



habarisoft
Enterprise Messaging Software for Delphi®

Getting started with Habari Client for ActiveMQ

Version 3.1

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

| | |
|---|-----------|
| Introduction | 5 |
| About Habari Client for ActiveMQ | 5 |
| About Apache ActiveMQ | 7 |
| Installation | 9 |
| Requirements | 9 |
| TCP/IP Communication Libraries | 10 |
| Starting ActiveMQ | 11 |
| Pre-Installation Requirements | 11 |
| Running the Broker | 11 |
| Startup using "xbean" Profiles | 12 |
| Monitoring ActiveMQ | 12 |
| Stopping ActiveMQ | 13 |
| ActiveMQ Authentication Configuration | 13 |
| Stomp configuration | 14 |
| SSL Configuration | 15 |
| Additional Resources | 15 |
| Communication Adapter Configuration | 16 |
| Introduction | 16 |
| The Programming Model | 18 |
| Connections and Sessions | 19 |
| Step by Step Example | 19 |
| Transacted Sessions | 21 |
| Failover Support | 22 |
| Destinations | 24 |
| Introduction | 24 |
| Create a new Destination | 25 |
| Destination Options | 25 |
| Producer and Consumer | 27 |
| Message Producer | 27 |
| Message Consumer | 27 |
| JMS Selectors | 28 |
| Using XPath to filter messages | 29 |
| Text Messages | 30 |
| Sending a TextMessage | 30 |
| Receive Text Messages | 31 |
| Binary Messages | 33 |
| Send Binary Messages | 33 |
| Object Messages | 34 |
| Introduction | 34 |
| Message Transformers in Habari Client for ActiveMQ | 36 |
| Code Examples | 36 |
| Map Messages | 40 |

| | |
|--|-----------|
| Introduction | 40 |
| Durable Subscriptions | 42 |
| Description..... | 42 |
| Virtual Destinations..... | 43 |
| Stomp 1.1 | 44 |
| Compiler Switch..... | 44 |
| Connection Configuration..... | 44 |
| Sending Client-side Heartbeat..... | 45 |
| Checking for Server-side Heartbeats..... | 46 |
| Example Applications | 47 |
| Example Application Index..... | 47 |
| ConsumerTool..... | 49 |
| ProducerTool..... | 56 |
| Throughput Test Tool..... | 61 |
| Broker Statistics Example..... | 62 |
| Performance Test..... | 66 |
| PHP Producer Demo..... | 67 |
| Object Exchange with JSON..... | 69 |
| Delay and Schedule Message Delivery..... | 71 |
| Message Options | 73 |
| JMS Standard Properties..... | 73 |
| User Defined Properties..... | 75 |
| Temporary Queues | 77 |
| Introduction..... | 77 |
| How should I implement request response with JMS?..... | 77 |
| Online Tutorials | 78 |
| Habari Client for ActiveMQ with Geronimo 2.2 and Eclipse..... | 78 |
| Using Habari Client for ActiveMQ with the Geronimo 2.2 application server..... | 78 |
| Using Habari Client for ActiveMQ with the Geronimo application server..... | 78 |
| Using Habari Client for ActiveMQ with FUSE Message Broker..... | 79 |
| Unit Tests | 80 |
| JUnit Tests..... | 80 |
| Conditional Symbols | 82 |
| HABARI_LOGGING..... | 82 |
| HABARI_RAW_TRACE..... | 82 |
| HABARI_STOMP_11..... | 82 |
| HABARI_USE_RTTI..... | 82 |
| Known Limitations | 83 |
| Internet Direct (Indy) Communication Adapter..... | 83 |
| Sessions..... | 83 |
| Messages..... | 84 |
| Multi Threading..... | 84 |
| BytesMessage and stomp+nio..... | 84 |
| Default Priority..... | 84 |

| | |
|---|------------|
| References | 86 |
| Habari Client for ActiveMQ License | 88 |
| Third Party Library Licenses | 91 |
| Synapse..... | 91 |
| Indy BSD License..... | 92 |
| IkJSON..... | 92 |
| SuperObject..... | 93 |
| Log4D..... | 93 |
| NativeXml..... | 94 |
| Release Notes | 95 |
| Version 3.1..... | 95 |
| Version 3.0..... | 96 |
| Version 2.9..... | 97 |
| Version 2.8..... | 97 |
| Version 2.7..... | 99 |
| Version 2.6..... | 100 |
| Version 2.5..... | 101 |
| Version 2.4..... | 102 |
| Version 2.3..... | 102 |
| Version 2.2..... | 103 |
| Version 2.1..... | 104 |
| Version 2.0..... | 105 |
| Version 1.9..... | 106 |
| Version 1.8..... | 107 |
| Version 1.7..... | 107 |
| Version 1.6..... | 108 |
| Version 1.5..... | 109 |
| Version 1.4..... | 111 |
| Version 1.3..... | 112 |
| Version 1.2..... | 112 |
| Version 1.1..... | 113 |
| Version 1.0.1..... | 113 |
| Version 1.0..... | 114 |
| Index | 115 |

Introduction

About Habari Client for ActiveMQ

Habari Client for ActiveMQ is a library for Delphi and Free Pascal which provides easy access to Apache ActiveMQ, the most popular and powerful open source Message Broker. With this library, Delphi developers can build integrated solutions, connecting cross language clients and protocols from Java(tm), C, C++, C#, Ruby, Perl, Python, and PHP, using the peer-to-peer or the publish and subscribe communication model. Habari Client for ActiveMQ uses the Stomp message protocol and a plug-in architecture for communication libraries and message transformers for XML and JSON object serialization.

How can I use it?

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

- **Application Server Integration:** [Apache ActiveMQ](#) is a key component of the [Apache Geronimo](#) Web Application Server and [IBM WebSphere Application Server Community Edition](#)
- **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.
- **Logging:** standard logging frameworks like [Apache log4j](#) support logging to a JMS topic (JMSAppender), with Habari it is possible to add Delphi applications to an existing logging infrastructure.
- **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

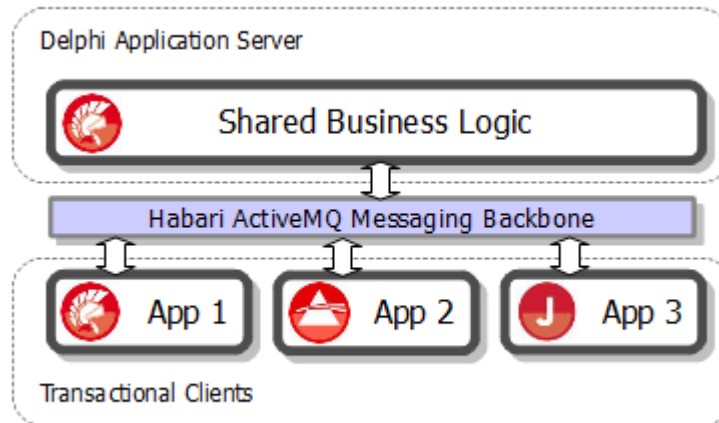


Illustration 1: Shared Business Logic

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

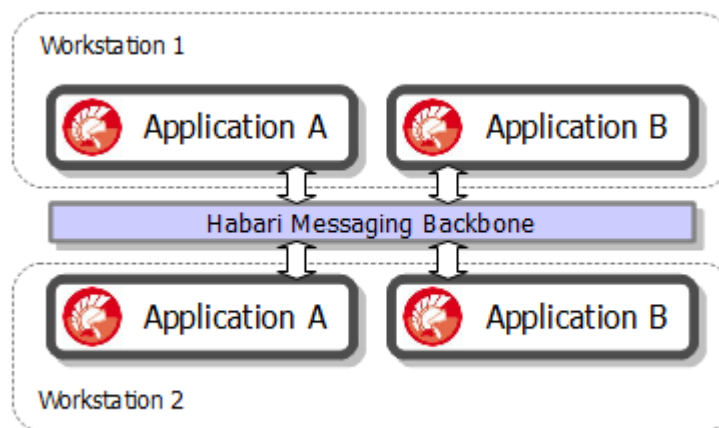


Illustration 2: Peer to Peer Communication

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

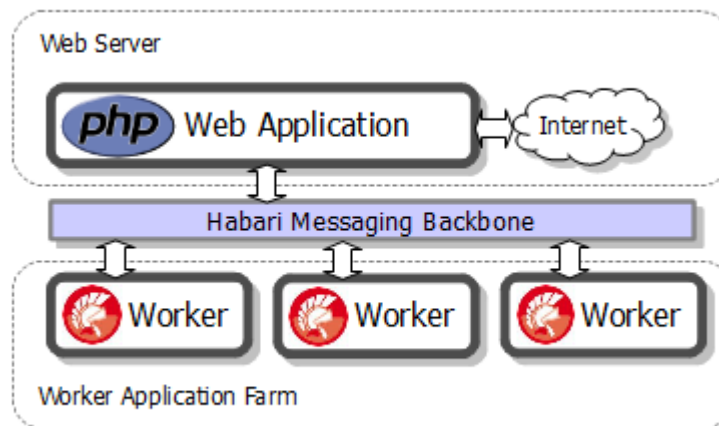


Illustration 3: Load Balancing

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.

About Apache ActiveMQ

Apache ActiveMQ is the most popular and powerful open source Message Broker and Enterprise Integration Patterns provider. Apache ActiveMQ is fast, supports many Cross Language Clients and Protocols, comes with easy to use Enterprise Integration Patterns and many advanced features while fully supporting JMS 1.1 and J2EE 1.4.

Apache ActiveMQ Features¹

- Supports a variety of [Cross Language Clients and Protocols](#) from Java, C, C++, C#, Ruby, Perl, Python, PHP
 - [OpenWire](#) for high performance clients in Java, C, C++, C#
 - [Stomp](#) support so that clients can be written easily in C, Ruby, Perl, Python, PHP, ActionScript/Flash, Smalltalk to talk to ActiveMQ as well as any other [popular Message Broker](#)
- full support for the [Enterprise Integration Patterns](#) both in the JMS client and the Message Broker
- Supports many [advanced features](#) such as [Message Groups](#), [Virtual Destinations](#), [Wildcards](#) and [Composite Destinations](#)
- Fully supports JMS 1.1 and J2EE 1.4 with support for transient, persistent, transactional and XA messaging
- [Spring Support](#) so that ActiveMQ can be easily embedded into Spring applications and configured using Spring's XML configuration mechanism
- Tested inside popular J2EE servers such as Geronimo, JBoss 4, GlassFish and WebLogic
 - Includes [JCA 1.5 resource adaptors](#) for inbound & outbound messaging so that ActiveMQ should auto-deploy in any J2EE 1.4 compliant server
- Supports pluggable [transport protocols](#) such as [in-VM](#), TCP, SSL, NIO, UDP, multicast, JGroups and JXTA transports
- Supports very fast [persistence](#) using JDBC along with a high performance journal
- Designed for high performance clustering, client-server, peer based communication
- [REST](#) API to provide technology agnostic and language neutral web based API to messaging
- [Ajax](#) to support web streaming support to web browsers using pure DHTML, allowing web browsers to be part of the messaging fabric
- [CXF and Axis Support](#) so that ActiveMQ can be easily dropped into either of these web service stacks to provide reliable messaging
- Can be used as an in memory JMS provider, ideal for [unit testing JMS](#)

¹ <http://activemq.apache.org/index.html>

Installation

Requirements

Development Environment

- Embarcadero Delphi 2009 or higher
- Free Pascal

Message Broker

- Apache ActiveMQ 5 or higher
- IONA FUSE Message Broker

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 (for Delphi 2009 to XE2 and Free Pascal)

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

Starting ActiveMQ

Pre-Installation Requirements²

Hardware:

- 40 MB of free disk space for the ActiveMQ binary distribution.
- 200 MB of free disk space for the ActiveMQ source or developer's distributions.

Operating Systems:

- Windows: Windows XP SP2, Windows 2000.
- Unix: Ubuntu Linux, Powerdog Linux, MacOS, AIX, HP-UX, Solaris, or any Unix platform that supports Java.

Environment:

- Java Developer Kit (JDK) 1.5.x or greater for deployment and 1.5.x (Java 5) for compiling/building.
- The JAVA_HOME environment variable must be set to the directory where the JDK is installed, e.g., `c:\Program Files\jdk.1.5.0_07-87`.

Download the binary distribution

After downloading from <http://activemq.apache.org/download.html> and unpacking ActiveMQ, you are ready to start the messages broker.

