



**habarisoft**  
Enterprise Messaging Software for Delphi®

# Getting started with Habari Client for OpenMQ

## *Version 2.1*

---

### **Trademarks**

Habari is a registered trademark of Michael Justin and is protected by the laws of Germany and other countries. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Embarcadero, the Embarcadero Technologies logos and all other Embarcadero Technologies product or service names are trademarks, service marks, and/or registered trademarks of Embarcadero Technologies, Inc. and are protected by the laws of the United States and other countries. 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 OpenMQ</b> .....	<b>4</b>
<b>Quick Start</b> .....	<b>7</b>
<b>Download and Installation</b> .....	<b>7</b>
<b>GlassFish v3 configuration</b> .....	<b>8</b>
<b>Dependencies</b> .....	<b>10</b>
<b>Requirements</b> .....	<b>10</b>
<b>TCP/IP Communication Libraries</b> .....	<b>11</b>
<b>Communication Adapter Configuration</b> .....	<b>12</b>
<b>Introduction</b> .....	<b>12</b>
<b>The JMS API Programming Model</b> .....	<b>13</b>
<b>Tutorials</b> .....	<b>14</b>
<b>Habari Quick Start Tutorial</b> .....	<b>14</b>
<b>Online Tutorials</b> .....	<b>16</b>
<b>Connections and Sessions</b> .....	<b>17</b>
<b>Step by Step Example</b> .....	<b>17</b>
<b>Transacted Sessions</b> .....	<b>19</b>
<b>Failover Support</b> .....	<b>19</b>
<b>Destinations</b> .....	<b>21</b>
<b>Introduction</b> .....	<b>21</b>
<b>Create a new Destination</b> .....	<b>21</b>
<b>Producer and Consumer</b> .....	<b>23</b>
<b>Message Producer</b> .....	<b>23</b>
<b>Message Consumer</b> .....	<b>23</b>
<b>JMS Selectors</b> .....	<b>24</b>
<b>Text Messages</b> .....	<b>25</b>
<b>Sending Text Messages</b> .....	<b>25</b>
<b>Receive Text Messages</b> .....	<b>26</b>
<b>Bytes Messages</b> .....	<b>28</b>
<b>Creation</b> .....	<b>28</b>
<b>Sending</b> .....	<b>28</b>
<b>Object Messages</b> .....	<b>29</b>
<b>Introduction</b> .....	<b>29</b>
<b>Message Transformers in Habari Client for OpenMQ</b> .....	<b>29</b>
<b>Durable Subscriptions</b> .....	<b>32</b>
<b>Description</b> .....	<b>32</b>
<b>Example Application Index</b> .....	<b>33</b>
<b>ConsumerTool</b> .....	<b>33</b>
<b>ProducerTool</b> .....	<b>34</b>
<b>Performance Test</b> .....	<b>35</b>
<b>Message Options</b> .....	<b>36</b>
<b>JMS Standard Properties</b> .....	<b>36</b>
<b>User Defined Properties</b> .....	<b>37</b>
<b>Useful Units</b> .....	<b>38</b>
<b>BTStreamHelper</b> .....	<b>38</b>
<b>BTJavaPlatform</b> .....	<b>38</b>
<b>Conditional Symbols</b> .....	<b>39</b>
<b>HABARI_LOGGING</b> .....	<b>39</b>
<b>HABARI_RAW_TRACE</b> .....	<b>39</b>
<b>HABARI_USE_RTTI</b> .....	<b>39</b>
<b>Known Limitations</b> .....	<b>40</b>
<b>Sessions</b> .....	<b>40</b>
<b>Messages</b> .....	<b>40</b>

<b>Security</b> .....	<b>41</b>
<b>References</b> .....	<b>42</b>
<b>Habari Client for OpenMQ License</b> .....	<b>43</b>
<b>Third Party Library Licenses</b> .....	<b>45</b>
<b>Synapse</b> .....	<b>45</b>
<b>Indy BSD License</b> .....	<b>45</b>
<b>IkJSON</b> .....	<b>46</b>
<b>SuperObject</b> .....	<b>46</b>
<b>Log4D</b> .....	<b>47</b>
<b>NativeXml</b> .....	<b>47</b>
<b>Release Notes</b> .....	<b>48</b>
<b>Version 2.1</b> .....	<b>48</b>
<b>Version 2.0</b> .....	<b>48</b>
<b>Version 1.9</b> .....	<b>49</b>
<b>Version 1.8</b> .....	<b>50</b>
<b>Version 1.7</b> .....	<b>51</b>
<b>Version 1.6</b> .....	<b>51</b>
<b>Version 1.5</b> .....	<b>52</b>
<b>Version 1.4</b> .....	<b>52</b>
<b>Version 1.3</b> .....	<b>53</b>
<b>Version 1.2</b> .....	<b>53</b>
<b>Version 1.1</b> .....	<b>54</b>
<b>Version 1.0</b> .....	<b>54</b>
<b>Index</b> .....	<b>55</b>

## Introduction

---

### About Habari® Client for OpenMQ

Habari Client for OpenMQ is a Delphi library for OpenMQ. With Habari Client for OpenMQ, Delphi developers can build integrated solutions, connecting cross language clients and protocols from C, Delphi, and Java using the peer-to-peer or the publish and subscribe communication model. The library uses a plug-in style architecture for communication libraries. It supports OpenMQ version 4.4 and 4.5, Delphi 2009 to XE2 and Free Pascal, and follows the specification of the JMS API.

