topic) on the server.

The messages will be delivered to an event handler which has to be provided by the client.

```
var
  Destination: IDestination;
  Consumer: IMessageConsumer;

begin
  ...
  // create a destination queue
  Destination := Session.CreateQueue('test');

  // create a consumer
  Consumer := Session.CreateConsumer(Destination);

  // set the message listener
  Consumer.MessageListener := Self;
  ...
end;
```

The asynchronous MessageListener is an object which implements the IMessageListener interface.

This interface only contains one procedure, OnMessage:

```
IMessageListener = interface(IIInterface)
  procedure OnMessage(Message: IMessage);
end;
```

## Synchronous Receive

A MessageConsumer offers a Receive method which can be used to consume exactly one message at a time.

Example:

```
while I < EXPECTED do
begin
  Msg := Consumer.Receive(1000) as ITextMessage;
  if Assigned(Msg) then
  begin
    Inc(I);
    WriteLn(Format('%d %s', [I, Msg.Text]));
  end;
end;
```

Compared with a MessageListener, the Receive method has the advantage that the application can stop consuming messages at any point in time (for example, after receiving 20 messages). With an asynchronous MessageListener, it is possible that the MessageConsumer will still receive some messages after calling the close method.

## Receive and ReceiveNowait

There are three different methods for synchronous receive:

- |                         |  |
|-------------------------|--|
| <b>Receive</b>          | The Receive method with no arguments will block (wait until a message is available).   |
| <b>Receive(Timeout)</b> | The Receive method with a timeout parameter will wait for the given time in milliseconds. If no message arrived, it will return nil. |
| <b>ReceiveNowait</b>    | The ReceiveNowait method will return immediately. If no message arrived, it will return nil.   |

# Binary Messages

---

## Send Binary Messages

### Reading Binary Content using BTStreamHelper

The BTStreamHelper unit contains the procedure LoadBytesFromStream which can be used to read a file into a BytesMessage. Example:

```
// create the message
Msg := Session.CreateBytesMessage;

// open a file
FS := TFileStream.Create('filename.dat', fmOpenRead);

try
  // read the file bytes into the message
  LoadBytesFromStream(Msg, FS);

  Size := Length(Msg.Content);

  // display message content size
  WriteLn(IntToStr(Size) + ' Bytes');

finally
  // release the file stream
  FS.Free;
end;
```

# Object Messages

---

## Introduction

### Object Serialization

Object serialization is the process of saving an object's state to a sequence of bytes, as well as the process of rebuilding those bytes into a live object at some future time.<sup>3</sup> In messaging applications, object serialization is required to transfer objects between clients, but also to store objects on the broker if they are declared persistent.

---

## Message Transformer Implementations

Transformation	Message Type	Library	Unit
<b>XML</b>	<b>ObjectMessage</b>	<b>OmniXML</b>	BTMessageTransformerXMLOmni
<b>XML</b>	<b>ObjectMessage</b>	<b>NativeXml</b>	BTMessageTransformerXMLNative
<b>XML</b>	<b>MapMessage</b>	<b>OmniXML</b>	BTMessageTransformerXMLMapOmni
<b>XML</b>	<b>MapMessage</b>	<b>NativeXml</b>	BTMessageTransformerXMLMapNative
<b>JSON</b>	<b>ObjectMessage</b>	<b>SuperObject</b>	BTMessageTransformerJSONSuperObject

Table 2: Message Transformer Implementations

## Memory Management

### Outgoing Objects

The message transformer will not free objects which have been sent. To release the memory, the application has to explicitly free them when they are no longer used.

### Incoming Objects

The message transformer will create an object instance when a object message has been received. To avoid memory leaks, the application must free this instance when it is no longer in use.

---

<sup>3</sup> <http://java.sun.com/developer/technicalArticles/Programming/serialization/>

## Assign a Message Transformer

To insert a object decoder / encoder in the message processing chain, create a message transformer instance and assign it to the connection's MessageTransformer property.

The constructor of message transformers for object exchange takes one argument, which is the class of the serialized object. In this example, SamplePojo is the class.

```
Connection: IConnection;
...

with (Connection as IMessageTransformerSupport) do
begin
  MessageTransformer := TBTMessageTransformerXMLOmni.Create(SamplePojo);
end;

...
Connection.Start;
```

You can also use the helper procedure SetTransformer in unit BTJMSConnection:

```
Connection: IConnection;
...

SetTransformer(Connection, TBTMessageTransformerXMLOmni.Create(SamplePojo));

...
Connection.Start;
```

## Create and Send an ObjectMessage

1. create a IObjectMessage instance using ISession#CreateObjectMessage
2. send the object message to the broker using IMessageProducer#Send

```
ObjectMessage := Session.CreateObjectMessage(Instance);
Producer.Send(ObjectMessage);
```

## Complete Example using NativeXml

From ObjectExchangeTests.pas.