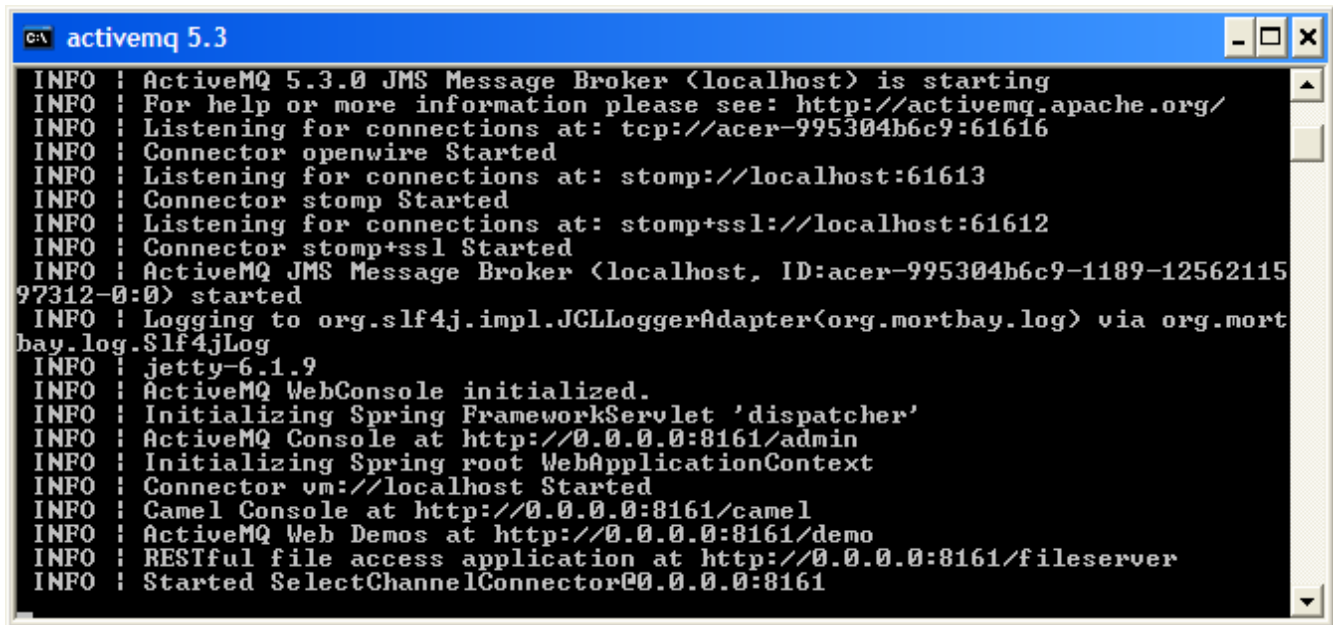
Running the Broker

From the binary distribution you can run the Apache ActiveMQ server pretty easily via the `bin/activemq` command. e.g. from a shell type

```
cd bin
activemq
```

² <http://activemq.apache.org/version-5-getting-started.html>

The Apache ActiveMQ broker should now have started.

A screenshot of a Windows command prompt window titled "activemq 5.3". The window displays the following log output:

```
INFO : ActiveMQ 5.3.0 JMS Message Broker (localhost) is starting
INFO : For help or more information please see: http://activemq.apache.org/
INFO : Listening for connections at: tcp://acer-995304b6c9:61616
INFO : Connector openwire Started
INFO : Listening for connections at: stomp://localhost:61613
INFO : Connector stomp Started
INFO : Listening for connections at: stomp+ssl://localhost:61612
INFO : Connector stomp+ssl Started
INFO : ActiveMQ JMS Message Broker (localhost, ID:acer-995304b6c9-1189-12562115
97312-0:0) started
INFO : Logging to org.slf4j.impl.JCLLoggerAdapter(org.mortbay.log) via org.mort
bay.log.Slf4jLog
INFO : jetty-6.1.9
INFO : ActiveMQ WebConsole initialized.
INFO : Initializing Spring FrameworkServlet 'dispatcher'
INFO : ActiveMQ Console at http://0.0.0.0:8161/admin
INFO : Initializing Spring root WebApplicationContext
INFO : Connector vm://localhost Started
INFO : Camel Console at http://0.0.0.0:8161/camel
INFO : ActiveMQ Web Demos at http://0.0.0.0:8161/demo
INFO : RESTful file access application at http://0.0.0.0:8161/fileserver
INFO : Started SelectChannelConnector@0.0.0.0:8161
```

Illustration 4: ActiveMQ 5.3.0 running with Stomp and Stomp+SSL connectors

Startup using "xbean" Profiles

Newer versions of ActiveMQ support different start configurations out of the box.

Every configuration is stored in a XML file, which can be passed in the command line with the prefix 'xbean:'.

For example, this command line will launch the broker with the Stomp configuration file `activemq-stomp.xml`:

```
cd \apache-activemq-5.4.2\bin
activemq-admin.bat start xbean:activemq-stomp.xml
```

Monitoring ActiveMQ

There are various ways to [monitor ActiveMQ](#). If you are on version 4.2 or later of ActiveMQ you can then monitor ActiveMQ using the [Web Console](#) by pointing your browser at

<http://localhost:8161/admin>

Or you can use the [JMX](#) support to view the running state of ActiveMQ.

Stopping ActiveMQ

For both Windows and Unix installations, terminate ActiveMQ by typing "CTRL-C" in the console or command shell in which it is running.

In newer releases of the broker, you can also use the `activemq-admin` script in the `bin` directory:

```
cd \apache-activemq-5.4.2\bin
activemq-admin.bat stop
```

ActiveMQ Authentication Configuration

If you have modest authentication requirements (or just want to quickly set up your testing environment) you can use `SimpleAuthenticationPlugin`.

With this plugin you can define users and groups directly in the broker's XML configuration.³

Take a look at the following snippet for example:

³ For more information see <http://activemq.apache.org/security.html>

```
<broker xmlns="http://activemq.apache.org/schema/core" brokerName="localhost"
dataDirectory="${activemq.base}/data">

...

<plugins>

  <simpleAuthenticationPlugin>
    <users>
      <authenticationUser username="system" password="manager"
        groups="users,admins"/>
      <authenticationUser username="user" password="password"
        groups="users"/>
      <authenticationUser username="guest" password="password" groups="guests"/>
    </users>
  </simpleAuthenticationPlugin>

</plugins>

</broker>
```

Caveat:

The default `activemq.xml` configuration file comes with three optional and enabled elements: `<commandAgent>`, `<camelContext>`, and `<jetty>`. If you enable authentication & authorization services, these enabled elements will cause the broker to throw security-related exceptions. This is because these elements represent functionality that is essentially represented by clients that need to connect to the broker and the connections are made without security credentials. If you do not require the functionality behind these elements, disable or comment-out the elements.

ActiveMQ 5.1

The ActiveMQ Stomp connector supports password authentication only in versions since version 5.1.

Stomp configuration

The default configuration of ActiveMQ 5.3 does not activate Stomp support. The Stomp connector can be activated by editing the configuration file `activemq.xml`⁴:

4 <http://activemq.apache.org/stomp.html>

```
<transportConnectors>
  ...
  <transportConnector name="stomp" uri="stomp://localhost:61613"/>
  ...
</transportConnectors>
```

SSL Configuration

For ActiveMQ 5.3, the SSL configuration is not included by default. Add these lines to activemq.xml:

```
...
<broker>
  ...
  <sslContext>
    <sslContext keyStore="file:${activemq.base}/conf/broker.ks"
keyStorePassword="password" trustStore="file:${activemq.base}/conf/broker.ts"
trustStorePassword="password"/>
  </sslContext>
  ...
</broker>
...
```

Additional Resources

FUSE Source [ActiveMQ Getting Started Guide](#) (**Note:** FUSE Message Broker is an open source Apache licenced enterprise version of ActiveMQ).

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 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 ActiveMQ libraries includes two adapters for TCP/IP libraries, one for Indy (Internet Direct) and one for Synapse.

| Adapter Class | Unit |
|--------------------------|--------------------------------------|
| TBTCommAdapterIndy | BTCommAdapterIndy |
| TBTCommAdapterSynapse | BTCommAdapterSynapse |
| TBTCommAdapterIndySSL | BTCommAdapterIndySSL |
| TBTCommAdapterSynapseSSL | BTCommAdapterSynapseSSL ⁵ |

Table 1: Communication Adapters

⁵ This adapter is still in an early development stage

The Programming Model

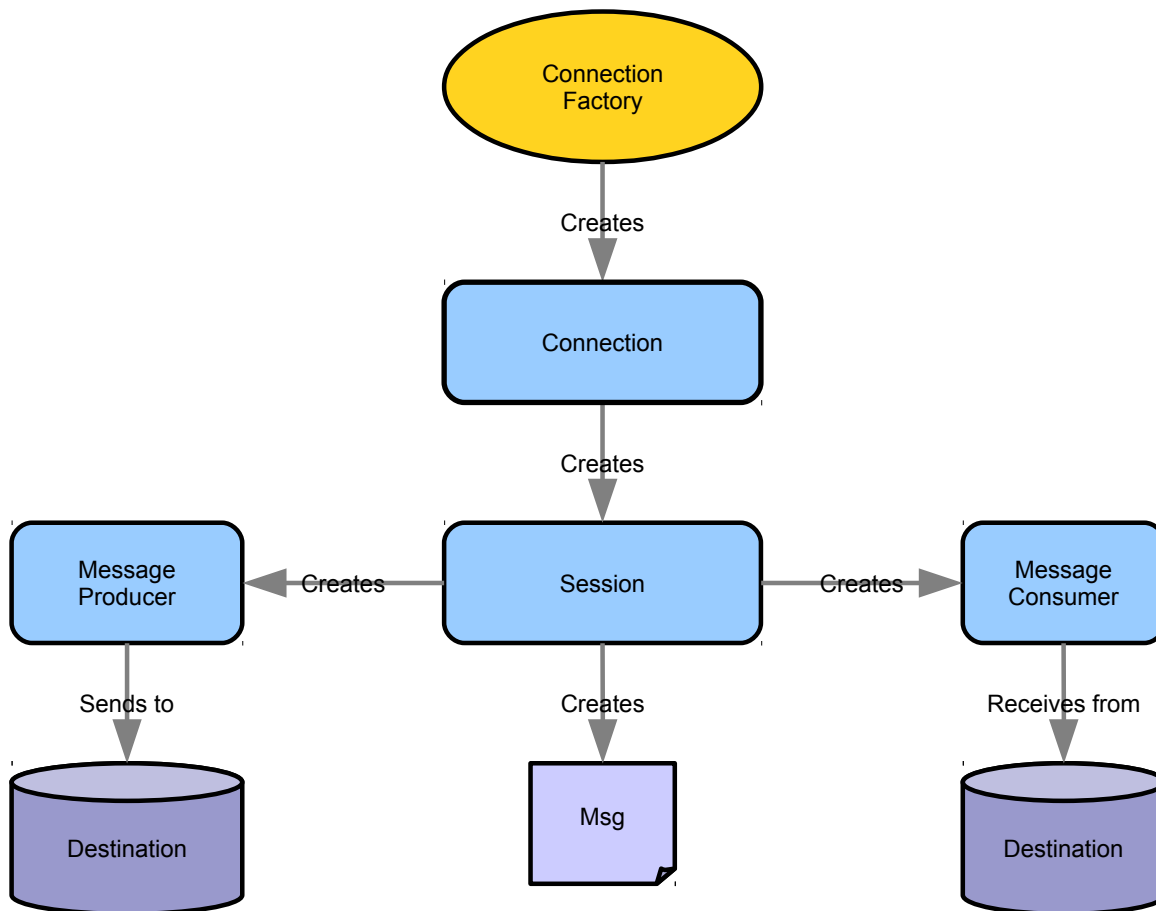


Illustration 5: Programming Model

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 (BTJMSSConnectionFactory or BTJMSSConnection)
- the unit containing the interface declarations (BTJMSSInterfaces)

The SysUtils unit is necessary for the exception handling.

```
program SendOneMessage;  
  
{$APPTYPE CONSOLE}  
  
uses  
  SysUtils,  
  BTCommAdapterIndy,  
  BTJMSSConnection,  
  BTJMSSInterfaces;  
  ...
```

Creating a new Connection

To create a new connection,

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

or

- use an instance of TBTJMSSConnectionFactory 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.⁶

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)
```

Failover 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 2: Failover Transport Options

Example URI:

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

Example code:

⁶ <http://activemq.apache.org/failover-transport-reference.html>

```
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:⁷

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.

⁷ Java Message Service. (2007, November 21). In Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Java_Message_Service

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);
```

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('bar');  
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).

Destination Options

Destination Options are a way to provide extended configuration options to a JMS consumer without having to extend the JMS API. The options are encoded using URL query syntax in the destination name that the consumer is created on.⁸

Example:

```
Destination := Session.CreateQueue('queue?activemq.retroactive=true');  
Consumer := Session.CreateConsumer(Destination);
```

⁸ <http://activemq.apache.org/destination-options.html>

activemq.dispatchAsync (boolean)

Should messages be dispatched synchronously or asynchronously from the producer thread for non-durable topics in the broker? For fast consumers set this to **false**. For slow consumers set it to **true** so that dispatching will not block fast consumers.

activemq.exclusive (boolean)

I would like to be an Exclusive Consumer on the queue.⁹

activemq.maximumPendingMessageLimit (int)

For Slow Consumer Handling on non-durable topics by dropping old messages - we can set a maximum-pending limit, such that once a slow consumer backs up to this high water mark we begin to discard old messages.¹⁰

activemq.prefetchSize (int)

Specifies the maximum number of pending messages that will be dispatched to the client. Once this maximum is reached no more messages are dispatched until the client acknowledges a message. Set to **1** for very fair distribution of messages across consumers where processing messages can be slow.

activemq.priority (byte)

Sets the priority of the consumer so that dispatching can be weighted in priority order.

activemq.retroactive (boolean)

A retroactive consumer is just a regular JMS consumer who indicates that at the start of a subscription every attempt should be used to go back in time and send any old messages (or the last message sent on that topic) that the consumer may have missed.¹¹

⁹ <http://activemq.apache.org/exclusive-consumer.html>

¹⁰ <http://activemq.apache.org/slow-consumer-handling.html>

¹¹ <http://activemq.apache.org/retroactive-consumer.html>

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);
```

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.

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.¹²

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`¹³.

ActiveMQ supports some JMS defined properties, as well as some ActiveMQ ones that the selector can use.

Note The Stomp protocol used by Habari Client for ActiveMQ only supports string type properties and operations in selectors.

Numeric selectors Apache ActiveMQ 5.6 introduced support for numeric expressions in JMS selectors¹⁴.