### How Can I Use It?

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

- **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.
- **Application Server Integration:** Open Message Queue is a key component of the GlassFish v2.1.1 and GlassFish v3 Application Server.
- **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.
- **Inter-process 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

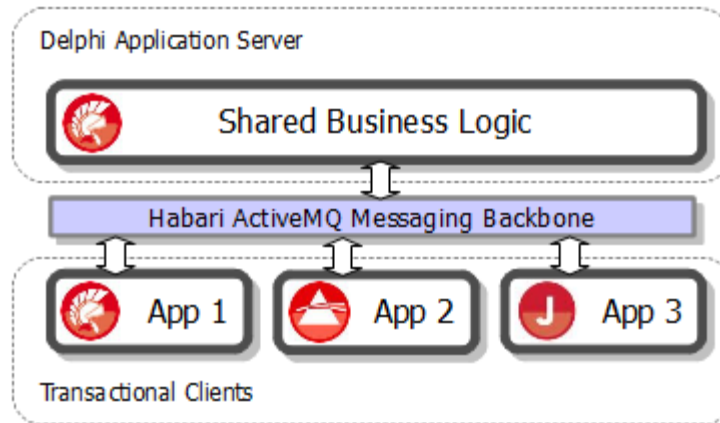


Illustration 1: Shared Business Logic

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

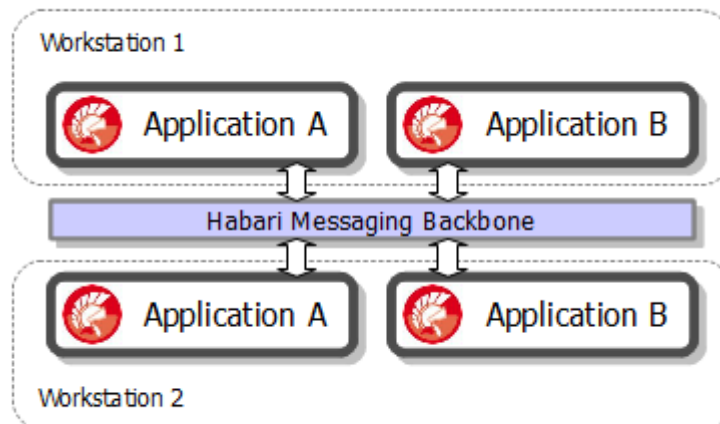
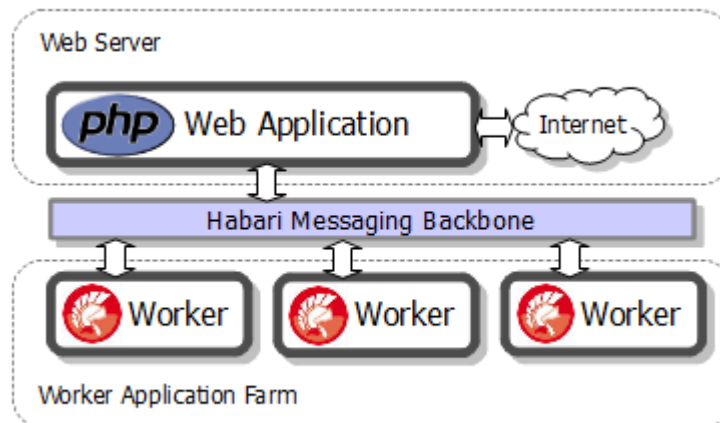


Illustration 2: Peer to Peer Communication

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



*Illustration 3: Load Balancing*

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.

## Quick Start

---

### Download and Installation

The OpenMQ 4.4 web page is located at <http://mq.java.net/4.4.html>.

The OpenMQ 4.5 web page is located at <http://mq.java.net/4.5.html>.

Note that this installation documentation uses the Windows installer, however the Habari Client for OpenMQ library would work with installations of OpenMQ on other operating system platforms as well.

Installation steps for OpenMQ 4.5:

1. download the binary image file with installer (for Microsoft Windows x86) from the OpenMQ web page
2. unpack the installer
3. navigate to the directory `openmq4_5-installer`
4. start the installer script `installer.vbs`

When the installer application is started, a graphical application will display allow you to install and configure Message Queue.

### Start the Broker

You are ready to start the OpenMQ server now via the `mq/bin/imqbrokerd` command, which will run the broker and display log messages in a window