Send:

```
procedure TObjExTestCase.TestXMLNative;
var
  ObjectMessage: IObjectMessage;
  Obj: SamplePojo;
begin
  // send
  Connection := TBTJMSConnection.MakeConnection;
```

```
try
  SetTransformer(Connection, TBMessageTransformerXMLNative.Create(SamplePojo));
  Connection.Start;
  Session := Connection.CreateSession(False, amAutoAcknowledge);
  Destination := Session.CreateQueue('TOOL.OBJECT.XML');
  Producer := Session.CreateProducer(Destination);
  Obj := SamplePojo.Create;
  try
    Obj.messageText := 'test';
    Obj.messageNo := 0;
    ObjectMessage := Session.CreateObjectMessage(Obj);
    ObjectMessage.SetStringProperty(SH_TRANSFORMATION + '-custom',
      TRANSFORMER_ID_OBJECT_XML); // required for "Delphi Only" object exchange
    Producer.Send(ObjectMessage);
  finally
    Obj.Free;
  end;
finally
  Connection.Close;
end;
```

Receive:

```
Connection := TBTJMSConnection.MakeConnection;
try
  SetTransformer(Connection, TBMessageTransformerXMLNative.Create(SamplePojo));
  Connection.Start;
  Session := Connection.CreateSession(False, amAutoAcknowledge);
  Destination := Session.CreateQueue('TOOL.OBJECT.XML');
  Consumer := Session.CreateConsumer(Destination);
  ObjectMessage := Consumer.Receive(1000) as IObjectMessage;
  if Assigned(ObjectMessage) then
    begin
      Obj := ObjectMessage.GetObject as SamplePojo;
      try
        CheckEquals('test', Obj.messageText);
        CheckEquals(0, Obj.messageNo);
      finally
        Obj.Free;
      end;
    end;
  finally
    Connection.Close;
  end;
end;
```

# Map Messages

---

## Introduction

The JMS API supports map messages which consist of key-value pairs. Habari RabbitMQ Client includes implementations (based on OmniXML and NativeXml) of map message support. They serialize all entries as string values at the moment.

Map message transformers take a nil parameter as argument.

## Complete Example

This example uses NativeXml, and is taken from ObjectExchangeTests.pas.

Send:

```
procedure TObExTestCase.TestXMLMapNative;
var
  MapMessage: IMapMessage;
begin
  // send
  Connection := TBTJMSConnection.MakeConnection;
  try
    SetTransformer(Connection, TBTMessageTransformerXMLMapNative.Create(nil));
    Connection.Start;
    Session := Connection.CreateSession(False, amAutoAcknowledge);
    Destination := Session.CreateQueue('TOOL.MAP.XML');
    Producer := Session.CreateProducer(Destination);
    MapMessage := Session.CreateMapMessage;
    MapMessage.SetString('first', '1');
    MapMessage.SetString('second', '2');
    Producer.Send(MapMessage);
  finally
    Connection.Close;
  end;
end;
```

Receive:

```
Connection := TBTJMSConnection.MakeConnection;
try
  SetTransformer(Connection, TBTMessageTransformerXMLMapNative.Create(nil));
  Connection.Start;
  Session := Connection.CreateSession(False, amAutoAcknowledge);
  Destination := Session.CreateQueue('TOOL.MAP.XML'
    + '?transformation=' + TRANSFORMER_ID_MAP_XML);
  Consumer := Session.CreateConsumer(Destination);
```

```
MapMessage := Consumer.Receive(1000) as IMapMessage;
if Assigned(MapMessage) then
begin
    CheckEquals('1', MapMessage.GetString('first'));
    CheckEquals('2', MapMessage.GetString('second'));
end;
finally
    Connection.Close;
end;
end;
```

# Durable Subscriptions

---

## Description

If a client needs to receive all the messages published on a topic, including the ones published while the subscriber is inactive, it uses a durable TopicSubscriber. The JMS provider retains a record of this durable subscription and insures that all messages from the topic's publishers are retained until they are acknowledged by this durable subscriber or they have expired.<sup>4</sup> The combination of the clientId and durable subscriber name uniquely identifies the durable topic subscription. After you restart your program and re-subscribe, the Broker will know which messages you need that were published while you were away.

## Creation

The Session interface contains the CreateDurableSubscriber method which creates a durable subscriber to the specified topic. A JMS durable subscriber MessageConsumer is created with a unique JMS clientId and durable subscriber name. Only **one** thread can be actively consuming from a given logical topic subscriber.

**Note:** For durable topic subscriptions you must specify the same clientId on the connection and subscriptionName on the subscribe.

**Incomplete** This feature is still in development

---

<sup>4</sup> <http://download.oracle.com/javaee/1.3/api/javax/jms/TopicSession.html>

# Example Applications

---

## Example Application Index

### Basic Features

Directory	Description
common-chat	Simple chat client. (HabariChat.dpr, requires Delphi 2009+)
common-consumertool	Receives messages from broker.
common-delphigui	Sends and receives messages. (GUIDemo.dpr, requires Delphi 2009+)
common-performance	Multi-threaded performance test application.
common-producertool	Sends messages to a broker.

Table 3: Basic Demo Applications

In `hornetq-jms.xml`, define all required destination queues and topics before running a demo.

Example:

```
<queue name="ExampleTopic">
  <entry name="/topic/ExampleTopic"/>
</queue>

<queue name="ExampleQueue">
  <entry name="/queue/ExampleQueue"/>
</queue>
```

In `hornetq-configuration.xml`, additional permissions may be defined for the guest user account. Here is an example for a configuration which grants the right to create and delete durable and non-durable queues to the guest user account.

```
<security-settings>
  <security-setting match="#">
    <permission type="createNonDurableQueue" roles="guest"/>
    <permission type="deleteNonDurableQueue" roles="guest"/>
    <permission type="createDurableQueue" roles="guest"/>
  </security-setting>
</security-settings>
```

```
<permission type="deleteDurableQueue" roles="guest"/>  
<permission type="consume" roles="guest"/>  
<permission type="send" roles="guest"/>  
</security-setting>  
</security-settings>
```

## ConsumerTool

The ConsumerTool demo may be used to receive messages from a queue or topic. This example application is configurable by command line parameters, all are optional.