Delphi example:

```
MessageConsumer := Session.CreateConsumer(Destination, 'foo = ''bar''');
```

12 <http://activemq.apache.org/selectors.html>

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

14 <https://issues.apache.org/jira/browse/AMQ-1609>

Using XPath to filter messages

Apache ActiveMQ also supports XPath based selectors when working with messages containing XML bodies. To use an XPath selector use the following syntax

```
XPATH '//title[@lang='eng']'
```

Note

The standard installation of ActiveMQ does not include the Xalan JAR files which are necessary for XPATH evaluation. The files xalan.jar, xercesImpl.jar and xml-apis.jar need to be placed in the lib folder of ActiveMQ.

Delphi example:

```
MessageConsumer := Session.CreateConsumer(Destination, 'XPATH '//title[@lang="en"]''');
```

Text Messages

Sending a TextMessage

Source code for a simple application which sends a test message to a broker running on the local system using default credentials:

```
program SendOneMessage;

{$APPTYPE CONSOLE}

uses
  BTCommAdapterIndy, BTJMSConnection, BTJMSInterfaces,
  SysUtils;

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

begin
  Connection := TBTJMSConnection.MakeConnection;
  try
    Connection.Start;
    Session := Connection.CreateSession(False, amAutoAcknowledge);
    Destination := Session.CreateQueue('testqueue');
    Producer := Session.CreateProducer(Destination);
    Producer.Send(Session.CreateTextMessage('Test message'));
  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(IInterface)
  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
  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:

- | | |
|-------------------------|--|
| Receive | The Receive method with no arguments will block (wait until a message is available). |
| Receive(Timeout) | The Receive method with a timeout parameter will wait for the given time in milliseconds. If no message arrived, it will return nil. |
| ReceiveNoWait | 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.¹⁵ 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.

ActiveMQ supports object exchange between Java and non-Java clients using a Message Transformation between native Java objects and XML or JSON serialized objects.¹⁶

“Delphi Only” vs. “Cross-Language” Object Exchange

Habari Client for ActiveMQ offers two object exchange methods.

Cross-Language

Objects on the broker are encoded using Java binary serialization. The Delphi application sends a JSON or XML serialized object to the broker, who transforms it into a binary serialized Java object first before it can be consumed by JMS clients. This broker-side transformation requires that Java class files for the class are in the broker's class path.

Delphi Only

Objects on the broker are encoded using JSON or XML. The message broker exchanges the objects between Delphi (and other) clients serialized as JSON or XML text, no special Java support files are required. **This exchange method is new in version 2.8 of Habari Client for ActiveMQ.**

¹⁵ <http://java.sun.com/developer/technicalArticles/Programming/serialization/>

¹⁶ <http://activemq.apache.org/message-transformation.html>

“Cross-Language” Object Exchange

On the Java side, a Java client application does not need any special preparation to send and receive objects over ActiveMQ. The JMS API support for `ObjectMessage` provides all necessary methods, a session uses `Session#createObjectMessage(Serializable object)`¹⁷ to create the message (passing a Java object as argument) which then can be sent and received just like a `TextMessage` or `BytesMessage`.

However, for the message transformation to and from JSON or XML, this object exchange methods requires that a JAR containing a matching Java class file has to be deployed in the message broker, which will be used by the brokers message transformer. If this Java class is not compatible with the JSON or XML structure, the message transformation fails!

Pros

- Java clients do not need any special modifications to exchange objects with non-Java clients, Delphi clients can be connected ('plugged in') / integrated easily with an existing JMS infrastructure
- Serialization from / to objects is performed on the server
- Serialization only occurs 'on demand' when the non-Java client reads or writes messages

Cons

- Requires installation of a JAR file in the message broker which contains the Java class (unless the class is already in the brokers classpath)
- The transformation fails if the Java class and Delphi class declaration don't match
- The transformation fails if the Delphi and Java transformer libraries (JSON / XML) are not compatible

“Delphi Only” Object Exchange

There are almost no differences between in the Delphi code for “Cross-Language” and “Delphi Only” object exchange methods. Switching to “Delphi Only” object exchange requires only an additional property assignment on the object message. The serialized objects will be stored in the messages broker as `TextMessage` instances. The XML or JSON text can be retrieved by a JMS Java client application just like any other JMS `TextMessage`. Java clients can use a JSON or XML parser to read the message content.

Pros

- Simple usage, no JAR installation required
- Java JMS client applications are still able to receive the serialized objects – they will appear as `TextMessage` instances, containing the JSON or XML text

¹⁷ <http://download.oracle.com/javase/1.4/api/javax/jms/Session.html#createObjectMessage%28java.io.Serializable%29>

Cons

- Deserialization of JSON or XML serialized Delphi objects to Java objects requires a decoder library (XStream or Jettison) on the Java client side

Message Transformers in Habari Client for ActiveMQ

| Transformation | Message Type | Library | Unit |
|----------------|---------------|-------------|-------------------------------------|
| XML | ObjectMessage | OmniXML | BTMessageTransformerXMLOmni |
| XML | ObjectMessage | NativeXml | BTMessageTransformerXMLNative |
| XML | MapMessage | OmniXML | BTMessageTransformerXMLMapOmni |
| XML | MapMessage | NativeXml | BTMessageTransformerXMLMapNative |
| JSON | ObjectMessage | IkJSON | BTMessageTransformerJSONIk |
| JSON | ObjectMessage | SuperObject | BTMessageTransformerJSONSuperObject |

Table 3: 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.

Code Examples

A complete code example for "Delphi Only" object exchange is in chapter "Object Exchange with JSON" (p. 69). Here are some basic code excerpts to get you started.

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 := TBMessageTransformerXMLNative.Create(SamplePojo);
end;

...
Connection.Start;
```

With version 2.8 and newer, you can also use the helper procedure `SetTransformer` in unit `BTJMSSConnection`:

```
Connection: IConnection;
...

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

...
Connection.Start;
```

Request the Transformation Format

The ActiveMQ Broker must know which serialization format shall be used for the connection.

This information can be added to the destination name, using a predefined constant for the `transformation` message option header, and the transformation ID. Note that the transformation ID must match the ID of the used message transformer.

For example, this code tells the broker to serialize messages in XML format:

```
const
  Dest = 'logTopic'
    + '?' + BTStompTypes.SH_TRANSFORMATION
```

```
+ '=' + BTSerialIntf.TRANSFORMER_ID_OBJECT_XML;  
...  
  Destination := Session.CreateTopic(Dest);  
...
```

Valid transformation ID values are defined in unit BTSerialIntf.

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);
```

The transformation-custom Message Header

To send object messages without invoking of ActiveMQ message transformations, set the 'transformation-custom' header of the message to the transformation id of the message transformer. Example:

```
ObjectMessage := Session.CreateObjectMessage(Instance);  
  
// set the additional header (we use JSON object transformation here)  
ObjectMessage.SetStringProperty('transformation-custom', TRANSFORMER_ID_OBJECT_JSON)  
  
Producer.Send(ObjectMessage);
```

Complete Example using NativeXml

From ObjectExchangeTests.pas.

Send:

```
procedure TObExTestCase.TestXMLNative;  
var  
  ObjectMessage: IObjectMessage;  
  Obj: SamplePojo;  
begin  
  // send
```

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

Map Messages

Introduction

The JMS API supports map messages which consist of key-value pairs. Habari Client for ActiveMQ includes experimental 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.

Delphi Only and Cross-Language

Unlike object messages, map messages work both cross-language and between Delphi applications without special message properties or class file deployments. They only require that the correct transformation ID is added to the destination name.

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);
  end;
end;
```

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

Receive:

```
Connection := TBTJMSConnection.MakeConnection;
try
  SetTransformer(Connection, TBTMessageTransformerXMLMapNative.Create(nil));
  Connection.Start;
  Session := Connection.CreateSession(False, amClientAcknowledge);
  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
      MapMessage.Acknowledge;
      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.¹⁸ 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.

Further information can be found at

<http://activemq.apache.org/how-do-durable-queues-and-topics-work.html>

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.

A second option to create a durable topic subscriber is the web administration interface. In the default installation of Apache ActiveMQ 5.3, you can use the 'Subscribers' page located at <http://localhost:8161/admin/subscribers.jsp>.

Example

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

¹⁸ <http://download.oracle.com/javaee/5/api/javax/jms/TopicSession.html>

```
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
```

Virtual Destinations

An alternative solution in ActiveMQ which has some benefits compared with durable subscriptions: <http://activemq.apache.org/virtual-destinations.html>

Stomp 1.1

Compiler Switch

STOMP 1.1 is an experimental feature in Habari Client for ActiveMQ 3.0 which can be activated with the HABARI_STOMP11 compiler switch.

If this switch is active, the compiler will output a warning with the text

```
Compiled with experimental STOMP 1.1 support
```

(This warning is defined in the BTStompCustomClient unit).

Connection Configuration

A connection string can use additional URL parameters to configure Stomp 1.1

All parameters can be omitted to use the default value.

| Switch | Description | Default |
|--|---|------------|
| stomp.accept-version ¹⁹ | Supported Stomp versions in ascending order | "1.0, 1.1" |
| stomp.server ²⁰ (optional) | The name of a virtual host that the client wishes to connect to | URI.Host |
| stomp.heart-beat ²¹ | Heart beat (outgoing, incoming) | "0,0" |

Connection Factory Code Example:

¹⁹ http://stomp.github.com/stomp-specification-1.1.html#protocol_negotiation

²⁰ http://stomp.github.com/stomp-specification-1.1.html#CONNECT_or_STOMP_Frame

²¹ <http://stomp.github.com/stomp-specification-1.1.html#Heart-beating>

```
Factory := TBTJMSConnectionFactory.Create(  
    'stomp://localhost:61613?stomp.accept-version=1.1&stomp.heart-beat=1000,0');
```

This example creates a connection factory with these connection settings

host = localhost, port = 61613

accept-version = 1.1

heart-beat = 1000,0

- virtual host is localhost
- the client requests Stomp 1.1 protocol
- client heart beat interval is 1000 milliseconds, no server heart beat signals

Specification

For details see the Stomp specification page at

<http://stomp.github.com/stomp-specification-1.1.html>

Sending Client-side Heartbeat

A client connection uses the **SendHeartbeat** method of the connection object to send a heart-beat (newline 0x0A) byte. Example:

```
(Connection as IHeartbeat).SendHeartbeat;
```

Notes:

- the application code is responsible for sending a heartbeat message within the maximum interval
- client messages which are sent after the heart-beat interval expires may be lost

Checking for Server-side Heartbeats

A client connection can use the **CheckHeartbeat** method of the connection object to detect whether the server has sent a heartbeat message.

Example:

```
(Connection as IHeartbeat).CheckHeartbeat;
```

CheckHeartbeat raises an exception if the server did not send any data or heartbeat messages within the heartbeat interval.

Example Applications

Example Application Index

Basic Features

The following tables list the included demo applications. The first table contains demo apps for basic features.

| Directory | Description |
|---------------------|---|
| common-chat | Simple chat client. (HabariChat.dpr, requires Delphi 2009+) |
| common-consumertool | Receives messages from broker. See chapter "ConsumerTool" (p. 49) |
| common-delphigui | Sends and receives messages. (GUIDemo.dpr, requires Delphi 2009+) |
| common-performance | Multi-threaded performance test application. See chapter "Performance Test" (p. 66) (PerfTest.dpr, requires Delphi 2009+) |
| common-producertool | Sends messages to a broker. See chapter "ProducerTool" (p. 56) |
| common-throughput | Continuously produces and consumes messages to monitor the average message throughput over time |
| publishsubscribe | Sends / receives 2000 messages. |
| sendonemessage | See chapter "Step by Step Example" (p. 19) |

Table 4: Basic Demo Applications

| | |
|--------------------|--|
| Shared Code | common-chat, common-consumertool, common-performance, common-producertool and common-delphigui demo applications use a single source code base for all Habari Client libraries |
|--------------------|--|

Common Code

The directory `demo/common` contains Shared units:

- connection configuration form (ConnCfgFrm.pas/dfm)
- command line parameter support class (CommandLineSupport.pas)

Advanced Features

Advanced JMS examples and broker-specific features

| Directory | Description |
|---------------------|---|
| activemq-schedule | Example code for "Delay and Schedule Message Delivery" (p. 71) |
| activemq-statistics | Example code for "Broker Statistics Example" (p. 62) |
| loadbalancing | <p>The LoadServer application will connect with ActiveMQ on localhost and create a directory for outgoing files. Copy a file to the files directory. The LoadServer will now send it every five seconds to a ActiveMQ queue, including the file name, file size and a sequence number. (For safety reasons in this demo, the file will not be deleted.)</p> <p>The LoadClient application will connect with ActiveMQ and create a directory for incoming files. If the LoadClient finds a file, it will be downloaded with a filename including a time stamp.</p> <p>If you start LoadClient multiple times, ActiveMQ will distribute the files to all running clients.</p> |
| log4d | <p>GUI demo app which uses a Log4D appender class (source code included) to send log messages to a ActiveMQ topic (JMSAppenderDemo.dpr, requires the Log4D logging framework, log4d.sourceforge.net)</p> <p>Example configuration in log4d.props:</p> <pre># Create a JMS appender log4d.appender.Jms1=TLogJMSAppender log4d.appender.Jms1.threshold=trace log4d.appender.Jms1.logTopic=TOOL.DEFAULT log4d.appender.Jms1.errorHandler=TLogOnlyOnceErrorHandler log4d.appender.Jms1.layout=TLogPatternLayout log4d.appender.Jms1.layout.pattern= %p %d %c - %m%n log4d.appender.Jms1.layout.dateFormat=hh:nn:ss.zzz</pre> |
| log4jconsumer | These demo apps subscribe to a topic which contains Log4J logger messages, and request the messages serialized in JSON or XML format. A Log4J Java app is included in the log4jproducer folder in the Habari Integration Examples download. |
| obex-superobject | See chapter "Object Exchange with JSON" (p. 69) - Delphi object exchange using JSON serialization. |
| php-producer | See chapter "PHP Producer Demo" (p. 67) |
| tempdest | See chapter "Temporary Queues" (p. 77) - Temporary destinations (TempDest.dpr). |

Table 5: Advanced Demo Applications

ConsumerTool

The ConsumerTool demo may be used to receive messages from an ActiveMQ queue or topic. It is based on the Java example class ConsumerTool.java in the ActiveMQ binary distribution.

This example application is configurable by command line parameters, all are optional.

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

Table 6: 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
```