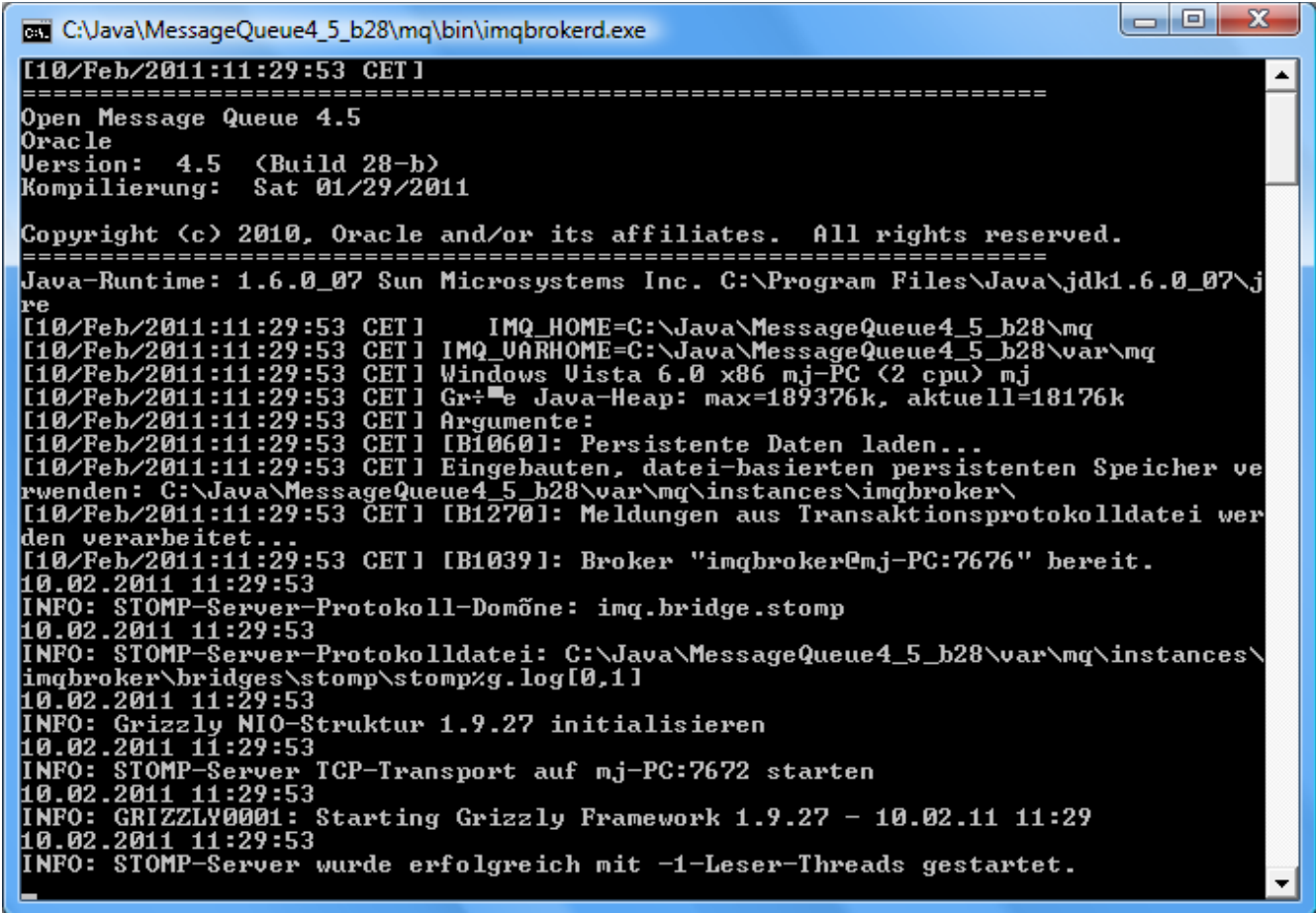
### Configuration

Now you will need to enable Stomp support in the broker and configure a user name and password for the broker bridge.

1. stop the broker
2. navigate to the configuration folder `.../instances/<instance name>/props` (this folder will be created when the broker started), for example `C:\MessageQueue-4.5\var\mq\instances\imqbroker\props`
3. open the configuration file `config.properties` with a text editor
4. add the following lines to configure the Stomp adapter with a test admin account:

```
imq.bridge.admin.user=admin
imq.bridge.admin.password=admin
imq.bridge.activelist=stomp
imq.bridge.enabled=true
```

Save the file. You are ready to start the broker now with Stomp support.



```

C:\Java\MessageQueue4_5_b28\mq\bin\imqbrokerd.exe
[10/Feb/2011:11:29:53 CET]
=====
Open Message Queue 4.5
Oracle
Version: 4.5 <Build 28-b>
Kompilierung: Sat 01/29/2011

Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
=====
Java-Runtime: 1.6.0_07 Sun Microsystems Inc. C:\Program Files\Java\jdk1.6.0_07\j
re
[10/Feb/2011:11:29:53 CET] IMQ_HOME=C:\Java\MessageQueue4_5_b28\mq
[10/Feb/2011:11:29:53 CET] IMQ_VARHOME=C:\Java\MessageQueue4_5_b28\var\mq
[10/Feb/2011:11:29:53 CET] Windows Vista 6.0 x86 mj-PC (2 cpu) mj
[10/Feb/2011:11:29:53 CET] Größte Java-Heap: max=189376k, aktuell=18176k
[10/Feb/2011:11:29:53 CET] Argumente:
[10/Feb/2011:11:29:53 CET] [B1060]: Persistente Daten laden...
[10/Feb/2011:11:29:53 CET] Eingebauten, datei-basierten persistenten Speicher ve
rwenden: C:\Java\MessageQueue4_5_b28\var\mq\instances\imqbroker\
[10/Feb/2011:11:29:53 CET] [B1270]: Meldungen aus Transaktionsprotokolldatei wer
den verarbeitet...
[10/Feb/2011:11:29:53 CET] [B1039]: Broker "imqbroker@mj-PC:7676" bereit.
10.02.2011 11:29:53
INFO: STOMP-Server-Protokoll-Domäne: imq.bridge.stomp
10.02.2011 11:29:53
INFO: STOMP-Server-Protokolldatei: C:\Java\MessageQueue4_5_b28\var\mq\instances\
imqbroker\bridges\stomp\stomp%g.log[0,1]
10.02.2011 11:29:53
INFO: Grizzly NIO-Struktur 1.9.27 initialisieren
10.02.2011 11:29:53
INFO: STOMP-Server TCP-Transport auf mj-PC:7672 starten
10.02.2011 11:29:53
INFO: GRIZZLY0001: Starting Grizzly Framework 1.9.27 - 10.02.11 11:29
10.02.2011 11:29:53
INFO: STOMP-Server wurde erfolgreich mit -1-Leser-Threads gestartet.

```

As you can see, the Stomp bridge is running and using TCP port 7672. The Habari Client for OpenMQ library will use port 7672 by default.

## Test the Stomp connection

To test the Stomp connection, you can use the ProducerTool demo application in the directory <Habari>\demo\producertool.

If you start the ProducerTool without command line parameters, it will send 10 messages to the broker. For a description of the parameters, please see chapter "ProducerTool" (p. 34).

## Adding user accounts

The command line tool imqusermgr can be used for user administration. Example:

```
imqusermgr add -u name -p pass
```

## GlassFish v3 configuration

In the default installation of GlassFish v3, OpenMQ is not started automatically when the broker starts, so the message broker configuration file for the default domain "domain1"

(`glassfish/domain1/imq/instances/imqbroker/props/config.properties` for example) is not present yet.

Follow these steps to initialize OpenMQ in GlassFish v3:

- **only for GlassFish v3.0** (no longer required in GlassFish v3.1): edit the domain configuration file `domain.xml` in the config folder of the default domain `glassfish\domains\domain1\config` to deactivate lazy initialization:

```
<jms-service default-jms-host="default_JMS_host" type="EMBEDDED">
  <jms-host host="localhost" name="default_JMS_host" lazy-init="false" />
</jms-service>
```

- start GlassFish
- run `asadmin jms-ping`
- edit the `config.properties` file to activate Stomp
- restart GlassFish

# Dependencies

---

## Requirements

### Development Environment

- Embarcadero Delphi 2009 or higher
- Free Pascal

### Message Broker

- Open Message Queue 4.4 or 4.5
- Java Runtime Environment 1.6

### 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>

Inofficial 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 39

# 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 run time.

## 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 run time by setting the adapter class (either by its name or by its type).