Parameter	Default Value	Description
<b>AckMode</b>	CLIENT_ACKNOWLEDGE	Acknowledgement mode, possible values are: CLIENT_ACKNOWLEDGE, AUTO_ACKNOWLEDGE or SESSION_TRANSACTED
<b>ClientId</b>		Client Id for durable subscriber
<b>ConsumerName</b>	Habari	name of the message consumer - for durable subscriber
<b>Durable</b>	false	true: use a durable subscriber
<b>MaximumMessages</b>	10	expected number of messages
<b>Password</b>		Password
<b>PauseBeforeShutDown</b>	false	true: wait for key press
<b>ReceiveTimeOut</b>	0	0: asynchronous receive, > 0: consume messages while they continue to be delivered within the given time out
<b>SleepTime</b>	0	time to sleep after asynchronous receive
<b>Subject</b>	TOOL.DEFAULT	queue or topic name
<b>Topic</b>	false	true: topic false: queue
<b>Transacted</b>	false	true: transacted session
<b>URL</b>	localhost	server url
<b>User</b>		user name
<b>Verbose</b>	true	verbose output

Table 4: ConsumerTool Command Line Options

## Requirements

In hornetq-jms.xml, define all required destination queues and topics before running a demo.

Example:

```
<queue name="ExampleTopic">
  <entry name="/topic/ExampleTopic"/>
</queue>

<queue name="ExampleQueue">
  <entry name="/queue/ExampleQueue"/>
</queue>
```

## Examples

Receive 1000 messages from local broker

```
ConsumerTool --MaximumMessages=1000
```

Receive 10 messages from local broker and wait for any key

```
ConsumerTool --PauseBeforeShutDown
```

Use a transacted session to receive 10,000 messages from local broker

```
ConsumerTool --MaximumMessages=10000 --Transacted --AckMode=SESSION_TRANSACTED
```

## ProducerTool

The ProducerTool demo can be used to send messages to the broker. It is configurable by command line parameters, all are optional.

Parameter	Default	Description
<b>MessageCount</b>	10	Number of messages
<b>MessageSize</b>	255	Length of a message in bytes
<b>Persistent</b>	false	Delivery mode 'persistent'
<b>SleepTime</b>	0	Pause between messages in milliseconds
<b>Subject</b>	TOOL.DEFAULT	Destination name
<b>TimeToLive</b>	0	Message expiration time
<b>Topic</b>	false	Destination is a topic
<b>Transacted</b>	false	Use a transaction
<b>URL</b>	localhost	Message broker URL
<b>Verbose</b>	true	Verbose output
<b>User</b>		User name
<b>Password</b>		Password

Table 5: ProducerTool Command Line Options

## Requirements

In hornetq-jms.xml, define all required destination queues and topics before running a demo.

Example:

```
<queue name="ExampleTopic">
  <entry name="/topic/ExampleTopic"/>
</queue>

<queue name="ExampleQueue">
  <entry name="/queue/ExampleQueue"/>
</queue>
```

## Examples

Send 10,000 messages to the queue `TOOL.DEFAULT` on the local broker

```
ProducerTool --MessageCount 10000
```

Send 10 messages to the topic ExampleTopic on the local broker

```
ProducerTool --Topic --Subject=ExampleTopic
```

---

## Performance Test

The performance test application provides a GUI for multi-threaded sending and receiving of messages.

- A broker configuration dialog can be invoked by clicking the URL field
- The communication library (Indy or Synapse) can be selected
- Number and length of messages and thread number can be adjusted using the sliders

For every thread a message queue with the name ExampleQueue.<n> will be used.