ConsumerTool Source Code

```
unit ConsumerToolUnit;

interface

uses
  BTJMSInterfaces;

type
{$M+}
  TConsumerTool = class(TInterfacedObject, IMessageListener)
  private
    Session: ISession;
    Running: Boolean;
    Consumer: IMessageConsumer;
    ReplyProducer: IMessageProducer;

    FAckMode: TAcknowledgementMode;
    FURL: string;
    FTopic: Boolean;
    FSubject: string;
    FDurable: Boolean;
    FSleepTime: Integer;
    FMaximumMessages: Integer;
    FTransacted: Boolean;
    FVerbose: Boolean;
    FUser: string;
    FPassword: string;
    FClientId: string;
    FConsumerName: string;
    FReceiveTimeOut: Integer;
    FPauseBeforeShutdown: Boolean;

    function TargetType: string;

    function DurableString: string;

    procedure SetAckMode(const Value: string);
```

```
procedure OnMessage(const Message: IMessage);

procedure ConsumeMessagesAndClose(const Conn: IConnection;
  const Session: ISession;
  const Consumer: IMessageConsumer); overload;

procedure ConsumeMessagesAndClose(const Conn: IConnection;
  const Session: ISession;
  const Consumer: IMessageConsumer; const TimeOut: Integer); overload;

public
  constructor Create;

  procedure Run;

published
  property AckMode: string write SetAckMode;
  property ClientId: string read FClientId write FClientId;
  property ConsumerName: string read FConsumerName write FConsumerName;
  property Durable: Boolean read FDurable write FDurable;
  property MaximumMessages: Integer read FMaximumMessages write
    FMaximumMessages;
  property Password: string read FPassword write FPassword;
  property PauseBeforeShutdown: Boolean read FPauseBeforeShutdown write
    FPauseBeforeShutdown;
  property ReceiveTimeOut: Integer read FReceiveTimeOut write FReceiveTimeOut;
  property SleepTime: Integer read FSleepTime write FSleepTime;
  property Subject: string read FSubject write FSubject;
  property Topic: Boolean read FTopic write FTopic;
  property Transacted: Boolean read FTransacted write FTransacted;
  property URL: string read FURL write FURL;
  property User: string read FUser write FUser;
  property Verbose: Boolean read FVerbose write FVerbose;

end;

implementation

uses
  CommandLineSupport,
  BTCommAdapterIndy,
  BTJMSTConnection,
  BTJMSTConnectionFactory,
  BTMgmtInterfaces,
  {$IFDEF VER130}
  Windows, // Sleep
  {$ENDIF}
  // StrUtils,
  SysUtils;

{ TConsumerTool }

constructor TConsumerTool.Create;
begin
  ConsumerName := 'Habari';
  FAckMode := amClientAcknowledge;
```

```
MaximumMessages := 10;
Password := BTJMSConnectionFactory.DEFAULT_PASSWORD;
Subject := 'TOOL.DEFAULT';
URL := BTJMSConnectionFactory.DEFAULT_BROKER_URL;
User := BTJMSConnectionFactory.DEFAULT_USER;
Verbose := True;
end;

procedure TConsumerTool.SetAckMode(const Value: string);
begin
  if Value = 'CLIENT_ACKNOWLEDGE' then
    FAckMode := amClientAcknowledge
  else if Value = 'AUTO_ACKNOWLEDGE' then
    FAckMode := amAutoAcknowledge
  else if Value = 'SESSION_TRANSACTED' then
    FAckMode := amTransactional
end;

function TConsumerTool.TargetType: string;
begin
  if Topic then
    Result := 'topic'
  else
    Result := 'queue';
end;

function TConsumerTool.DurableString: string;
begin
  if Durable then
    Result := 'durable'
  else
    Result := 'non-durable';
end;

procedure TConsumerTool.OnMessage(const Message: IMessage);
var
  TxtMsg: ITextMessage;
  Msg: string;
begin
  try
    try
      if Supports(Message, ITextMessage, TxtMsg) then
        begin
          if Verbose then
            begin
              Msg := TxtMsg.Text;
              if Length(Msg) > 50 then
                Msg := Copy(Msg, 1, 50) + '...';
              WriteLn('Received: ' + Msg);
            end;
          end
        else
          begin
            if Verbose then
              WriteLn('Received: Message');
          end;
        end;
      end;
    end;
  end;
```

```
    if Message.JMSReplyTo <> nil then
    begin
        ReplyProducer.Send(Message.JMSReplyTo,
            Session.CreateTextMessage('Reply: ' + Message.JMSMessageID));
    end;

    if Transacted then
        Session.Commit
    else if FAckMode = amClientAcknowledge then
        Message.Acknowledge;

except
    on E: Exception do
    begin
        WriteLn(E.Message);
    end;

end;
finally
    if SleepTime > 0 then
    begin
        Sleep(SleepTime);
    end;
end;
end;

procedure TConsumerTool.ConsumeMessagesAndClose(const Conn: IConnection; const
    Session:
    ISession; const Consumer: IMessageConsumer);
var
    I: Integer;
    Message: IMessage;
begin
    WriteLn('We are about to wait until we consume: ' + IntToStr(MaximumMessages)
        + ' message(s) then we will shutdown');

    I := 0;
    while (I < MaximumMessages) and Running do
    begin
        Message := Consumer.Receive(1000);
        if Message <> nil then
        begin
            Inc(I);
            OnMessage(Message);
        end;
    end;

    WriteLn('Closing connection');
    Consumer.Close;
    Session.Close;
    Conn.Close;
    if PauseBeforeShutdown then
    begin
        WriteLn('Press return to shut down');
```

```
    ReadLn;
  end;
end;

procedure TConsumerTool.ConsumeMessagesAndClose(const Conn: IConnection; const
  Session:
  ISession; const Consumer: IMessageConsumer; const Timeout: Integer);
var
  Message: IMessage;
begin
  WriteLn('We will consume messages while they continue to be delivered within: '
    + IntToStr(Timeout) + ' ms, and then we will shutdown');

  Message := Consumer.Receive(Timeout);
  while (Message <> nil) do
  begin
    OnMessage(Message);
    Message := Consumer.Receive(Timeout);
  end;

  WriteLn('Closing connection');
  Consumer.Close;
  Session.Close;
  Conn.Close;
  if PauseBeforeShutdown then
  begin
    WriteLn('Press return to shut down');
    ReadLn;
  end;
end;

end;

procedure TConsumerTool.Run;
var
  ConnectionFactory: IConnectionFactory;
  Connection: IConnection;
  Destination: IDestination;
  LibraryInfoProvider: IClientLibraryInfoProvider;
  LibInfo: IClientLibraryInfo;
  {$IFDEF VER130}
  Dummy: IDestination;
  {$ENDIF}
begin
  TCommandLineSupport.Configure(Self);

  Running := True;

  ConnectionFactory := TBTJMSConnectionFactory.Create(User, Password, URL);
  if Supports(ConnectionFactory, IClientLibraryInfoProvider, LibraryInfoProvider) then
  begin
    LibInfo := LibraryInfoProvider.ClientLibraryInfo;
    WriteLn(LibInfo.LibraryName + ' ' + LibInfo.LibraryVersion
      + ' ' + LibInfo.LibraryCopyright);
  end;

  WriteLn('Connecting to URL: ' + URL);
```

```
WriteLn('Consuming ' + TargetType + ': ' + Subject);
WriteLn('Using a ' + DurableString + ' subscription');

Connection := ConnectionFactory.CreateConnection;
if (Durable and (ClientId <> '')) then
begin
    Connection.ClientID := ClientId;
end;
Connection.Start;

// Create the session.
Session := Connection.CreateSession(Transacted, FAckMode);

// Create the Producer for the Destination.
if Topic then
    Destination := Session.CreateTopic(Subject)
else
    Destination := Session.CreateQueue(Subject);

ReplyProducer := Session.CreateProducer({$IFDEF VER130}Dummy{$ELSE}nil{$ENDIF});
ReplyProducer.SetDeliveryMode(dmNonPersistent);

if (Durable and Topic) then
    Consumer := Session.CreateDurableSubscriber(ITopic(Destination),
        ConsumerName)
else
    Consumer := Session.CreateConsumer(Destination);

if MaximumMessages > 0 then
begin
    ConsumeMessagesAndClose(Connection, Session, Consumer);
end
else
begin
    if ReceiveTimeOut = 0 then
    begin
        Consumer.SetMessageListener(Self);
        while True do
            Sleep(10); // run forever
        end
    else
        ConsumeMessagesAndClose(Connection, Session, Consumer, ReceiveTimeOut);
    end;
end;

end.
end.
```

ProducerTool

The ProducerTool demo can be used to send messages to the broker. It is based on the Java example class `ProducerTool.java` in the ActiveMQ binary distribution.

It is configurable by command line parameters, all are optional.

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

Table 7: 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
```

ProducerTool Source Code

```
unit ProducerToolUnit;

interface

uses
  BTJMSInterfaces;

type
{$M+}
  TProducerTool = class(TObject)
  private
    FURL: string;
    FMessageSize: Integer;
    FTopic: Boolean;
    FSubject: string;
    FPersistent: Boolean;
    FSleepTime: Integer;
    FTimeToLive: Integer;
    FMessageCount: Integer;
    FTransacted: Boolean;
    FVerbose: Boolean;
    FPassword: string;
    FUser: string;

    function TargetType: string;
    function PersistentString: string;

    procedure SendLoop(const Session: ISession;
      const Producer: IMessageProducer);

  public
    constructor Create;

    procedure Run;

  published
    property MessageCount: Integer read FMessageCount write FMessageCount;
    property MessageSize: Integer read FMessageSize write FMessageSize;
    property Password: string read FPassword write FPassword;
    property Persistent: Boolean read FPersistent write FPersistent;
    property SleepTime: Integer read FSleepTime write FSleepTime;
    property Subject: string read FSubject write FSubject;
    property TimeToLive: Integer read FTimeToLive write FTimeToLive;
    property Topic: Boolean read FTopic write FTopic;
    property Transacted: Boolean read FTransacted write FTransacted;
    property URL: string read FURL write FURL;
    property User: string read FUser write FUser;
    property Verbose: Boolean read FVerbose write FVerbose;

  end;
end;
```

```
implementation

uses
  CommandLineSupport,
  BTCommAdapterIndy, BTJMSConnection, BTJMSConnectionFactory, BTMgmtInterfaces,
  {$IFDEF VER130}
  Windows, // Sleep
  {$ELSE}
  StrUtils,
  {$ENDIF}
  SysUtils;

{ TProducerTool }

constructor TProducerTool.Create;
begin
  MessageCount := 10;
  MessageSize := 255;
  Password := BTJMSConnectionFactory.DEFAULT_PASSWORD;
  Subject := 'TOOL.DEFAULT';
  URL := BTJMSConnectionFactory.DEFAULT_BROKER_URL;
  User := BTJMSConnectionFactory.DEFAULT_USER;
  Verbose := True;
end;

function TProducerTool.TargetType: string;
begin
  if Topic then
    Result := 'topic'
  else
    Result := 'queue';
end;

function TProducerTool.PersistentString: string;
begin
  if Persistent then
    Result := 'persistent'
  else
    Result := 'non-persistent';
end;

procedure TProducerTool.Run;
var
  ConnectionFactory: IConnectionFactory;
  Connection: IConnection;
  Session: ISession;
  Destination: IDestination;
  Producer: IMessageProducer;
  LibraryInfoProvider: IClientLibraryInfoProvider;
  LibInfo: IClientLibraryInfo;
begin
  TCommandLineSupport.Configure(Self);

  ConnectionFactory := TBTJMSConnectionFactory.Create(User, Password, URL);
  if Supports(ConnectionFactory, IClientLibraryInfoProvider, LibraryInfoProvider) then
    begin
```

```
    LibInfo := LibraryInfoProvider.ClientLibraryInfo;
    WriteLn(LibInfo.LibraryName + ' ' + LibInfo.LibraryVersion
      + ' ' + LibInfo.LibraryCopyright);
  end;

  WriteLn('Connecting to URL: ' + URL);
  WriteLn('Publishing a Message with size ' + IntToStr(MessageSize) + ' to ' +
    TargetType + ': ' + Subject);
  WriteLn('Using ' + PersistentString + ' messages');
  WriteLn('Sleeping between publish ' + IntToStr(SleepTime) + ' ms');
  if TimeToLive <> 0 then
  begin
    WriteLn('Messages time to live ' + IntToStr(TimeToLive) + ' ms');
  end;

  Connection := ConnectionFactory.CreateConnection;
  Connection.Start;

  // Create the session.
  Session := Connection.CreateSession(Transacted, amAutoAcknowledge);

  // Create the Producer for the Destination.
  if Topic then
    Destination := Session.CreateTopic(Subject)
  else
    Destination := Session.CreateQueue(Subject);