## Available Communication Adapters

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

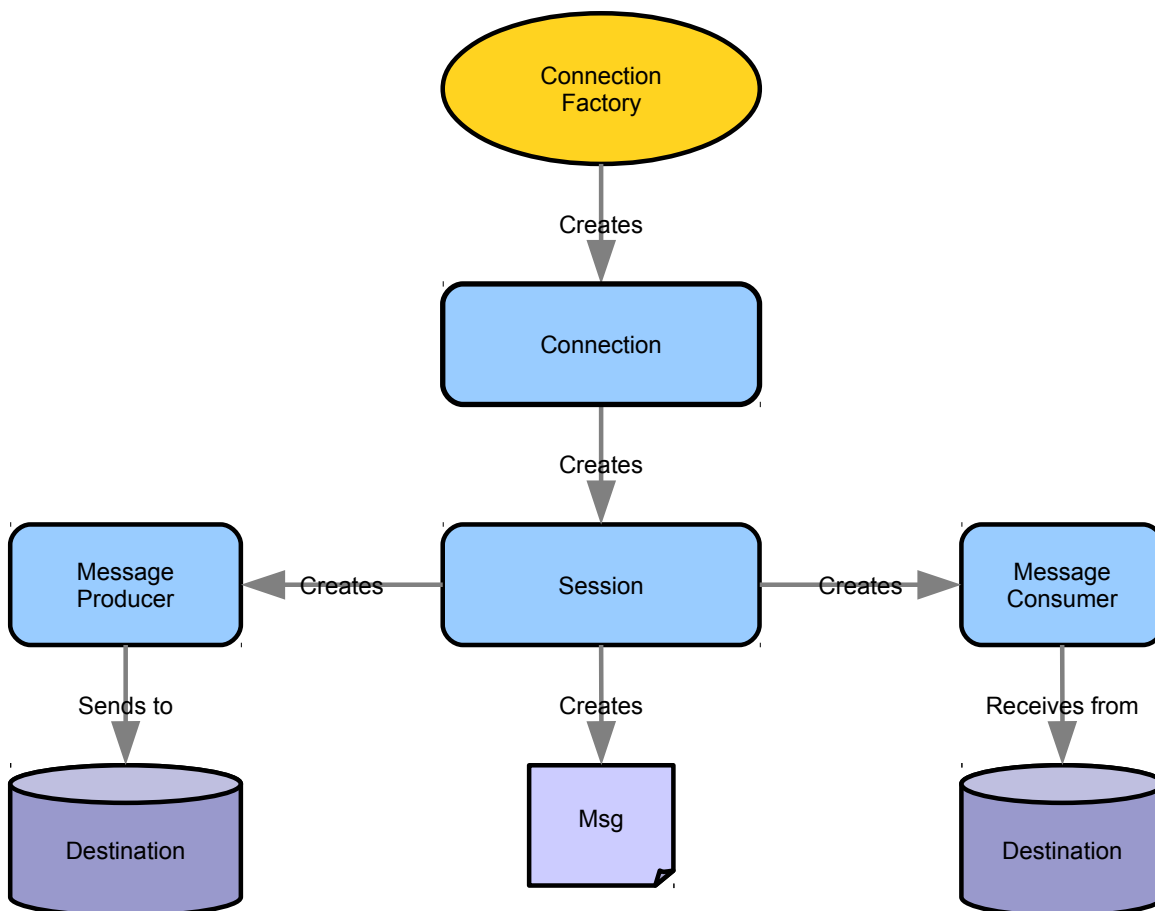
## Overview: Adapter classes and units

Adapter	Unit name
TBTCommAdapterIndy	BTCommAdapterIndy
TBTCommAdapterSynapse	BTCommAdapterSynapse
TBTCommAdapterIndySSL (beta)	BTCommAdapterIndySSL

## The JMS API Programming Model

The Sun online documentation contains a description of the JMS API Programming model:

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



The JMS API Programming Model: Overview

## Tutorials

---

### Habari Quick Start Tutorial

This tutorial provides a very simple and quick introduction to the Habari client library by walking you through the creation of a simple "Hello World" application. Once you are done with this tutorial, you will have a general knowledge of how to create and run Habari applications.

This tutorial takes less than 10 minutes to complete.

**To complete this tutorial, you need the following software and resources:**

<u>Software or Resource</u>	<u>Version required</u>
Delphi	2009 <sup>1</sup>
Synapse	Revision 39
OpenMQ	Version 4.4

### Setting up the project

To create a new project:

1. Start the Delphi IDE.
2. In the IDE, choose File > New > VCL Forms Application – Delphi
3. Choose Project > Options ... to open the Project Options dialog
4. In the options tree on the left, select 'Delphi Compiler'
5. Add the source directory of Habari and the Synapse source directory to the 'Search path'
6. Choose Ok to close the Project Options dialog
7. Save the project as HelloOpenMQ

Now the project is created and saved.

You should see the main form in the GUI designer now.

### Adding code to the project

To use the Habari client library, you need to add the required units to the source code.

8. Switch to Code view (F12)
9. Add the required units to the interface uses list:

---

<sup>1</sup> Delphi 2009 to XE2 are supported, only the IDE related steps are different

```
uses
  BTJMSTConnection,
  BTJMSTInterfaces,
  BTCommAdapterSynapse,
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs;
```

10. Compile and save the project.

11. Switch to Design view (F12), go to the Tool palette (Ctrl+Alt+P) and select TButton, add a Button to the form.

12. Double click on the new button to jump to the Button Click handler

13. Add the following code to send the message:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Connection: IConnection;
  Session: ISession;
  Destination: IDestination;
  Producer: IMessageProducer;
begin
  Connection := TBTJMSTConnection.MakeConnection('admin', 'admin',
    'stomp://localhost:7672');
  Connection.Start;
  Session := Connection.CreateSession(False, amClientAcknowledge);
  Destination := Session.CreateQueue('TEST.DEFAULT');
  Producer := Session.CreateProducer(Destination);
  Producer.Send(Session.CreateTextMessage('Hello world!'));
  Connection.Close;
end;
```

14. Add a second button and double click on the new button to jump to the Button Click handler

15. Add the following code to receive and display the message:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  Connection: IConnection;
  Session: ISession;
  Destination: IDestination;
  Consumer: IMessageConsumer;
  Msg: ITextMessage;
begin
  Connection := TBTJMSTConnection.MakeConnection('admin', 'admin',
    'stomp://localhost:7672');
  Connection.Start;
  Session := Connection.CreateSession(False, amAutoAcknowledge);
  Destination := Session.CreateQueue('TEST.DEFAULT');
  Consumer := Session.CreateConsumer(Destination);
  Msg := Consumer.Receive(1000) as ITextMessage;
  if Assigned(Msg) then
    ShowMessage(Msg.Text);
  Connection.Close;
end;
```

---

16. Compile and save the project

## Run the demo

- Launch OpenMQ
- Start the application
- Click on Button 1 to send the message to the OpenMQ queue
- Click on Button 2 to receive the message and display it

You can run two instances of the application at the same time, and also on different computers if the IP address of the message broker is used instead of localhost.