## Requirements

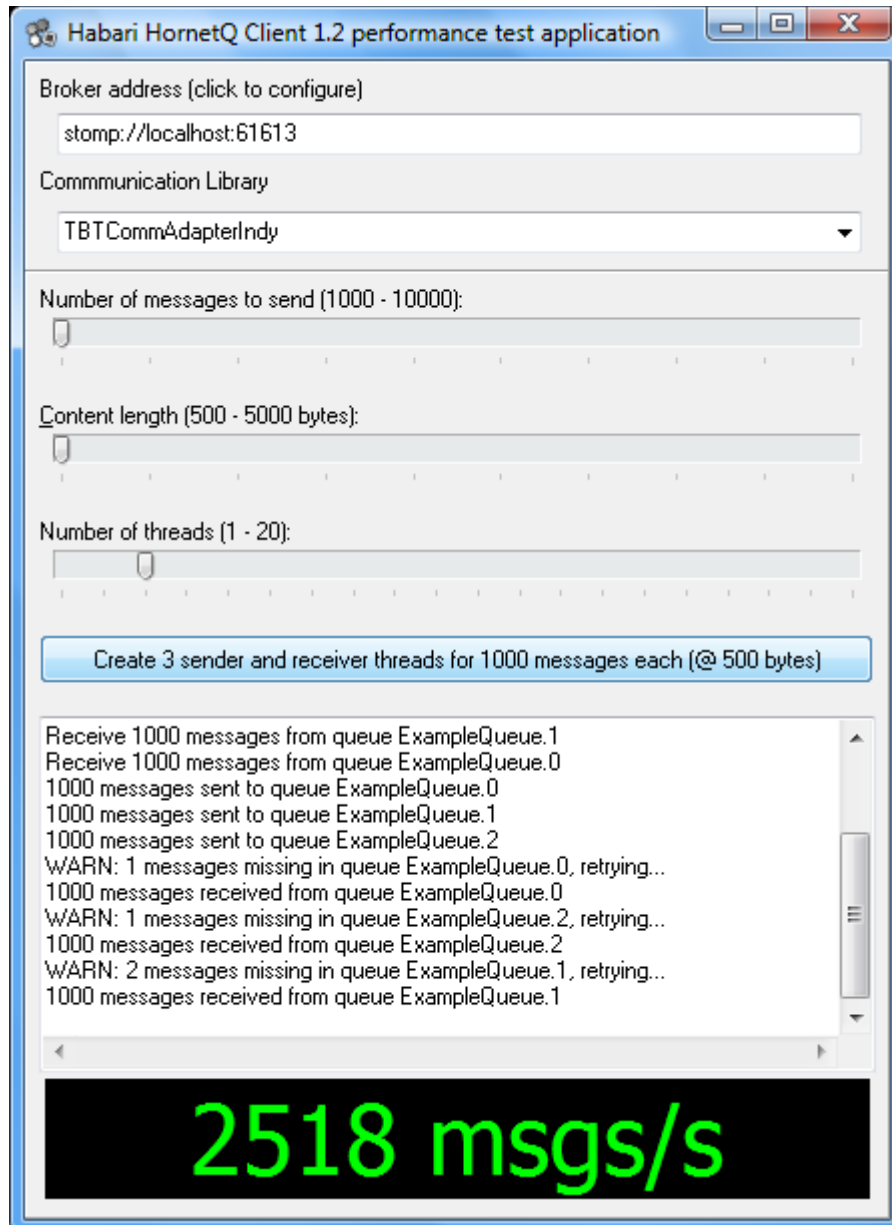
HornetQ does not support dynamic creation of Stomp destinations. So the queues for this demo application need to be added to the configuration.

In hornetq-jms.xml, define a queue for every thread. Every queue name is composed of 'ExampleQueue', a dot and an index, starting at 0.

For example, up to ten threads can be used with this destination definitions:

```
<queue name="ExampleQueue.0">
  <entry name="/queue/ExampleQueue.0"/>
</queue>
<queue name="ExampleQueue.1">
  <entry name="/queue/ExampleQueue.1"/>
</queue>
<queue name="ExampleQueue.2">
  <entry name="/queue/ExampleQueue.2"/>
</queue>
<queue name="ExampleQueue.3">
  <entry name="/queue/ExampleQueue.3"/>
</queue>
<queue name="ExampleQueue.4">
  <entry name="/queue/ExampleQueue.4"/>
</queue>
<queue name="ExampleQueue.5">
  <entry name="/queue/ExampleQueue.5"/>
</queue>
<queue name="ExampleQueue.6">
  <entry name="/queue/ExampleQueue.6"/>
</queue>
<queue name="ExampleQueue.7">
  <entry name="/queue/ExampleQueue.7"/>
</queue>
<queue name="ExampleQueue.8">
  <entry name="/queue/ExampleQueue.8"/>
</queue>
<queue name="ExampleQueue.9">
  <entry name="/queue/ExampleQueue.9"/>
</queue>
```

It is possible to add destinations while the broker is running. HornetQ will detect that the configuration has changed and will try to deploy the new queue.



*Illustration 3: performance demo*

# JBoss AS 5 Integration

---

## Tutorial

This tutorial takes you through some of the basic steps of integrating Delphi applications and JBoss AS 5 with the help of Habari Client for HornetQ.

## Requirements

To complete this tutorial, you need the software and resources listed in the following list.

- **JDK 6**
- **JBoss 5.1.0.GA for JDK 6**
- **JBoss HornetQ 2.1.0**
- **Habari Client for HornetQ ProducerTool and ConsumerTool**
- **Subversion client**

---

## Creating AS 5 Profiles

HornetQ is not shipped by default with the JBoss 5 application server, so you will need to create new AS 5 profiles to run AS 5 with HornetQ.

To create AS 5 profiles:

1. Set the environment property `JBOSS_HOME` to point to the directory where you installed JBoss AS 5
2. run `./build.sh` (or `build.bat` if you are on Windows) in HornetQ's `config/jboss-as-5` directory

This will create 2 new profiles in `$JBOSS_HOME/server`:

- `default-with-hornetq` — it corresponds to AS 5 default profile with HornetQ as its JMS provider. In this profile, HornetQ is *non-clustered*
- `all-with-hornetq` — it corresponds to AS 5 all profile with HornetQ as its JMS provider. In this profile, HornetQ is *clustered*

---

## Broker Configuration

Open the file `$JBOSS_HOME/default-with-hornetq/deploy/hornetq.sar/hornetq-configuration.xml` to enable the Stomp acceptor.

```
<acceptors>
  ...
  <acceptor name="stomp">
    <factory-class>org.hornetq.core.remoting.impl.netty.NettyAcceptorFactory</factory-
class>
    <param key="protocol" value="stomp"/>
    <param key="port" value="61613"/>
  </acceptor>
  ...
</acceptors>
```

Open the file `$JBOSS_HOME/default-with-hornetq/deploy/hornetq.sar/hornetq-jms.xml` to add an example JMS queue.

```
<configuration ...>
  ...
  <queue name="ExampleQueue">
    <entry name="/queue/ExampleQueue"/>
  </queue>
  ...
</configuration>
```

---

## Start JBoss

You can now start JBoss AS 5 using the `default-with-hornetq` profile:

- `$JBOSS_HOME/bin/run.sh -c default-with-hornetq`

or (on Windows)

- `$JBOSS_HOME/bin/run.bat -c default-with-hornetq`

With `netstat /a` you can verify if the Stomp acceptor is running on port 61613.