  // Create the producer.
  Producer := Session.CreateProducer(Destination);

  if Persistent then
    Producer.DeliveryMode := dmPersistent
  else
    Producer.DeliveryMode := dmNonPersistent;

  if TimeToLive <> 0 then
    Producer.TimeToLive := TimeToLive;

  SendLoop(Session, Producer);

  Connection.Close;

  WriteLn('Done.');
```

```
end;

procedure TProducerTool.SendLoop(const Session: ISession;
  const Producer: IMessageProducer);
var
  I: Integer;
  TextMessage: ITextMessage;
  Msg: string;

  {$IFDEF VER130}
  function DupeString(const AText: string; ACount: Integer): string;
  var
```

```
    i,l: Integer;
begin
  Result := '';
  if ACount>=0 then
  begin
    l := Length(atext);
    SetLength(Result, aCount*l);
    for i:=0 to ACount-1 do
      Move(AText[l], Result[l*i+1],l);
    end;
  end;
end;
{$ENDIF}

function CreateMessageText(const Index: Integer): string;
begin
  Result := 'Message: ' + IntToStr(Index) + ' sent at: ' + DateTimeToStr(Now);

  if Length(Result) > MessageSize then
    Result := Copy(Result, 1, MessageSize)
  else
    Result := Copy(Result + DupeString(' ', MessageSize), 1, MessageSize);
  end;
end;

begin
  for I := 0 to MessageCount - 1 do
  begin
    Msg := CreateMessageText(I);
    TextMessage := Session.CreateTextMessage(Msg);
    if Verbose then
    begin
      if Length(Msg) > 50 then
      begin
        Msg := Copy(Msg, 1, 50) + '...';
      end;
      WriteLn('Sending message: ' + Msg);
    end;
    Producer.Send(TextMessage);
    if Transacted then
    begin
      Session.Commit;
    end;
    Sleep(SleepTime);
  end;
end;

end.
```

Throughput Test Tool

This example application is configurable by command line parameters, all are optional.

| Parameter | Default Value | Description |
|-----------------|---------------|---------------------|
| Password | | Password |
| Subject | TOOL.DEFAULT | queue or topic name |
| URL | localhost | server url |
| User | | user name |

Table 8: Throughput Test Tool Command Line Options

Examples

Use remote broker 'amq56' and specify user and password

```
tptest --url=stomp://amq56 --user=test1 --password=secret
```

Broker Statistics Example

Introduction

ActiveMQ supports Broker plugins, which allows the default functionality to be extended, and new with version 5.3 of Apache ActiveMQ is a Statistics plugin, which enables statistics about the running broker, or Queues and Topics to be queried.

The statistics plugin looks for messages sent to particular destinations. To query the running statistics of a the message broker, send an empty message to a Destination (Queue or Topic) named `ActiveMQ.Statistics.Broker`, and set the `JMSReplyTo` field with the Destination you want to receive the result on. The statistics plugin will send a `IMapMessage` filled with the statistics for the running ActiveMQ broker.

Similarly, if you want to query the statistics on a Destination, send a message to the Destination name, prepended with `ActiveMQ.Statistics.Destination`. For example, to retrieve the statistics on a Queue named `test.foo` send an empty message to the Queue `ActiveMQ.Statistics.DestinationTest.Foo`.

You can also use wildcards too, and receive a separate message for every destination matched.

Configuration

To configure ActiveMQ to use the statistics plugin, add the following to the ActiveMQ XML configuration:

```
...
<plugins>
  <statisticsBrokerPlugin/>
</plugins>
...
```

Source Code

```
program DestStatistics;
{$APPTYPE CONSOLE}
```

```
uses
  SysUtils,
  BTCommAdapterIndy, BTMessageTransformerXMLMapOmni,
  BTJMSInterfaces, BTJMSConnection, BTSessionIntf, BTSerialIntf,
  BTStompTypes, BTTypes,
  Classes;

var
  Connection: IConnection;
  Session: ISession;
  Producer: IMessageProducer;
  Consumer: IMessageConsumer;
  Destination, ReplyQueue: IQueue;
  JMSMessage: ITextMessage;
  Reply: IMapMessage;
  MapNames: PMStrings;
  I: Integer;
  Key: string;

begin
  Connection := TBTJMSConnection.MakeConnection;
  try
    try
      // Create and assign the message transformer
      SetTransformer(Connection, TBTMessageTransformerXMLMapOmni.Create(nil));

      Connection.Start;
      Session := Connection.CreateSession(False, amAutoAcknowledge);

      // listen on reply queue
      ReplyQueue := Session.CreateQueue('Habari' + '?' +
        BTStompTypes.SH_TRANSFORMATION + '=' + TRANSFORMER_ID_MAP_XML);
      Consumer := Session.CreateConsumer(ReplyQueue);

      // see: https://issues.apache.org/activemq/browse/AMQ-2379
      // see: http://rajdavies.blogspot.com/2009/10/query-statistics-for-apache-
      activemq.html
      // You can also use wildcards too, and receive a separate message for every
      destination matched.

      // create the pseudo destination
      if ParamCount = 0 then
        begin
          Destination := Session.CreateQueue('ActiveMQ.Statistics.Broker');
        end
      else
        begin
          Destination := Session.CreateQueue('ActiveMQ.Statistics.Destination' +
ParamStr(1));
        end;

      // display destination name
      WriteLn('Request statistics for ' + Destination.QueueName + ' ...');

      // create the message and set reply queue name
      JMSMessage := Session.CreateTextMessage;
```

```
JMSMessage.JMSReplyTo := ReplyQueue;
Producer := Session.CreateProducer(Destination);
Producer.Send(JMSMessage);

// read the result message
Reply := Consumer.Receive(3000) as IMapMessage;

// list the map key/values
if Assigned(Reply) then
begin
  MapNames := Reply.GetMapNames;
  for I := 0 to Length(MapNames) - 1 do
  begin
    Key := MapNames[I];
    WriteLn(Key + '=' + Reply.GetString(Key));
  end;
end
else
  WriteLn('Received no message on queue ' + ReplyQueue.QueueName);

Connection.Stop;

except
  on E: Exception do
    WriteLn(E.Message);
  end;
finally
  Connection.Close;
end;

WriteLn('Press any key');
ReadLn;
end.
```

Example Output

When launched with parameter example.A, the demo application activemq-statistics will retrieve the information for queue example.A, and the output would look similar to this:

```
Request statistics for ActiveMQ.Statistics.Destinationexample.A ...
memoryUsage=0
dequeueCount=0
inflightCount=0
messagesCached=0
averageEnqueueTime=0.0
destinationName=queue://example.A
size=0
memoryPercentUsage=0
producerCount=0
consumerCount=1
minEnqueueTime=0.0
```

```
maxEnqueueTime=0.0
dispatchCount=0
expiredCount=0
enqueueCount=0
memoryLimit=67108864
Press any key
```

Without a parameter, broker statistics will be returned:

```
Request statistics for ActiveMQ.Statistics.Broker ...
vm=vm://localhost
memoryUsage=0
storeUsage=66434225
tempPercentUsage=0
openwire=tcp://mj-PC:61616
brokerId=ID:mj-PC-52958-1272975061672-0:0
consumerCount=3
brokerName=localhost
expiredCount=0
dispatchCount=2
maxEnqueueTime=3.0
storePercentUsage=0
dequeueCount=2
inflightCount=0
messagesCached=0
tempLimit=107374182400
averageEnqueueTime=1.5
memoryPercentUsage=0
size=0
tempUsage=0
producerCount=0
minEnqueueTime=0.0
dataDirectory=C:\Java\apache-activemq-5.3.1\data
enqueueCount=64
stomp=stomp://mj-PC:61613?transport.closeAsync=false
storeLimit=107374182400
memoryLimit=67108864
Press any key
```

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.

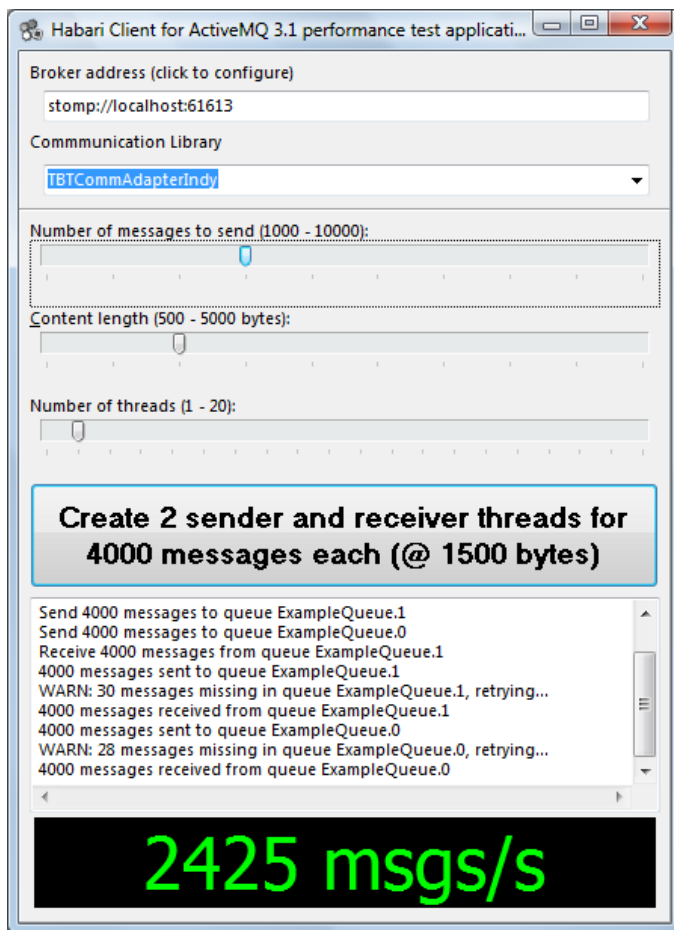


Illustration 6: Performance Test Application

PHP Producer Demo

The demo folder includes a simple demo project for data exchange between a PHP application and Delphi. It sends a message to the queue `TOOL.DEFAULT` every time when the web client sends a request to the `index.php` page.

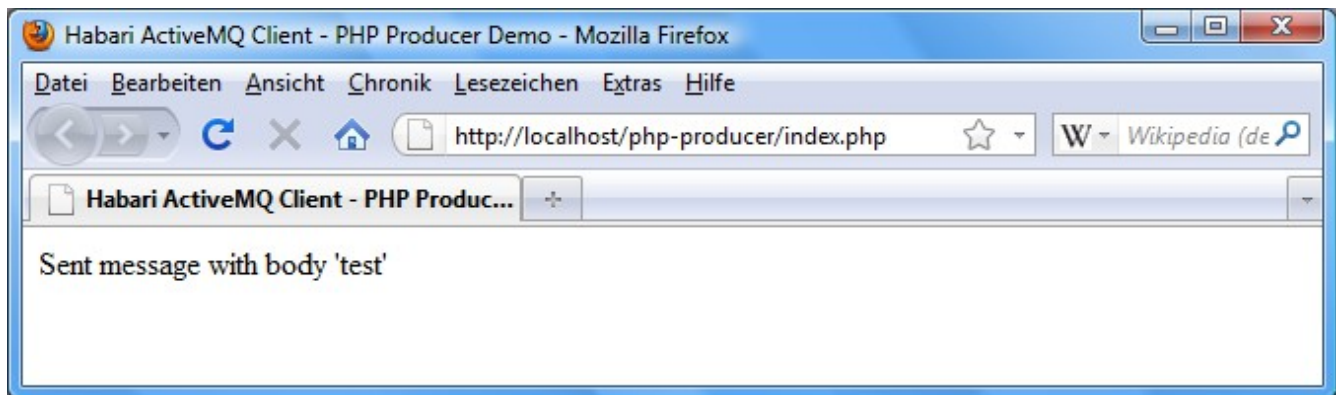


Illustration 7: PHP Producer Demo – Web Interface

The Delphi ConsumerTool demo application displays the incoming messages on the `TOOL.DEFAULT` queue.

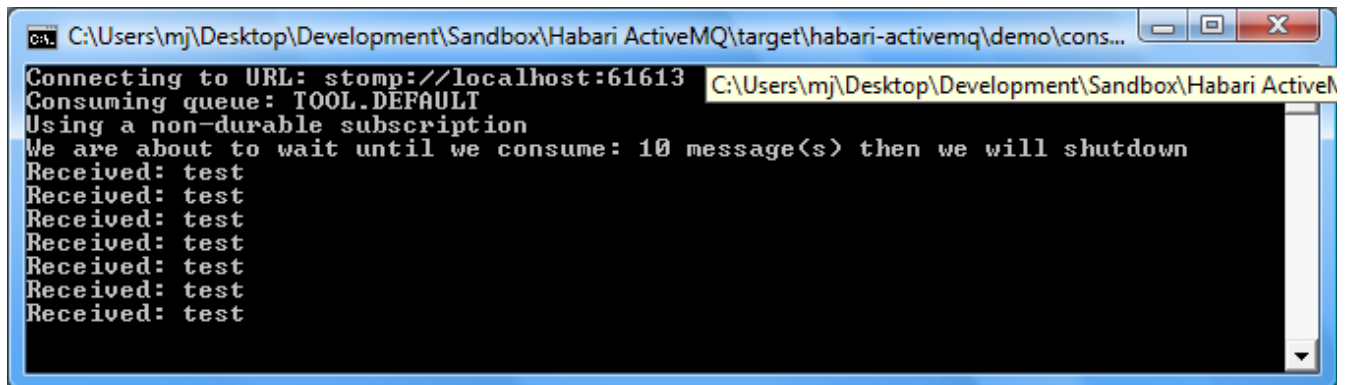


Illustration 8: PHP Producer Demo - Incoming Messages

Requirements

- Web Server with PHP 5

- PHP Stomp client library from <http://stomp.fusesource.org/>
- Apache ActiveMQ 5.3
- ConsumerTool demo application

Steps

- install index.php and Stomp PHP client in htdocs/php-producer
- run ActiveMQ and web server on the same computer
- run ConsumerTool demo application
- use a web browser to navigate to <http://localhost/php-producer/index.php>

Source Code

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Habari Client for ActiveMQ - PHP Producer Demo</title>
  </head>
  <body>
    <?php
// include Stomp library
    require_once("Stomp.php");
// make a connection
    $con = new Stomp("tcp://127.0.0.1:61613");
// connect
    $con->connect();
// send a message to the queue
    $con->send("/queue/TOOL.DEFAULT", "test");
    echo "Sent message with body 'test'\n";
// disconnect
    $con->disconnect();
    ?>
  </body>
</html>
```

Object Exchange with JSON

SuperObject Example Applications

The `demo/obex-superobject` projects **JSONPublisher** and **JSONSubscriber** show how object messages can be exchanged between two Delphi applications, using the `TBTMessageTransformerJSONSuperObject` transformer class.