## Next steps

You now know how to accomplish some of the most common programming tasks for Habari. The next chapters provide details about the basic interfaces which are the building blocks for message broker clients.

---

## Online Tutorials

### Delphi integration with the GlassFish v3 application server

[This tutorial](#) will guide you through the creation of a simple web application for GlassFish V3 which uses a Servlet to send messages to a message queue on the embedded ActiveMQ broker.

<https://mikejustin.fogbugz.com/default.asp?W11>

The [second part of the tutorial](#) will guide you through the creation of a simple EJB application for GlassFish v3 which uses a Message Driven Bean to receive messages from a message queue on the embedded OpenMQ broker. The Delphi ProducerTool application sends messages to the message queue.

<https://mikejustin.fogbugz.com/default.asp?W12>

## 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.<sup>2</sup>

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

<sup>2</sup> <http://activemq.apache.org/failover-transport-reference.html>

**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>3</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:

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

---

<sup>3</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)

The queue can then be used to send or receive messages using implementations of the `IMessageProducer` and `IMessageConsumer` interfaces. The methods of a queue are defined in the `IQueue` interface and the parent interface `IDestination`. (See next chapter for an example)

## Topics

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

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

The topic can then be used to send or receive messages using implementations of the `IMessageProducer` and `IMessageConsumer` interfaces. The methods of a topic are defined in the `ITopic` interface and the parent interface `IDestination`. (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('foo');
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('foo');
Consumer := Session.CreateConsumer(Destination);
Consumer.MessageListener := Self;
...
```

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.

See also:

[JMS Message Listeners](#) in "The JMS API Programming Model"

[Interface MessageListener](#) in "Java Message Service (JMS) API"

---

## JMS Selectors

Selectors are a way of attaching a filter to a subscription to perform content based routing. Selectors are defined using SQL 92 syntax and typically apply to message headers; whether the standard properties available on a JMS message or custom headers you can add via the JMS code.

Here is an example

```
JMSType = 'car' AND color = 'blue' AND weight > 2500
```

For more documentation on the detail of selectors see the reference on `javax.jmx.Message`.

See also:

[JMS Message Selectors](#) in "The JMS API Programming Model"

[Interface MessageConsumer](#) in "Java Message Service (JMS) API"

## Text Messages

---

### Sending Text Messages

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('user', 'pass', '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
  procedure OnMessage(const 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
  TextMessage := Consumer.Receive(1000) as ITextMessage;
  if Assigned(TextMessage) then
  begin
    Inc(I);
    TextMessage.Acknowledge;
    L.Info(Format('%d %s', [I, TextMessage.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.   |

## Bytes Messages

---

### Creation

```
var
  Msg: IBytesMessage;
begin
  ..
  Producer := Session.CreateProducer(OutQueue);
  Msg := Session.CreateBytesMessage;
```

---

### Sending

#### 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
  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>4</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 Transformers in Habari Client for OpenMQ

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 1: 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>4</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 BTJMSSConnection:

```
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 TObExTestCase.TestXMLNative;
var
  ObjectMessage: IObjectMessage;
  Obj: SamplePojo;
begin
  // send
  Connection := TBTJMSSConnection.MakeConnection;
  try
    SetTransformer(Connection, TBTMessageTransformerXMLNative.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, TBTMessageTransformerXMLNative.Create(SamplePojo));
  Connection.Start;
  Session := Connection.CreateSession(False, amClientAcknowledge);
  Destination := Session.CreateQueue('TOOL.OBJECT.XML');
  Consumer := Session.CreateConsumer(Destination);
  ObjectMessage := Consumer.Receive(1000) as IObjectMessage;
  if Assigned(ObjectMessage) then
    begin
      ObjectMessage.Acknowledge;
      Obj := ObjectMessage.GetObject as SamplePojo;
      try
        CheckEquals('test', Obj.messageText);
        CheckEquals(0, Obj.messageNo);
      finally
        Obj.Free;
      end;
    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>5</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.

### Example

With the ProducerTool and ConsumerTool demo applications, you can send messages to a durable topic:

```
ProducerTool --MessageCount=1000 --Topic --Persistent -Subject=test-durable
```

and receive them from a client:

```
ConsumerTool --MaximumMessages=1000 --Topic --Subject=test-durable --Durable  
--ClientID=12345 --ConsumerName=12345 -Verbose
```

---

<sup>5</sup> <http://download.oracle.com/javase/5/api/javax/jms/TopicSession.html>

## 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 2: Basic Demo Applications

### 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 3: ConsumerTool Command Line Options

## 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 4: ProducerTool Command Line Options

## 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

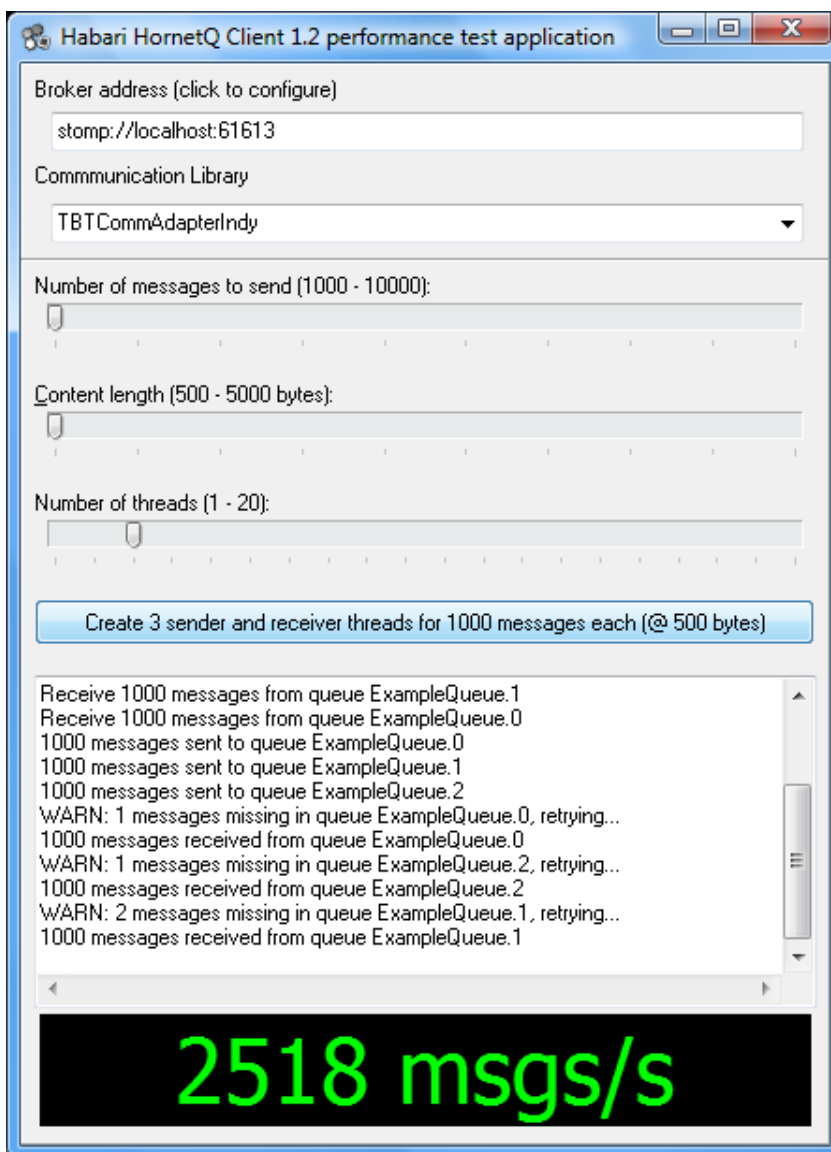
```
ProducerTool --Topic --Subject=ExampleTopic
```