---

## Running the Delphi Test

Now you are ready to produce and consume messages with Habari Client for HornetQ.

1. start ProducerTool application to send messages to the queue ExampleQueue
2. start the ConsumerTool application to receive the messages

# JBoss AS 7.1 integration

This chapter is still under construction.

If you have a support contract please contact Habarisoft for configuration information.

# Management API

---

## Management Notifications

A useful feature of HornetQ are management notifications. HornetQ servers emit management notifications when events of interest occur (consumers are created or closed, destinations are created or deleted, security authentication fails, etc.).

These notifications can be received either by using [JMX](#) or by receiving [JMS](#) Messages from a well-known destination.

### Example

In the broker configuration file **hornetq-configuration.xml**, a JMS topic name can be set for the management name:

```
<management-notification-address>
  jms.topic.notificationsTopic
</management-notification-address>
```

The Delphi code below creates a connection and subscribes to the topic. Next, it uses a loop to receive the notification messages:

```
Connection := TBTJMSConnection.MakeConnection;
Connection.Start;
Session := Connection.CreateSession(False, amAutoAcknowledge);
Destination := Session.CreateTopic('notificationsTopic');
Consumer := Session.CreateConsumer(Destination);

while True do
begin
  Reply := Consumer.Receive;
  PropNames := Reply.GetPropertyNames;
  for I := 0 to Length(PropNames) - 1 do
  begin
    PropName := PropNames[I];
    WriteLn(Format(' %s: %s', [PropName, Reply.GetStringProperty(PropName)]));
  end;
end;
Connection.Close;
```

# Message Options

---

## JMS Standard Properties

### API Documentation

JMS Standard properties are documented in more detail in the API documentation for the `TBTMessage` class. They are based on the JMS specification of the `Message` interface.<sup>5</sup>

### JMS properties for outgoing messages

Messages sent by Habari Client for HornetQ can set these JMS standard properties:

<b>JMSCorrelationID</b>	The correlation ID for the message.
<b>JMSExpiration</b>	The message's expiration value.
<b>JMSDeliveryMode</b>	Whether or not the message is persistent.
<b>JMSPriority</b>	The message priority level.
<b>JMSReplyTo</b>	The Destination object to which a reply to this message should be sent.

### JMS properties for incoming messages

Messages received by Habari Client for HornetQ may contain these JMS standard properties:

<b>JMSCorrelationID</b>	The correlation ID for the message.
<b>JMSExpiration</b>	The message's expiration value.
<b>JMSDeliveryMode</b>	Whether or not the message is persistent.
<b>JMSPriority</b>	The message priority level.
<b>JMSTimestamp</b>	The timestamp the broker added to the message.
<b>JMSMessageId</b>	The message ID which is set by the provider.
<b>JMSReplyTo</b>	The Destination object to which a reply to this message should be sent.

---

<sup>5</sup> <http://download.oracle.com/javaee/5/api/javax/jms/Message.html>

# Conditional Symbols

---

## HABARI\_LOGGING

This conditional symbol enables logging.

Logging requires the open source logging framework for Delphi Log4D.

Log4D is available on Sourceforge at

<http://log4d.sourceforge.net/>

---

## HABARI\_RAW\_TRACE

In unit BTStompFrame. Enables detailed logging of Stomp message frames. If this symbol is defined, a compiler warning will be emitted:

```
Compiled with HABARI_RAW_TRACE
```

---

## HABARI\_STOMP\_11

Enables experimental support for Stomp 1.1 - STOMP 1.1 is not yet supported by Habari Client for OpenMQ, defining the HABARI\_USE\_STOMP11 symbol will cause a compiler error

---

## HABARI\_USE\_RTTI

By default extended RTTI in Delphi 2010 and newer will be disabled by new compiler directives in every source unit, it can be enabled with the symbol HABARI\_USE\_RTTI

# Known Limitations

---

## Broker Limitations

Some of the documented limitations of the Stomp protocol implementation are listed below with their fix version:

Open	<a href="#">Support temporary destinations in Stomp implementation</a>
Fixed in 2.2.0.CR1	<a href="#">STOMP connection does not support TTL</a>
Fixed in 2.2.0.CR1	<a href="#">Change Stomp default message priority to 4 instead of 0</a>
Fixed in 2.1.0.CR1	<a href="#">Stomp expiration header is not supported</a>

## Acknowledgements are not transactional

The HornetQ Stomp documentation states<sup>6</sup>

“Message acknowledgements are not transactional. The ACK frame can not be part of a transaction (it will be ignored if its `transaction` header is set).”

---

## Sessions

### Acknowledgment Modes

Acknowledgment mode “**amDupsOkAcknowledge**” is unsupported.

---

## Messages

### Message Property Data Types

The Stomp protocol uses `string` type key/value lists for the representation of message properties. Regardless of the method used to set message properties, all message properties will be interpreted as Java Strings by the Message Broker.

---

<sup>6</sup> <http://docs.jboss.org/hornetq/2.2.2.Final/user-manual/en/html/interoperability.html#stomp>

Time stamp properties are converted to an Unix time stamp value, which is the internal representation in Java. But still, these values can not be used with date type expressions.

---

## Destinations

### Durable Subscriptions

Removing durable subscriptions is not implemented in the Stomp acceptor of HornetQ.