- **JSONPublisher** sends 5 object messages to the Queue `TOOL.OBJECT.JSON`
- **JSONSubscriber** receives messages from Queue `TOOL.OBJECT.JSON` until the timeout is hit (1000 msec)

JSONPublisher Source Code

```
program JSONPublisher;

{$APPTYPE CONSOLE}

uses
  superobject,
  BTMessageTransformerJSONSuperObject,
  BTCommAdapterIndy, BTJMSConnection, BTSerialIntf, BTJMSInterfaces,
  BTStompTypes,
  SysUtils;

const
  Dest = 'TOOL.OBJECT.JSON';

const
  EXPECTED = 5;

var
  Connection: IConnection;
  Session: ISession;
  Destination: IDestination;
  ObjectMessage: IObjectMessage;
  Producer: IMessageProducer;
  I: Integer;
  Wrapper: TSuperObjectWrapper;

begin
  WriteLn('Connect to server');
  Connection := TBTJMSConnection.MakeConnection;

  try
```

```
try
  SetTransformer(Connection, TBTMessageTransformerJSONSuperObject.Create(nil));

  Connection.Start;

  Session := Connection.CreateSession(False, amClientAcknowledge);
  Destination := Session.CreateQueue(Dest);
  Producer := Session.CreateProducer(Destination);

  for I := 0 to EXPECTED - 1 do
  begin
    Wrapper := TSuperObjectWrapper.Create;
    try
      try
        Wrapper.SuperObject.S['name'] := 'myname ...';

        ObjectMessage := Session.CreateObjectMessage(Wrapper);
        ObjectMessage.SetStringProperty(SH_TRANSFORMATION + '-custom',
          TRANSFORMER_ID_OBJECT_JSON);
        Producer.Send(ObjectMessage);

        WriteLn('Content: ' + WideString(ObjectMessage.Content));
        WriteLn(SH_TRANSFORMATION + '-custom: ',
          ObjectMessage.GetStringProperty(SH_TRANSFORMATION + '-custom'));
      except
        on E: Exception do
          begin
            WriteLn(E.Message);
          end
        end;
      finally
        Wrapper.Free;
      end;
    end;
  finally
    Connection.Close;
    WriteLn('Hit any key');
    ReadLn;
  end;
except
  on E: Exception do
  begin
    WriteLn(E.Message);
    ReadLn;
  end;
end;
end.
```

Delay and Schedule Message Delivery

Introduction

Apache ActiveMQ from version 5.4 has a persistent scheduler built into the ActiveMQ message broker. An ActiveMQ client can take advantage of a delayed delivery by using message properties.²²

By setting properties of the JMS message, a client can

- set the time in milliseconds that a message will wait before being scheduled to be delivered by the broker
- set the time in milliseconds to wait after the start time to wait before scheduling the message again
- set the number of times to repeat scheduling a message for delivery
- or use a Cron entry (for example "0 * * * *" to set the schedule)

Example

The example application shows how a message can be scheduled for delivery after 5 seconds.

```
program ScheduledMsgs;

{$APPTYPE CONSOLE}

uses
  BTCommAdapterIndy, BTJMSInterfaces, BTJMSConnection,
  SysUtils, Windows;

const
  // ScheduledMessage.
  AMQ_SCHEDULED_DELAY = 'AMQ_SCHEDULED_DELAY';

var
  Connection: IConnection;
  Session: ISession;
  Producer: IMessageProducer;
  Consumer: IMessageConsumer;
  Destination, ReplyQueue: IQueue;
  JMSMessage: ITextMessage;
  Started, Elapsed: Int64;

begin
  WriteLn('Connect ...');
```

²² <http://activemq.apache.org/delay-and-schedule-message-delivery.html>

```
Connection := TBTJMSConnection.MakeConnection;
try
  try
    Connection.Start;
    Session := Connection.CreateSession(False, amAutoAcknowledge);

    // listen for messages
    ReplyQueue := Session.CreateQueue('Habari');
    Consumer := Session.CreateConsumer(ReplyQueue);

    // create the destination
    Destination := Session.CreateQueue('Habari');

    // create the message and set delay to 5 seconds
    JMSMessage := Session.CreateTextMessage('test msg');
    JMSMessage.SetIntProperty(AMQ_SCHEDULED_DELAY, 5000);
    Producer := Session.CreateProducer(Destination);

    Producer.Send(JMSMessage);
    Started := GetTickCount;

    WriteLn('Message has been sent to queue ' + Destination.QueueName);

    // wait for the delayed message
    JMSMessage := Consumer.Receive(6000) as ITextMessage;
    Elapsed := GetTickCount - Started;

    if Assigned(JMSMessage) then
      begin
        WriteLn(Format('Message text: %s (after %d msec)', [JMSMessage.Text,
          Elapsed]));
      end
    else
      WriteLn('Received no message on queue ' + ReplyQueue.QueueName);

    Connection.Stop;

  except
    on E: Exception do
      WriteLn(E.Message);
    end;
  finally
    Connection.Close;
  end;

  WriteLn('Press any key');
  ReadLn;
end.
```

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.²³

JMS properties for outgoing messages

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

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

JMS properties for incoming messages

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

| | |
|-------------------------|-------------------------------------|
| JMSCorrelationID | The correlation ID for the message. |
| JMSExpiration | The message's expiration value. |

²³ <http://download.oracle.com/javaee/5/api/javax/jms/Message.html>

| | |
|------------------------|---|
| JMSDeliveryMode | Whether or not the message is persistent. |
| JMSPriority | The message priority level. |
| JMSTimestamp | The timestamp the broker added to the message. |
| JMSMessageId | The message ID which is set by the provider. |
| JMSReplyTo | The Destination object to which a reply to this message should be sent. |

Stomp extensions for JMS message semantics

JMSXGroupID

This message header specifies the [Message Group](#)²⁴.

Message groups are an enhancement to the [Exclusive Consumer](#) feature to provide

- guaranteed ordering of the processing of related messages across a single queue
- load balancing of the processing of messages across multiple consumers
- high availability / auto-fail-over to other consumers if a JVM goes down

A way of explaining Message Groups is that it provides sticky load balancing of messages across consumers; where the JMSXGroupID is like a HTTP session ID or cookie value and the message broker is acting like a HTTP load balancer.

JMSXGroupSeq

Optional header that specifies the sequence number in the [Message Group](#) .

Note

This Stomp extension in ActiveMQ has not yet been tested with Habari Client for ActiveMQ to verify it is working as expected. If you would like to see some Delphi examples included in the demos or documentation please feel free to contact me.

²⁴ <http://activemq.apache.org/message-groups.html>

User Defined Properties

Supported Data Types

The Stomp protocol only supports string type properties. Apache ActiveMQ 5.6 introduced support for numeric expressions in message selectors (see <https://issues.apache.org/jira/browse/AMQ-1609>).

Reserved Names

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

- activemq.* (everything starting with activemq is a reserved name)
- 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.

Temporary Queues

Introduction

“Temporary destinations ([temporary queues](#) or [temporary topics](#)) are proposed as a lightweight alternative in a scalable system architecture that could be used as unique destinations for replies. Such destinations have a scope limited to the connection that created it, and are removed on the server side as soon as the connection is closed.” (“Designing Messaging Applications with Temporary Queues”, by Thakur Thribhuvan ²⁵)

Support in Habari Client for ActiveMQ

Habari ActiveMQ 2.3 introduced support for `Session#createTemporaryQueue` and `Session#createTemporaryTopic`.

Resource Management

The session should be closed as soon as processing is completed so that `TemporaryQueues` will be deleted on the server side.

How should I implement request response with JMS?

The ActiveMQ documentation presents a resource-friendly solution for request-response style communication, which uses the `JMSCorrelationID` message property, including example source. The article can be found here:

<http://activemq.apache.org/how-should-i-implement-request-response-with-jms.html>

²⁵ <http://onjava.com/pub/a/onjava/2007/04/10/designing-messaging-applications-with-temporary-queues.html>

Online Tutorials

Habari Client for ActiveMQ with Geronimo 2.2 and Eclipse

This tutorial uses Eclipse 3.5 (Galileo) and Geronimo with the embedded ActiveMQ 5.3 message broker to exchange messages between a Java web application and Delphi code.

<https://mikejustin.fogbugz.com/default.asp?W15>

Using Habari Client for ActiveMQ with the Geronimo 2.2 application server

This tutorial provides a very simple and quick introduction to the NetBeans IDE workflow by walking you through the creation of a simple Java messaging client application.

<https://mikejustin.fogbugz.com/default.asp?W14>

Using Habari Client for ActiveMQ with the Geronimo application server

This tutorial will guide you through the creation of a simple web application for Apache Geronimo which uses a Servlet to send messages to the embedded ActiveMQ broker.

<https://mikejustin.fogbugz.com/default.asp?W9>

Using Habari Client for ActiveMQ with FUSE Message Broker

This tutorial provides a quick introduction to the installation of FUSE Message Broker and communication tests with Habari Client for ActiveMQ.

<https://mikejustin.fogbugz.com/default.asp?W7>

Unit Tests

DUnit Tests

Unit Tests Location

The source distribution of Habari Client for ActiveMQ includes all DUnit tests in

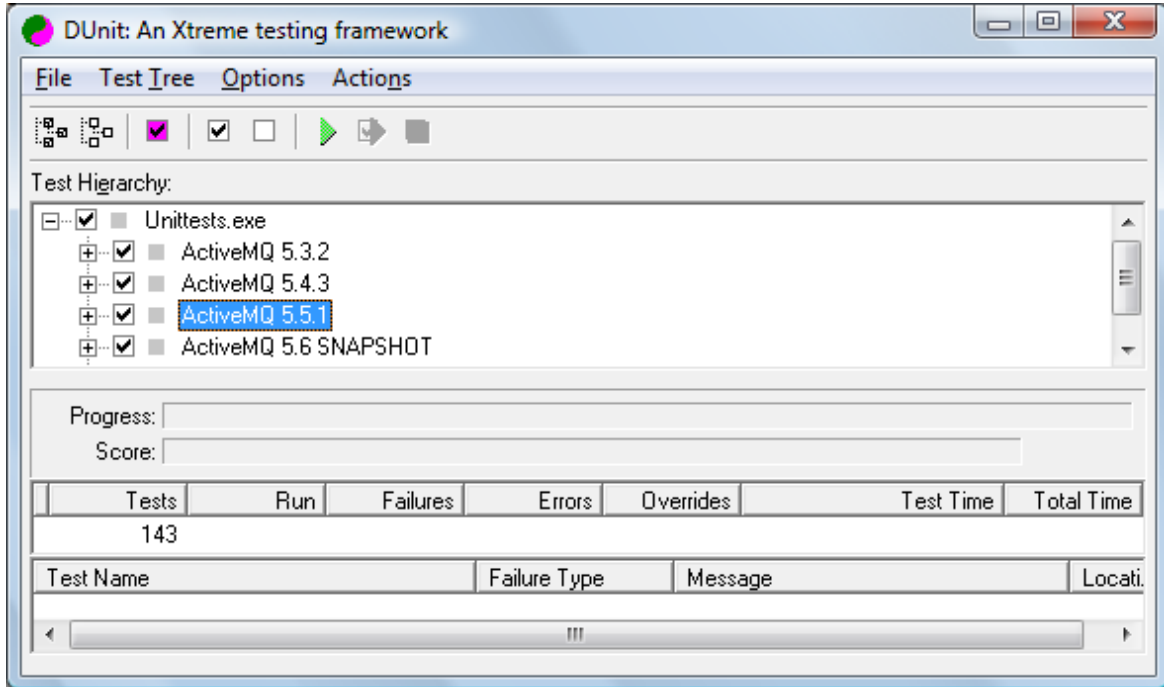
- demo\unittests (tests suites for different broker versions, see next section)
- demo\common-tests (common tests for all Habari Client libraries)

Broker Autostart

If the DUnit tests are started with /autolaunch, the tests will start and stop the ActiveMQ broker automatically. The binary distributions of the ActiveMQ message broker versions need to be installed in the directories

- \Java\apache-activemq-5.3.3
- \Java\apache-activemq-5.4.3
- \Java\apache-activemq-5.5.1

Illustration 9: DUnit GUI Testrunner



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

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

Internet Direct (Indy) Communication Adapter

Text message size limitation

To support text messages with more than 16 kB size, the Indy IOHandler property `MaxLineLength` needs to be set to a higher value.