## 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.



# 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>6</sup>

### JMS properties for outgoing messages

Messages sent by Habari Client for OpenMQ 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 OpenMQ 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>6</sup> <http://download.oracle.com/javase/5/api/javax/jms/Message.html>

---

## User Defined Properties

### Supported Data Types

The Stomp protocol only supports string type properties.

### Reserved Names

The following names are reserved Stomp header properties and can not be used as names for user defined properties:

- login
- passcode
- transaction
- session
- message
- destination
- id
- ack
- selector
- type
- content-length
- correlation-id
- expires
- persistent
- priority
- reply-to
- message-id
- timestamp
- transformation
- client-id
- redelivered

The client library detects overwriting of Stomp defined message properties. It will raise an Exception if the application tries to send a message with a reserved property name.

## Useful Units

---

### BTStreamHelper

This unit contains the procedure `LoadBytesFromStream` which can be used to read a file into a `BytesMessage`.

Example:

```
Msg := Session.CreateBytesMessage;

FS := TFileStream.Create('filename.dat', fmOpenRead);
try
  LoadBytesFromStream(Msg, FS);
  Size := Length(Msg.Content);
  WriteLn(IntToStr(Size) + ' Bytes');
finally
  FS.Free;
end;

Producer.Send(Msg);
```

### BTJavaPlatform

This unit contains some helper functions for Java dates. Java dates are `Int64` values based on the Unix date.

```
function JavaDateToTimeStamp(const JavaDate: Int64): TDateTime;
```

```
function TimeStampToJavaDate(const TimeStamp: TDateTime): Int64;
```

## 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\_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 HABARI\_USE\_RTTI

## Known Limitations

---

### Sessions

#### Acknowledgement Modes

Acknowledgment mode **"amDupsOkAcknowledge"** is unsupported.

Acknowledgment mode **"amAutoAcknowledge"** may cause message loss if you do not read all remaining messages in the Queue before closing the connection.

Background information: The Indy and Synapse libraries reads messages into an client-side buffer, and even when the client does not fetch the messages from the buffer (using one of the "Receive" methods), the server will handle them as 'delivered' and acknowledged. If the client reconnects, these messages will not be sent again.

---

### Messages

#### Message Types

OpenMQ only supports TextMessage and BytesMessage message types.

It is not possible to detect the message type for incoming messages (sent from the OpenMQ broker to the Stomp client) because there is no indicator in the message for the type. As a workaround, the Habari library for OpenMQ uses a non-standard Stomp header to indicate the message type:

- `messagetype=text`
- `messagetype=byte`

This header will be included in outgoing messages (from the Habari Stomp client to the message broker) and helps the receiver to identify the message type.

ObjectMessage support in the library is provided based on a message transformer architecture. However, native Java objects sent from Java clients will not work because OpenMQ will not send them to Stomp clients.

#### Unicode properties in Object Messages

Unicode string content is not supported for object properties in non-Unicode versions of Delphi.

---

---

## 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 (e.g. SetInt or SetDate), all message properties will be interpreted as Java Strings by the Message Broker.

As a side effect, the expressions in a Selector are limited to operations which are valid for strings.

Timestamp 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.

---

## Security

### Default account

The library currently uses admin / admin for the default user name / password.

## References

### Message Broker

Home page	<a href="http://mq.java.net/">http://mq.java.net/</a>
Forum	<a href="https://forums.oracle.com/forums/forum.jspa?forumID=874">https://forums.oracle.com/forums/forum.jspa?forumID=874</a>

### IDE

Embarcadero Delphi	<a href="http://www.embarcadero.com/products/delphi">http://www.embarcadero.com/products/delphi</a>
Lazarus	<a href="http://www.lazarus.freepascal.org/">http://www.lazarus.freepascal.org/</a>

### JMS

JMS Specification	<a href="http://www.oracle.com/technetwork/java/jms/index.html">http://www.oracle.com/technetwork/java/jms/index.html</a>
-------------------	---

### Stomp

Project home	<a href="http://stomp.github.com/">http://stomp.github.com/</a>
--------------	---

### Communication Libraries

Synapse	<a href="http://www.synapse.ararat.cz">http://www.synapse.ararat.cz</a>
Internet Direct (Indy)	<a href="http://www.indyproject.org/">http://www.indyproject.org/</a>
Indy Snapshot	<a href="http://indy.fulgan.com/ZIP">http://indy.fulgan.com/ZIP</a>

### Logging

Log4D	<a href="http://log4d.sourceforge.net/">http://log4d.sourceforge.net/</a>
-------	---

## Habari Client for OpenMQ License

Habari Client for OpenMQ (c) 2009-2012 Michael Justin

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.