---

## JMSXGroupID

This message property is not implemented

# References

## Message Broker

HornetQ <http://hornetq.org/>

## IDE

Embarcadero Delphi <http://www.embarcadero.com/delphi>

Lazarus <http://www.lazarus.freepascal.org>

## JMS

JMS Spec (PDF) <http://www.oracle.com/technetwork/java/jms/index.html>

## JSON

SuperObject <http://www.progdigy.com/>

## Stomp

Project home <http://stomp.github.com/>

## Communication Libraries

Synapse <http://www.synapse.ararat.cz>

Indy 10 <http://www.indyproject.org>

Indy 10 Snapshot <http://indy.fulgan.com/ZIP>

# Habari Client for HornetQ License

Habari Client for HornetQ (c) 2010-2012 Michael Justin - Habarisoft

This copyright applies to all source code, compiled code, documentation, graphics and auxiliary files, except those parts written by other people (which are normally copyright their authors).

## **GENERAL TERMS THAT APPLY TO COMPILED PROGRAMS AND REDISTRIBUTABLES**

You may write and compile your own application programs using the library. You may reproduce and distribute, in executable form only, programs which you create using the library without additional license or fees, subject to all of the conditions in this statement.

The license granted in this statement for you to create your own compiled programs and distribute your programs and the Redistributables (if any) is subject to all of the following conditions: (i) all copies of the programs you create must bear a valid copyright notice, either your own or the Habarisoft copyright notice that appears on the Software; (ii) you may not remove or alter any Habarisoft copyright, trademark or other proprietary rights notice contained in any portion of Habarisoft libraries, source code, Redistributables or other files that bear such a notice; (iii) Habarisoft provides no warranty at all to any person, other than the Limited Warranty provided to the original purchaser of the Software, and you will remain solely responsible to anyone receiving your programs for support, service, upgrades, or technical or other assistance, and such recipients will have no right to contact Habarisoft for such services or assistance; (iv) you will indemnify and hold Habarisoft, its related companies and its suppliers, harmless from and against any claims or liabilities arising out of the use, reproduction or distribution of your programs; (v) your programs must be written using a licensed, registered copy of the Software; (vi) your programs must add primary and substantial functionality,

and may not be merely a set or subset of any of the libraries (including runtime libraries), code, Redistributables or other files of the Software; (vii) regardless of any modifications which you make and regardless of how you might compile, link, or package your programs, the libraries (including runtime libraries), code, Redistributables, and/or other files of the Software (including any portions thereof) may not be used in programs created by your end users (i.e., users of your programs) and may not be further redistributed by your end users; and (viii) you may not use Habarisoft's or any of its suppliers' names, logos, or trademarks to market your programs, except to state that your program was written using the Software.

All Habarisoft libraries, source code, Redistributables and other files remain Habarisoft's exclusive property. Regardless of any modifications that you make, you may not distribute any files (particularly Habarisoft source code and other non-executable files).

### **LIMITED WARRANTY**

No warranty of any sort, expressed or implied, is provided in connection with the library, including, but not limited to, implied warranties of merchantability or fitness for a particular purpose. Any cost, loss or damage of any sort incurred owing to the malfunction or misuse of the library or the inaccuracy of the documentation or connected with the library in any other way whatsoever is solely the responsibility of the person who incurred the cost, loss or damage. Furthermore, any illegal use of the library is solely the responsibility of the person committing the illegal act. By using this program you accept these responsibilities, and give up any right to seek any damages against the authors in connection with this program.

# Third Party Library Licenses

## Synapse

The following software may be included in this product: Ararat Synapse; Use of any of this software is governed by the terms of the license below:

```
| Copyright (c)1999-2008, Lukas Gebauer |
| All rights reserved. |
| |
| Redistribution and use in source and binary forms, with or without |
| modification, are permitted provided that the following conditions are met: |
| |
| Redistributions of source code must retain the above copyright notice, this |
| list of conditions and the following disclaimer. |
| |
| Redistributions in binary form must reproduce the above copyright notice, |
| this list of conditions and the following disclaimer in the documentation |
| and/or other materials provided with the distribution. |
| |
| Neither the name of Lukas Gebauer nor the names of its contributors may |
| be used to endorse or promote products derived from this software without |
| specific prior written permission. |
| |
| THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" |
| AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE |
| IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE |
| ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR |
| ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL |
| DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR |
| SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER |
| CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT |
| LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY |
| OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH |
| DAMAGE. |
| ===== |
| The Initial Developer of the Original Code is Lukas Gebauer (Czech Republic). |
| Portions created by Lukas Gebauer are Copyright (c)1999-2008. |
| All Rights Reserved. |
```

## Indy BSD License

### Copyright

Portions of this software are Copyright (c) 1993 - 2003, Chad Z. Hower (Kudzu) and the Indy Pit Crew - <http://www.IndyProject.org/>

### License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation, about box and/or other materials provided with the distribution.
- No personal names or organizations names associated with the Indy project may be used to endorse or promote products derived from this software without specific prior written permission of the specific individual or organization.