For your convenience, the `ReadMessageBuffer` function in `BTCommAdapterIndy.pas` already includes a line which can be uncommented. This example source code line will allow for a message size of up to 1 MB:

```
// uncomment this line to support text messages with up to 1 MB size
// IOHandler.MaxLineLength := 1024 * 1024;
```

Note: only text messages and object messages, which are internally handled as text, are affected by this limitation.

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.

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.

Multi Threading

The unit test suite includes multi threading tests, but there is no guarantee for error-free operation of the library in applications which make extensive use of multi threading.

A session supports transactions and it is difficult to implement transactions that are multithreaded; a session should not be used concurrently by multiple threads.

BytesMessage and stomp+nio

With ActiveMQ 5.3.1, 5.3.2 and 5.4.0, a BytesMessage can not be received over the stomp+nio transport.

Fixed in ActiveMQ 5.4.1

This issue has been fixed in ActiveMQ 5.4.1. For details see issue 2883 in the bug tracker:

<https://issues.apache.org/activemq/browse/AMQ-2883>

Default Priority

The JMS specification defines that the default priority value for a MessageProducer is 4.

Habari Client for ActiveMQ adds a priority header to Stomp frames only if the message producer has a non-default priority. This has the advantage that the frame only has a priority header when it is necessary, to keep the message header small. However, Apache ActiveMQ broker versions below 5.5 use the value 0 (lowest) instead of 4 as default priority for incoming Stomp messages.

Starting with Apache ActiveMQ 5.5, the broker handles a missing priority header correctly.²⁶

²⁶ <https://issues.apache.org/jira/browse/AMQ-3006>

References

Message Broker

| | |
|-----------------|---|
| Apache ActiveMQ | http://activemq.apache.org |
| FUSE | http://fusesource.com/products/enterprise-activemq/ |

IDE

| | |
|--------------------|---|
| Embarcadero Delphi | http://www.embarcadero.com/delphi |
| Free Pascal | http://freepascal.org |
| Lazarus | http://www.lazarus.freepascal.org |

JMS

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

JSON

| | |
|-------------|---|
| IkJSON | http://sourceforge.net/projects/ikjson |
| SuperObject | http://www.progdigy.com |

Stomp

| | |
|--------------|---|
| In ActiveMQ | http://activemq.apache.org/stomp.html |
| Project home | http://stomp.codehaus.org/ |

Communication

| | |
|------------------------|---|
| Synapse | http://www.synapse.ararat.cz |
| Internet Direct (Indy) | http://www.indyproject.org |
| Indy Snapshot | http://indy.fulgan.com/ZIP |

Logging

| | |
|-------|---|
| Log4D | http://log4d.sourceforge.net/ |
|-------|---|

XML

| | |
|-----------|---|
| NativeXml | http://www.simdesign.nl/xml.html |
| OmniXML | http://www.omnixml.com/ |
| XStream | http://xstream.codehaus.org/ |

Habari Client for ActiveMQ License

Habari Client for ActiveMQ (c) 2008-2012 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.

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 3.1

Released May 8, 2012

New

- amClientIndividual** The library supports a proprietary acknowledge mode, "Client Individual", to acknowledge a single message only instead of all received messages. This mode is useful in multi-threaded client applications which consume messages from the same queue when there is no guarantee on the order in which message will be processed and acknowledged. To create a Session with this mode, use `Connection.CreateSession(False, amClientIndividual)`. This acknowledge mode requires ActiveMQ 5.2 or newer
- 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

- ActiveMQ** Tested with ActiveMQ 5.3.2, 5.4.3, 5.5.1 and 5.6.0 (released May 7, 2012)
- ReceiveNoWait** The `ReceiveNoWait` method has been improved 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** The visual memory manager monitor for FastMM (VisualMM) demo allows to choose from a list of running processes

| | |
|------------------------|---|
| Throughput test | The throughput test 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 |

Version 3.0

Released December 27, 2011

New

| | |
|----------------------|--|
| STOMP 1.1 | Experimental support for STOMP 1.1 |
| Queue Browser | The CreateBrowser function of a session object returns an IQueueBrowser, which can be used to iterate messages in a queue without consuming them (requires STOMP 1.1) |
| Heartbeat | Experimental support of client-side and server-side heartbeat signals (requires STOMP 1.1), new methods SendHeartbeat and CheckHeartbeat (Delphi on Windows platform only as GetTickCount64 is not implemented in FPC version) |
| ACK frame | The subscription id is included in ACK frames (only with Stomp 1.1 enabled, where this id is required) |
| ActiveMQ | Tested with ActiveMQ 5.3.2, 5.4.3, 5.5.1 and 5.6-SNAPSHOT |
| BTPlatform | The new unit BTPlatform provides the GetTickCount64 function. |

Changed

| | |
|-----------------------|--|
| Indy rev. 4713 | Tested with revision 4713 of Indy 10.5.8 |
| 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 |
| Demo | The GUI demo allows to save the file which is in an incoming message. |

Version 2.9

Released September 6, 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 |
| ActiveMQ | Tested with ActiveMQ 5.3.2, 5.4.2, 5.5.0 and 5.6-SNAPSHOT |
| Numeric Expressions | Tested Apache ActiveMQ 5.6-SNAPSHOT support for numeric expressions in JMS selectors |
| 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) |
| DUnit tests | Fixed a memory leak in the test case for BTQueueRequestor |
| ReadLineTimeout | The Indy adapter uses IOHandler.ReadLnTimeout to detect a timeout after read |
| Indy 9 | Experimental support for Indy 9 has been removed |

Version 2.8

Released June 7, 2011

New

| | |
|------------------------|--|
| Object Exchange | Support for "Delphi Only" and "Cross-Language" Object Exchange |
| NativeXml | Support for the NativeXml open source XML parser library, classes TBTMessageTransformerXMLNative for IObjectMessage and TBTMessageTransformerXMLMapNative for IMapMessage exchange |
| Transformer | A helper method, SetTransformer, can be used to set a IMessageTransformer on a IConnection |
| Unit Tests | New unit ObjectExchangeTests for IObjectMessage and IMapMessage object exchange tests |
| ActiveMQ | Tested with ActiveMQ 5.3.2, 5.4.2 and 5.5 |
| Demos | Cleaned up and documented demo applications |

Changed

| | |
|-------------------------|--|
| Performance Demo | Shared code with Habari Client for ActiveMQ and Habari OpenMQ Client, display transfer speed in msgs/s |
| TransformationId | The client verifies if the transformation id of the message transformer matches the 'transformation' header of incoming object messages |
| TTransformable | Class TTransformable is deprecated |
| Refactoring | Refactored to new broker-independent units BTSerialIntf, BTSessionIntf instead of unit BTAMQInterfaces |
| Frame Decoder | The new unit BTStompDecoder contains a new Stomp Frame decoder implementation, it can be enabled with the conditional symbol HABARI_USE_TBYTES |
| Indy 10.5.8 | Tested with revision 4639 of Indy 10 |
| FPC 2.4.4 | Build tested with Free Pascal 2.4.4 |
| Logging | Removed old unused logging units (BTLog etc) |
| ActiveMQTypes | A new unit ActiveMQTypes contains declarations which are specific for the broker. |
| Minimum Timeout | TBTCommAdapterIndy#ReadOneMessage uses a minimum timeout of 1000 msec for reading the message body to avoid partial message reads |

Version 2.7

Released March 1, 2011

New

- Failover transport** 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
- CreateMessage** Function Session#CreateMessage returns a IMessage object
- Durable Consumer** Method Session#Unsubscribe can be used to unsubscribe a durable subscription

Changed

- ReceiveNoWait** The internal method ReadOneMessageNoWait in the Indy adapter class TBTCCommAdapterIndy no longer uses CanReadOneMessage to check for data on the socket, now it calls ReadOneMessage with a timeout value of one millisecond
- ReadMessageBuffer** The internal method ReadMessageBuffer in TBTCCommAdapterIndy has a new parameter, ATimeout
- Log4D rev. 37** Log4D updated to revision 37
- FPC 2.4.2** Build tested with Free Pascal 2.4.2
- SuperObject rev. 39** SuperObject updated to revision 39
- ActiveMQ** Tested with ActiveMQ 5.3.2, 5.4.2 and 5.5-SNAPSHOT
- OnVerifyPeer** BTCommAdapterIndySSL updated to use the method signature of current Indy version with an additional AError parameter
- /autolaunch** If a command line parameter /autolaunch is specified, the DUnit test program will launch the ActiveMQ brokers 5.3.2, 5.4.2 and 5.5-SNAPSHOT automatically in the test suite setup and terminate the broker in the teardown stage
- GUI demo fix** Fixed an AV which occurred when the demo program was compiled without assertions
- Stomp constants** Stomp header constants are defined in unit BTStompTypes

Documentation

| | |
|-------------------------|--|
| Default Priority | Documented Default Priority handling fix (issue tracker id AMQ-3006) |
| Object Exchange | HabariActiveMQObjectExchange.pdf updated to Apache ActiveMQ 5.4.2 |

Version 2.6

Released December 7, 2010

New

| | |
|------------------------|--|
| IConnectionInfo | New interface in BtMgmtInterfaces which provides connection state information |
| ActiveMQ 5.4.2 | Tested with Apache ActiveMQ 5.4.2 final |
| Single Source | All versions of the Habari JMS Client library share the source code for the basic demo applications ConsumerTool, ProducerTool, DelphiGUI and HabariChat |

Fixed

| | |
|-------------------|--|
| Thread | Fixed compiler warning about deprecated Thread methods Resume and Suspend |
| Connection | Fixed code to avoid a EIdConnClosedGracefully exception in BTStompCustomClient |

Examples

| | |
|--------------------------|---|
| DelphiGUI logging | Use Log4D for logging |
| DelphiGUI dialog | Added a Connection Factory configuration dialog |
| DelphiGUI flicker | Reduced flickering of the Delphi GUI demo application |
| ExampleQueue | DelphiGUI Demo uses default name 'ExampleQueue' for the JMS destination |

- Log4D JMSAppender** Provided an example implementation of a JMS log appender for the open source Log4D logging framework (unit LogJMSAppender)
- Log4D demo** Added a demo application which uses the LogJMSAppender unit

Documentation

- xbean startup** Added information about broker startup using "xbean" Profiles
- request-response** Added chapter: "How should I implement request response with JMS?"
- Indy** Updated download information as Indy no longer uses the Tiburon branch
- stomp+nio fix** Added note about Apache 5.4.1 fix for BytesMessages and stomp+nio bug to the Known Limitations chapter
- default priority** Added note about the "Default Priority" handling by ActiveMQ brokers to the Known Limitations chapter

Version 2.5

Released October 14, 2010

New

- Delphi XE** Ready for Delphi XE
- Indy 10.5.8** Tested with Indy 10.5.8 for the Indy communication adapter
- ActiveMQ 5.4.1** New unit tests for Apache ActiveMQ 5.4.0 and 5.4.1 broker
- ActiveMQ 5.5** Tested with ActiveMQ 5.5 Snapshot

Changed

- SuperObject version** Updated to SuperObject 1.2.4 r33

Version 2.4

Released August 17, 2010

New

- | | |
|----------------------------|--|
| PHP demo | The demo folder includes a simple PHP example which sends messages to the TOOL.DEFAULT queue |
| ActiveMQ 5.3.2 | New unit tests for Apache ActiveMQ 5.3.2 broker |
| Library Information | The Connection Factory class now implements a new interface, IClientLibraryInfoProvider |
| Durable Subscriber | TBTSession now offers a second method signature for CreateDurableSubscriber |
| Log4D library | The Log4D logging library is now included and used when HABARI_LOGGING is defined |

Changed

- | | |
|-------------------------|--------------------------|
| BTQueueRequestor | New TimeOut property |
| doxygen | Updated to doxygen 1.7.1 |

Version 2.3

Released May 11, 2010

New

- | | |
|------------------------|--|
| Temporary Queue | The JMS API method Session#CreateTemporaryQueue is supported now. A ITemporaryQueue object is a unique IQueue object created for the duration of a IConnection. It is a system-defined queue that can be consumed only by the IConnection that created it. An example application is included in the demo/tempdest folder. |
| Temporary Topic | The JMS API method Session#CreateTemporaryTopic is supported now. A ITemporaryTopic object is a unique IQueue object created |

for the duration of a IConnection. It is a system-defined queue that can be consumed only by the IConnection that created it.

Queue Requestor

Class TBTQueueRequestor helper class simplifies making RPC-style service requests. The TBTQueueRequestor constructor is given a non-transacted IQueueSession and a destination IQueue. It creates a ITemporaryQueue for the responses and provides a Request method that sends the request message and waits for its reply.

Changed

DUnit tests

DUnit tests start and stop the ActiveMQ broker (version 5.3, 5.3.1 and 5.4-SNAPSHOT) in the SetUp and TearDown methods. Please see demo\unittests\TestSuites.pas for details.

Delay / Schedule

The documentation includes an introduction to new feature in ActiveMQ 5.4: Delay and Schedule Message Delivery

Broker Statistics

The documentation includes an introduction to new feature in ActiveMQ 5.3: Broker Statistics

Version 2.2

Released March 22, 2010

New

MapMessage

This version introduces basic support for MapMessage. It includes a XML message transformer (based on OmniXML) in the new unit BTMessageTransformerXMLMapOmni. Map items are only of type string in this release. For a demo application, see also 'Broker Statistics'.