---

## lkJSON

```
LkJSON v1.07
```

```
06 november 2009
```

```
* Copyright (c) 2006,2007,2008,2009 Leonid Koninin
* leon_kon@users.sourceforge.net
* 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 the <organization> 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 Leonid Koninin ``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 Leonid Koninin 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 2.1

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`

#### Changed

- ReceiveNoWait** ReceiveNoWait 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
- OpenMQ 4.5.2** Tested with OpenMQ 4.5.2
- Free Pascal 2.6.0** Compiles with Free Pascal 2.6.0
- Chat demo** The chat demo uses a connection configuration dialog, some user interface improvements
- 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 OpenMQ, defining the `HABARI_USE_STOMP11` symbol will cause a compiler error)

---

### Version 2.0

Released December 28, 2011

## Changed

<b>Frame Decoder</b>	The new Stomp Frame decoder implementation is always used, the conditional symbol HABARI_USE_TBYTES has been removed
<b>Indy rev. 4713</b>	Tested with revision 4713 of Indy 10.5.8
<b>OpenMQ 4.6</b>	Tested with OpenMQ version 4.4u1, 4.4u2, 4.5, 4.5.1 and 4.6 (build 3-c)
<b>Fixes</b>	Improvements in the Indy and Synapse communication adapter, and in the JSON SuperObject and OmniXml message transformers
<b>Demo</b>	The GUI demo allows to save the file which is in an incoming message

---

## Version 1.9

Released September 20, 2011

## New

<b>Delphi XE2</b>	Compilation tested with Win32, Win64 and OSX (new versions of DUnit, Synapse and Indy, and NativeXML are required)
<b>MapMessage</b>	Support for MapMessages, implementations for NativeXml and OpenXML
<b>Throughput Demo</b>	The throughput tool continuously produces and consumes messages to monitor the average message throughput
<b>OpenMQ 4.6</b>	Tested with OpenMQ version 4.4u1, 4.4u2, 4.5.1 and 4.6 (build 3-c)
<b>common-tests</b>	New unit test suites (shared with all Habari JMS Client libraries)
<b>IkJSON</b>	Added experimental support for the IkJSON library (unit BTMessageTransformerJSONIk)

## Changed

<b>DUnit rev. 41</b>	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)
<b>Synapse rev. 144</b>	Upgraded for XE2 support (included)
<b>NativeXML v. 4.01</b>	Upgraded for XE2 support (included)
<b>Indy rev. 4676</b>	Tested with revision 4676 of Indy 10.5.8
<b>Unicode Tests</b>	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)
<b>DUnit tests</b>	Fixed a memory leak in the test case for BTQueueRequestor
<b>ReadLineTimeout</b>	The Indy adapter uses IOHandler.ReadLnTimeout to detect a timeout after read

---

<b>OmniXml</b>	Fixed TBtMessageTransformerXMLOmni.Decode for Unicode properties in object messages
<b>Object Messages</b>	Units for object message transformers moved from source/beta to source folder

---

## Version 1.8

Released June 14, 2011

### New

<b>Durable Subscribers</b>	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. This version introduces support for Durable Subscribers. For details see chapter Durable Subscriptions
<b>NativeXml</b>	Support for the NativeXml open source XML parser library, class TBtMessageTransformerXMLNative for IObjectMessage
<b>OpenMQ 4.5 /autolaunch</b>	Tested with OpenMQ version 4.4u1, 4.4u2 and 4.5 If a command line parameter /autolaunch is specified, the DUnit test program will launch the OpenMQ brokers 4.4u1, 4.4u2 and 4.5 automatically in the test suite setup and terminate the broker in the teardown stage
<b>NoLocal</b>	If this parameter of ISession#CreateConsumer is set, it inhibits the delivery of messages published by its own connection.
<b>Transformer</b>	A helper method, SetTransformer, can be used to set a message transformer on a connection
<b>Object Exchange</b>	Unit tests for object messages (unit ObjectExchangeTests) have been added to the test suites, they replace the omnixml and superobject demo applications

### Changed

<b>Refactoring</b>	Refactored to new broker-independent units BTSerialIntf, BTSessionIntf instead of unit BTOMQInterfaces
<b>Frame Decoder</b>	The new unit BTStompDecoder contains a new Stomp Frame decoder implementation, it can be enabled with the conditional symbol HABARI_USE_TBYTES
<b>Performance Demo</b>	Shared code with Habari ActiveMQ Client and Habari Client for OpenMQ, display transfer speed in msgs/s
<b>Indy 10.5.8</b>	Tested with Indy 10.5.8 revision 4639
<b>FPC 2.4.4</b>	Build tested with Free Pascal 2.4.4
<b>Logging</b>	Removed unused logging units (BTLog etc)
<b>TransformationId</b>	The client verifies if the transformation id of the message transformer matches the 'transformation' header of incoming object messages
<b>TTransformable</b>	Class TTransformable is deprecated

---

<b>Stomp headers</b>	Repeated headers will be ignored, only the first value will be used
<b>BTypes</b>	Uses System.SetString for TBytes -> RawByteString conversion and SysUtils.ToBytes for RawByteString -> TBytes conversion

---

## Version 1.7

Released March 8, 2011

### New

<b>OpenMQ 4.5b29</b>	Tested with OpenMQ version 4.4u1, 4.4u2 and 4.5b29
<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>CreateMessage</b>	Function Session#CreateMessage returns a IMessage object

### Changed

<b>ReadOneMessage</b>	The internal method ReadOneMessage in TBTCCommAdapterIndy has a new parameter, ATimeout
<b>OnVerifyPeer</b>	BTCommAdapterIndySSL updated to use the method signature of current Indy version with an additional AError parameter
<b>Log4D</b>	Log4D updated to revision 37
<b>SuperObject</b>	SuperObject updated to revision 39
<b>GUI demo fix</b>	Fixed an AV which occurred when the demo program was compiled without assertions
<b>ERROR frame</b>	The body of ERROR frames is included in the exception message

---

## Version 1.6

Released December 14, 2010

### New

<b>IConnectionInfo</b>	New interface in BtMgmtInterfaces which provides connection state information
<b>OpenMQ 4.5b19</b>	Tested with OpenMQ version 4.5b19 (Milestone 6 build for GlassFish 3.1 project)
<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>FPC 2.4.2</b>	Tested with Free Pascal 2.4.2