THIS SOFTWARE IS PROVIDED BY Chad Z. Hower (Kudzu) and the Indy Pit Crew "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## SuperObject

```
*                               Super Object Toolkit
*
* Usage allowed under the restrictions of the Lesser GNU General Public License
* or alternatively the restrictions of the Mozilla Public License 1.1
*
* Software distributed under the License is distributed on an "AS IS" basis,
* WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for
* the specific language governing rights and limitations under the License.
*
* Unit owner : Henri Gourvest <hgourvest@gmail.com>
* Web site   : http://www.progdigy.com
*
```

```
* This unit is inspired from the json c lib:  
* Michael Clark <michael@metaparadigm.com>  
* http://oss.metaparadigm.com/json-c/
```

---

## Log4D

```
The contents of this file are subject to the Mozilla Public  
License Version 1.1 (the "License"); you may not use this file  
except in compliance with the License. You may obtain a copy of  
the License at http://www.mozilla.org/MPL/MPL-1.1.html
```

```
Software distributed under the License is distributed on an "AS  
IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or  
implied. See the License for the specific language governing  
rights and limitations under the License.
```

---

## NativeXml

Copyright (c) 2003 - 2011 Simdesign BV. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY SIMDESIGN BV "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL SIMDESIGN BV OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# Release Notes

---

## Version 1.7

Released May 15, 2012

### New

**RTTI suppression** By default extended RTTI in Delphi 2010 and newer will be disabled by new compiler directives in every source unit, it can be enabled with the symbol `HABARI_USE_RTTI`

**HornetQ.2.2.14** Tested with HornetQ.2.2.14.Final and with HornetQ 2.2.12 which is included in JBoss AS 7.1.1.Final

### Changed

**ReceiveNoWait** ReceiveNoWait/ReadOneMessageNoWait has been improved both in the Indy and Synapse communication adapter

**Transaction ID** Transactions now use a GUID as identifier to make them unique across connections

**Chat demo** The chat demo uses a connection configuration dialog, some user interface improvements

**VisualMM demo** The VisualMM demo allows to choose from a list of running processes

**Throughput test** The utility displays more statistics about message count and transfer speed

**Indy rev. 4736** Tested with revision 4736 of Indy 10.5.8

**Shared code** Core library source code is shared between Habari Client libraries – some parts of the source code might be included which are not supported by the HornetQ message broker or the current version of the client library (for example, STOMP 1.1 is not yet supported by Habari Client for HornetQ, defining the `HABARI_USE_STOMP11` symbol will cause a compiler error)

---

## Version 1.6

Released January 17, 2012

### New

**VisualMM**

A simple web server written with Indy (source included) which collects current FastMM usage statistics about its own process, sends them to a broker queue and consumes the data to generate usage charts

### Changed

**Indy rev. 4733**

Tested with revision 4733 of Indy 10.5.8

**FPC 2.6.0**

Compiles with Free Pascal 2.6.0

**HornetQ.2.2.10**

Tested with HornetQ 2.2.10 (AS 7 development branch) – this version consumes up to 30,000 messages per second, but does not throttle the producer to prevent resource exhaustion, until this is fixed in the broker, client apps should reduce producer rates (see throughput test demo)

**Frame Decoder**

The new Stomp Frame decoder implementation is always used, the conditional symbol HABARI\_USE\_TBYTES has been removed

**Fixes**

Improvements in the Indy and Synapse communication adapter, and warnings in object message transformers

**Throughput Demo**

The throughput tool displays more information about message transfer rates

**GUI Demo**

The GUI demo allows to save the file which is in an incoming message

**Packages**

Removed packages for unsupported demo component HabariExpress

---

## Version 1.5

Released September 20, 2011

### New

**Delphi XE2**

Compilation tested with Win32, Win64 and OSX (new versions of DUnit, Synapse and Indy, and NativeXML are required)

**Throughput Demo**

The throughput tool continuously produces and consumes messages to monitor the average message throughput

**common-tests** New unit test suites, shared with all Habari JMS Client libraries

## Changed

**DUnit rev. 41** Upgraded for XE2 support (please note that the current trunk version of DUnit no longer supports older Delphi versions, at least Delphi 2007 is required)

**Synapse rev. 144** Upgraded for XE2 support (included)

**NativeXML v. 4.01** Upgraded for XE2 support (included)

**Indy rev. 4676** Tested with revision 4676 of Indy 10.5.8

**Unicode Tests** DUnit tests now include Unicode in object message exchange (this revealed a problem with Delphi 6 so it is recommended to use Delphi 2009 or newer for object message exchange)

**Frame Decoder** The new Stomp Frame decoder implementation is now also available for the Synapse communication adapter, it can be enabled with the conditional symbol `HABARI_USE_TBYTES`

---

## Version 1.4

Released July 4, 2011

## New

**NativeXml** Support for ObjectMessage messages type with another implementation based on NativeXml

**MapMessage** Support for MapMessage message type, implementations based on NativeXml and OmniXML

**Transformer** A helper method, SetTransformer (in unit BTJMSConnection), can be used to set a IMessageTransformer on a IConnection

**Unit Tests** New unit ObjectExchangeTests for IObjectMessage and IMapMessage object exchange tests

**Frame Decoder** The new unit BTStompDecoder contains a new Stomp Frame decoder implementation, it can be enabled with the conditional symbol `HABARI_USE_TBYTES`

**FPC 2.4.4** Tested with Free Pascal 2.4.4

**HornetQ 2.2.5** Tested with HornetQ 2.2.5.Final

## Changed

**Name** The library name changed to Habari Client for HornetQ

---

<b>TransformationId</b>	The client verifies if the transformation id of the message transformer matches the 'transformation' header of incoming object messages
<b>Performance Demo</b>	Shared code with Habari ActiveMQ Client and Habari OpenMQ Client, display transfer speed in msgs/s
<b>TTransformable</b>	Class TTransformable is deprecated

---

## Version 1.3

Released May 8, 2011

### Fixed

<b>Content Type</b>	Message type detection (TextMessage or BytesMessage) is now based on the presence of a content-length message header instead of a message-type message header. This change is necessary to ensure compatibility with JMS message types on the Java side.
<b>HABARI_STOMP11</b>	A new compiler constant HABARI_STOMP11 activates code which will support Stomp 1.1 in a future release.

---

## Version 1.2

Released April 4, 2011

### New

<b>Failover transport</b>	The Failover transport layers reconnect logic on top of the Stomp transport. The URL for a connection factory can be configured with failover:(uri1,...,uriN)?transportOptions
<b>HornetQ</b>	Tested with HornetQ version 2.1.2.Final and 2.2.2.Final
<b>CreateMessage</b>	Function Session#CreateMessage returns a IMessage object
<b>IConnectionInfo</b>	New interface in BtMgmtInterfaces which provides connection state information

### Changed

<b>FPC 2.4.2</b>	Build tested with Free Pascal 2.4.2
<b>Performance Demo</b>	Shared code with Habari ActiveMQ Client and Habari OpenMQ Client, display transfer speed in msgs/s

---

---

<b>GUI demo fix</b>	Fixed an AV which occurred when the demo program was compiled without assertions
<b>Stomp constants</b>	Stomp header constants are defined in unit BTStompTypes
<b>Documentation</b>	Fixes in HornetQ 2.2: issue 560 (stomp default priority), issue 526 (TTL support), added notes about broker limitations
<b>Single Source</b>	All versions of the Habari JMS Client library share the source code for the basic demo applications ConsumerTool, ProducerTool, DelphiGUI and HabariChat
<b>Indy Adapter</b>	Method TBTCCommAdapterIndy.ReadOneMessageNoWait now uses ReadOneMessage(1) to return faster to the calling routine - old code is still included in comments
<b>ReadMessageBuffer</b>	The internal method ReadMessageBuffer in TBTCCommAdapterIndy has a new parameter, ATimeout
<b>Refactorings</b>	Refactored to new units BTSerialIntf, BTSessionIntf

## Fixed

<b>Thread</b>	Fixed compiler warning about deprecated Thread methods Resume and Suspend
<b>Connection</b>	Fixed code to avoid a EIdConnClosedGracefully exception in BTStompCustomClient

---

## Version 1.1

Released October 14, 2010

## New

<b>Delphi XE</b>	Ready for Delphi XE
<b>HornetQ 2.1.2</b>	Tested with version 2.1.2.Final
<b>Indy 10.5.8</b>	Tested with Indy 10.5.8
<b>Log4D library</b>	The Log4D logging library is now included and used when HABARI_LOGGING is defined

## Changed

<b>doxygen</b>	Updated to doxygen 1.7.1
<b>Documentation</b>	Documentation updates

## **Version 1.0**

Released June 5, 2010

# Index

## Reference

Binary Message.....	26	JMSCorrelationID.....	45
BTJMSConnection.....	28	JMSDeliveryMode.....	45
BTStreamHelper.....	26	JMSExpiration.....	45
Connection.....	15	JMSMessageId.....	45
connection factory.....	15	JMSPriority.....	45
ConsumerTool.....	35	JMSReplyTo.....	45
createDurableQueue.....	33	JMSTimestamp.....	45
CreateDurableSubscriber.....	32	JSON.....	49
CreateMessage.....	58	LoadBytesFromStream.....	26
createNonDurableQueue.....	33	Log4D.....	46, 54
CreateObjectMessage.....	28	Management Notifications.....	44
deleteDurableQueue.....	34	Map Message.....	30
deleteNonDurableQueue.....	33	Message Consumer.....	21
Destination.....	19	Message Producer.....	21
ExampleQueue.....	33	MessageTransformer.....	28
ExampleTopic.....	33	NativeXml.....	27, 54
Failover Support.....	17	Notifications.....	44
HABARI_LOGGING.....	46	Object Message.....	27
HABARI_RAW_TRACE.....	46	OmniXML.....	27
HABARI_STOMP_11.....	46	OnMessage.....	24
HABARI_USE_RTTI.....	46	point-to-point.....	19
HornetQ.....	<b>49</b>	ProducerTool.....	37
hornetq-configuration.xml.....	33	publish and subscribe.....	19
hornetq-jms.xml.....	33	Queue.....	19
IConnection.....	15	Receive.....	25
IDestination.....	<b>24</b>	ReceiveNoWait.....	25
IMessage.....	<b>24</b>	SamplePojo.....	28
IMessageConsumer.....	<b>24</b>	Session.....	16
IMessageListener.....	<b>24</b>	SetTransformer.....	28
IMessageProducer.....	28	Stomp.....	49
Indy.....	7, 49	SuperObject.....	27, 49, 53
Internet Direct (Indy).....	<b>8</b>	Synapse.....	7, <b>8</b> , 49
ISession.....	28	Text Message.....	23
JBoss.....	7, 41	Topic.....	20
JMS.....	49	TopicSubscriber.....	32

## Table Index

Table 1: Failover transport options.....	18
--	----

---

Table 2: Message Transformer Implementations.....	27
Table 3: Basic Demo Applications.....	33
Table 4: ConsumerTool Command Line Options.....	35
Table 5: ProducerTool Command Line Options.....	37

## Illustration Index

Illustration 1: HornetQ log.....	11
Illustration 2: JMX console.....	11
Illustration 3: performance demo.....	40