## 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 |

## Examples

- |                          |  |
|--------------------------|--|
| <b>DelphiGUI logging</b> | Use Log4D for logging  |
| <b>DelphiGUI dialog</b>  | Added a Connection Factory configuration dialog  |
| <b>DelphiGUI flicker</b> | Reduced flickering of the Delphi GUI demo application  |
| <b>ExampleQueue</b>      | DelphiGUI Demo uses default name 'ExampleQueue' for the JMS destination  |
| <b>Log4D JMSAppender</b> | Provided an example implementation of a JMS log appender for the open source Log4D logging framework (unit LogJMSAppender) |

---

## Version 1.5

Released October 14, 2010

## New

- |                            |   |
|----------------------------|---|
| <b>Delphi XE</b>           | Ready for Delphi XE   |
| <b>Log4D Library</b>       | The Log4D logging library is now included and used when HABARI_LOGGING is defined       |
| <b>Unit Tests</b>          | Unit tested with Open Message Queue version 4.4u1, 4.4u2 and 4.5 b16                    |
| <b>Library Information</b> | The Connection Factory class now implements a new interface, IClientLibraryInfoProvider |
| <b>Indy 10.5.8</b>         | Tested with Indy 10.5.8   |

## Changed

- |                            |   |
|----------------------------|---|
| <b>doxygen</b>             | Updated to doxygen 1.7.1, fixes   |
| <b>Subscription Header</b> | The library now removes the Stomp header 'subscription' which is included in incoming messages. It is added by the broker so that the client knows which subscription the message relates to. |
| <b>Documentation</b>       | Documentation updates and fixes   |

---

## Version 1.4

Released March 30, 2010

## New

- Logging Switch** If the HABARI\_LOGGING compiler condition is set, logging code will be included. This will reduce code size in production and also improve performance
- IPv6** Prepared for IPv6 on Linux
- UMS Monitor demo** A new demo application shows how message broker information can be retrieved over HTTP and the imqums web application

## Changed

- Synapse** Improved performance for Synapse communication adapter
- Free Pascal** Tested with Free Pascal 2.5.1
- SuperObject** Update to SuperObject 1.2.4

---

## Version 1.3

Released January 12, 2010

## New

- GlassFish v3** The documentation includes configuration information for OpenMQ embedded in the GlassFish v3 application server and links to online tutorials which describe Delphi and Java integration with GlassFish and the NetBeans IDE

## Changed

- GlassFish support** Tested with GlassFish v2.1.1 and GlassFish v3
- OpenMQ version** Tested with OpenMQ build 4.4 u1 Final released Dec 4, 2009
- SuperObject version** Updated to SuperObject 1.2.2 released Dec 22, 2009
- Linux** Tested on Ubuntu 9.10 with Free Pascal 2.2
- Multithreading demo** The performance test demo application can create up to 20 threads

## Fixed

- ConnectTimeout** Fixed default value for the ConnectTimeout property

---

## Version 1.2

Released November 10, 2009

## New

**ConnectTimeout** New property in BTJMSConnection and BTJMSConnectionFactory

## Changed

**OpenMQ version** Tested with OpenMQ build 4.4u1 b3 released October 27, 2009. According to the release notes, this is release candidate 1 for planned inclusion in GlassFish v3. (There will probably be another release candidate) – see: <http://download.java.net/mq/openmq/4.4u1/b3-rc1/changes.html>

**Indy 10 version** Tested with Indy 10.5.7 revision revision 3865. Fixed Unicode conversion error in Indy communication adapter method ReadMessageBuffer (below Delphi 2009).

**Synapse version** Updated to release 39

---

## Version 1.1

Released September 8, 2009

## New

**ReceiveNoWait** MessageConsumer now provides three methods to read messages from a destination. The new method ReceiveNoWait is non-blocking (it will immediately return nil if there is no message), it replaces the old Receive method. Receive will now block until a message arrives.

**Delphi 2010** Tested with Delphi 2010 (all unit tests passed)

## Changed

**OpenMQ version** Tested with OpenMQ build 4.4 b15 released August 14, 2009.

**Indy 10 version** Updated to Indy 10.5.6.

**Demo applications** Demo applications will report memory leaks on shutdown.

**Minor changes** JMS interface and Stomp command key cleanup.

---

## Version 1.0

Released July 7, 2009

## Index

### Reference

AmAutoAcknowledge.....	40	JMSReplyTo.....	36
AmDupsOkAcknowledge.....	40	JMSTimestamp.....	36
BTJMSConnection.....	30	LkJSON.....	46
BTStreamHelper.....	28	LoadBytesFromStream.....	28
Connection.....	17	Log4D.....	39, 42
Connection factory.....	17	Message Consumer.....	23
ConsumerTool.....	33	Message Producer.....	23
CreateDurableSubscriber.....	32	MessageListener.....	24, 26
CreateObjectMessage.....	30	MessageTransformer.....	30
Failover Support.....	19	NativeXml.....	29, 47
HABARI_LOGGING.....	39	Object Message.....	29
HABARI_RAW_TRACE.....	39	OmniXML.....	29
HABARI_USE_RTTI.....	39	OnMessage.....	24, 26
IConnection.....	17	Point-to-point.....	21
IDestination.....	22, <b>26</b>	ProducerTool.....	34
IMessage.....	<b>26</b>	Publish and subscribe.....	21
IMessageConsumer.....	22, <b>26</b>	Queue.....	21
IMessageListener.....	<b>26</b>	Receive.....	27
IMessageProducer.....	22, 30	ReceiveNoWait.....	27
Internet Direct (Indy).....	10, <b>11</b> , 42	SamplePojo.....	30
IQueue.....	22	Session.....	18
ISession.....	30	SetTransformer.....	30
ITopic.....	22	Stomp.....	42
JMS.....	42	SuperObject.....	29, 46
JMS Selector.....	24	Synapse.....	10, <b>11</b> , 42
JMSCorrelationID.....	36	Text Message.....	25
JMSDeliveryMode.....	36	Topic.....	22
JMSExpiration.....	36	TopicSubscriber.....	32
JMSMessageId.....	36	Transacted Sessions.....	19
JMSPriority.....	36		

## Illustrations

Illustration 1: Shared Business Logic.....	5
Illustration 2: Peer to Peer Communication.....	5
Illustration 3: Load Balancing.....	6

## Tables

Table 1: Message Transformer Implementations.....	29
Table 2: Basic Demo Applications.....	33
Table 3: ConsumerTool Command Line Options.....	33
Table 4: ProducerTool Command Line Options.....	34