Broker Statistics

A new demo in the activemq-statistics folder shows how broker and destination statistics can be retrieved (this feature requires ActiveMQ 5.3 or higher)

Geronimo 2.2

Three new online tutorials show how to exchange messages with the Apache Geronimo application server, using the Eclipse 3.5 or NetBeans 6.8 IDE. The complete projects are included in the new tutorials folder. See also the new section 'Online Tutorials' in the Getting Started Guide.

| | |
|------------------|--|
| Logging | Only if the HABARI_LOGGING conditional symbol is set, logging code will be included. This will reduce code size in production and also improve performance. |
| stomp+nio | Tested with Stomp over NIO transport. For stomp+nio transport configuration please see http://activemq.apache.org/stomp.html |
| FPC 2.5.1 | Tested with Free Pascal 2.5.1 (snapshot) |
| OmniXML | Tested with OmniXml release of 2010-01-03 |

Changed

| | |
|----------------------------|---|
| SuperObject version | Updated to SuperObject 1.2.4 |
| IKJSON | Updated to IKJSON 1.07 |
| Unsupported | Unsupported communication adapters have been removed |
| THabariExpress | The THabariExpress and THabariExpressAdmin components will no longer be registered in the IDE component palette |

Version 2.1

Released January 28, 2010

New

| | |
|---------------------|--|
| IPv6 support | The Indy and Synapse communication adapters now support Internet Protocol Version 6 addresses, for example <code>stomp://[2001:db8:85a3::8a2e:370:7334]:61613</code> |
|---------------------|--|

Changed

| | |
|----------------------------|---|
| SuperObject version | Updated to SuperObject 1.2.2 released Dec 22, 2009 |
| Linux | Tested on Ubuntu 9.10 Linux 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 |
| Memory Leak | Fixed a possible memory leak in TBTJMSTConnection |
| Exception handling | Improved exception handling when broker is down |
| Unicode | Fixed a bug in the Indy communication adapter which affected versions before Delphi 2009 |

Version 2.0

Released November 10, 2009

New

| | |
|---------------------|---|
| ActiveMQ 5.3 | This version passed all unit tests with ActiveMQ 5.3.0. For new features in ActiveMQ 5.3.0, please visit the release notes page http://activemq.apache.org/activemq-530-release.html |
| Synapse SSL | Support for SSL using Synapse is functional now. Please note that the source for BTCommAdapterSynapseSSL is still in the beta directory. A console app demo is in the folder sslsynapse. |

Changed

| | |
|---------------------------|--|
| Indy 10.5.7 | This version requires Indy 10.5.7 for the Indy communication adapter. Tested with revision 3865. |
| FPC 2.2.4 | Tested with Free Pascal 2.2.4 and Lazarus 0.9.28 |
| SuperObject 1.2 | SuperObject has been updated to release 1.2 |
| Synapse release 39 | Synapse has been updated to release 39 |
| Read Timeout | Improved handling of EidReadTimeout exceptions in the Indy communication adapter. |
| Connect | If a Connect fails, the message of the underlying exception will be included in the message of the EConnectionFailedException. |

Version 1.9

Released September 8, 2009

New

- FUSE Broker** This version passed all unit tests with the current release 5.3.0.3 of FUSE Message Broker.
- Transformation** If the `USE_TCLASS_TRANSFORMER` compiler condition is not set, the new `TTransformable` class declared in unit `BTTransformable` will be used.
- Connect TimeOut** The connection factory class now has a `ConnectTimeOut` property. Please note this is still an experimental feature.
- Tutorials** Online tutorials are available for using Habari ActiveMQ with FUSE Message Broker and with Apache Geronimo.
- Delphi 2010** Tested with Delphi 2010 (all unit tests passed)

Changed

- Indy 10.5.6** This version requires Indy 10.5.6 for the Indy communication adapter.
- Text message size** Support for receiving of text messages with more than 16 KB size over the Indy communication adapter can be enabled in method `ReadMessageBuffer` in unit `BTCommAdapterIndy.pas`.
- Connection.Close** The destructor of `TBTJMSConnection` calls `Close` to avoid memory leaks.
- ERROR frame** The body of ERROR frames is included in the exception message.
- inline keyword** The `inline` keyword will be used in some performance critical places if the `USEINLINE` compiler condition is set.
- HabariExpress** A hint has been added to notify about the unsupported / demo status of `HabariExpress` and `HabariExpressAdmin`.

`HabariActiveMQExpress` and `HabariActiveMQExpressAdmin` subclasses have been added to the demo component classes `HabariExpress` and `HabariExpressAdmin`.
- Memory leaks** Some potential memory leaks have been fixed in the source and the demo applications.

Version 1.8

Released July 7, 2009

Changed

Indy 10 update

The deprecated en8bit encoding has been replaced with Indy8BitEncoding. If you use an old version of Indy 10, please update to a newer revision.

Synapse update

This version includes Synapse revision 98.

JMS API

The API now follows the JMS specification more closely. Some interface declarations (IStartable, IStoppable, IDispatching, IDisposable) have moved to BTAMQInterfaces unit. CreateTemporaryQueue and CreateTemporaryTopic methods have been added but are not implemented in the ActiveMQ message broker.

Properties

Old property access methods have been removed. Use Set/GetStringProperty, Set/GetIntProperty and Set/GetBooleanProperty instead. Use Message.PropertyNames to get an array of the property names for the message. PrimitiveMap is now declared in the new unit BTPrimitiveMap.

Acknowledge

The non-functional method Session.Acknowledge has been removed. Use Message.Acknowledge instead.

CreateConsumer

The non-standard method CreateConsumer with MessageListener parameter has been removed. The MessageListener of a MessageConsumer can be assigned directly using the MessageConsumer.MessageListener property.

Demo apps

Fixed username / password support for ProducerTool and ConsumerTool.

Version 1.7

Released May 5, 2009

New

- Message Interface** The Message interface now includes methods to get and set message properties: `GetBooleanProperty`, `SetBooleanProperty`, `GetIntProperty`, `SetIntProperty`, `GetStringProperty`, `SetStringProperty`. A new method in the Message interface, `GetPropertyNames`, returns an array of property names. The properties are still accessible through the Properties property, but usage of the new direct access methods is recommended
- BTStreamHelper** A new unit provides helper methods which read stream content into binary messages

Fixed

- Binary Messages** A bug in the `ReadMessageBufferLen` method in the Indy communication adapter caused errors with binary messages

Version 1.6

Released April 2, 2009

New

- Exceptions** Introduced a new `EConnectionFailedException` which will be thrown when the connect failed
- Transformation** Server side errors for object transformation which are included as a Stomp header ('`transformation-error`') will raise an `EJMSEException` in the client
- ISession Interface** The `ISession` interface has a new '`AcknowledgeMode`' property
- SuperObject 1.1** Includes version 1.1 of `SuperObject` - note that version 1.1 uses a different parse method name, `ParseString`, instead of `Parse` in versions up to 1.0
- ActiveMQ 5.3** Tested with Apache ActiveMQ 5.3-SNAPSHOT

Changed

- Lost connections** Improved handling for 'connection reset by peer' and 'connection closed gracefully'
- Durable subscriber** Workaround which allows to unsubscribe a durable subscription (method `ISession.Unsubscribe` with two parameters)
- Acknowledge mode** Documentation update, use `amClientAcknowledge` (ActiveMQ support for `amAutoAcknowledge` is malfunctioning)
- Transaction ID** The `TBTTransactionContext` class uses a thread safe sequence generator instead of a GUID to create a new transaction id
- IDisposable** Minor clean up of interfaces in unit `JMSInterfaces` (introduced `IDisposable` to remove `Dispose` from `IMessageProducer` and use it in consumer and producer classes)
- Documentation** Extracted Enterprise Integration examples to new document (`HabariEnterpriseExamples.pdf`)
- Transactions** Transaction commands (Commit, Rollback) may throw `EJMSEException`
- Durable Subscriber** `CreateDurableSubscriber` throws a `EJMSEException` if you do not specify the same `clientId` on the connection and `subscriptionName` on the subscribe

Fixed

- FillStompFrame** Sets the `JMSReplyTo` destination as 'reply-to' Stomp header (for outgoing messages)
- FillCreatedMessage** Supports the 'destination' Stomp header in incoming messages
- Demo projects** Fixed problems in the `ConsumerTool` and `ProducerTool` demo projects
- BTStompConnection** Fixed a memory leak in the `TBTStompConnection` class
- Cleanup** Removed unnecessary uses

Version 1.5

Released March 3, 2009

New

- XML transformation** Support for object exchange using XML serialization, based on persistency helper methods in the OpenXML library (see example in xmljava folder)
- ProducerTool demo** Command line tool which generates test messages, many configuration parameters (inspired by the ActiveMQ ProducerTool class)
- ConsumerTool demo** Command line tool which consumes test messages, many configuration parameters (inspired by the ActiveMQ ConsumerTool class)

Fixed

- BinaryMessage** Fixed a bug in the Indy communication adapter (Delphi 2009)
- Message header** The FillCreatedMessage method now only copies user-defined Stomp headers to the properties of the incoming JMS message

Changed

- Synapse exceptions** The Synapse adapter raises exceptions in case of connection failures (this is now consistent with the Indy implementation)
- TBytes data type** All communication adapters use the TBytes data type in the StompTransmit method
- Transformer** Registration of default transformers has been replaced by explicit creation, the transformer constructor parameter is the class of the serialized objects
- SOAP transformer** The BTMessageTransformerSOAP unit is beta / experimental now
- DelphiGUI demo** The demo application includes a new administration page which displays server, client, topic and queue information (see readme.txt for details about message broker configuration)
- JSON Toolkit** Deprecated JSON Toolkit adapter has been deleted
- beta folder** The folder <installdir>\source\beta contains beta versions of new units (note that these units are not guaranteed to be included in future versions)

Version 1.4

Released February 9, 2009

New

| | |
|---------------------------|---|
| BrokerURL | The factory methods to create a JMS connection now use URI syntax. For example, the BrokerURL using the stomp or the stomp+ssl protocol would be 'stomp://localhost' or 'stomp+ssl://localhost:61612' |
| Durable Subscriber | Session.CreateDurableSubscriber method creates a durable subscriber to the specified topic |
| SSL Support | A new communication adapter with SSL support is included, TBTCOMMAdapterIndySSL |
| Send Timeout | The send timeout can be set using a new property of the JMS connection |
| Synapse Support | Delphi 2009 can be used with revision 95 of the Synapse library |
| IkJSON | Delphi 2009 can be used with version 1.05 of the IkJSON library (and USE_D2009 conditional symbol) |

Changed

| | |
|----------------------|---|
| Demo | SSL support has been added to the delphigui demo application |
| Renamed file | The BTConnectionFactory.pas unit has been renamed to BTJMSSessionFactory.pas to avoid name conflicts with other Habari Client libraries |
| Performance | The performance of the Synapse based communication adapter has been improved |
| JMS Selectors | The manual now includes information about the usage of JMS Selectors in SQL and XPath syntax |
| JMSReplyTo | JMSReplyTo headers are now supported in for incoming messages |
| XPath support | The documentation has been updated to include the information about required XPath support libraries (JAR files) |
| Delphi 2009 | Fixed all compiler warnings (except for third party libraries like SuperObject, IkJSON, Synapse) |

Version 1.3

Released January 8, 2009

New

| | |
|---------------------|---|
| Transformer | Like communication adapters, all object message transformers now use a transformer registry. A message transformation unit is provided for every JSON and SOAP implementation library |
| JSON Support | JSON serialization is now supported in Delphi 2009 with the new SuperObject library |
| SOAP Support | SOAP message transformation adapter with demo application |

Changed

| | |
|-----------------------------|--|
| Interface Parameters | The const keyword has been added to interface type parameters to avoid unnecessary reference counting |
| ActiveMQ 5.2 | This release has been tested with the new release 5.2 of Apache ActiveMQ |
| ICS V6 RC 1 | This release has been tested with ICS V6 RC1, the release candidate of the Internet Component Suite |
| ICS/TServerSocket | The source code for unsupported TCP/IP communication libraries is now located in the folder source/unsupported/commlib |
| Multi Threading | The DUnit test suite includes new tests for multi threaded usage of the core library |

Version 1.2

Released September 6, 2008

New

| | |
|--------------------|--|
| Delphi 2009 | The library compiles and runs in Delphi 2009. Unicode is supported in the message body and message property values. Note: JSON object transformation is not supported for Delphi 2009. |
|--------------------|--|

IkJSON Support Support for the IkJSON library has been added. The default library for JSON transformation is json_toolkit. To activate IkJSON, add the conditional symbol LKJSON.

Load Balancing The demo source code now include a simple file based load balancing example.

Fixed

Expiration Time The library used the local time zone to calculate the expiration time in the message expiration header. This has been changed to UTC.

Closed Connections Closing a closed connection does not throw an EBTSmpClientAlreadyDisconnectedError anymore.

Packages The pre-built package files include the source path to the JSON library now.

Version 1.1

Released March 31, 2008

New

ObjectMessage A new message type supports data exchange using the Apache ActiveMQ standard JSON object message transformation

The property OptionsMessageTransformer has been added in HabariExpress to support JSON object message transformation

Subscription config You can add custom headers to configure a subscription. (see Chapter 'Destinations')

The property OptionsConsumer has been extended in HabariExpress to support subscription configuration

Version 1.0.1

Released March 11, 2008

New

| | |
|--------------------|---|
| Unicode properties | String type message properties now support Unicode |
| Palette bitmap | HabariExpress and HabariExpressAdmin now have palette bitmaps (component icons) |

Fixed

| | |
|--------------|--|
| Unicode body | Incoming text messages which used Unicode in the message body have not been converted back to WideString. This has been fixed. |
| Examples | The SoapTransfer example application has been fixed |

Version 1.0

Released March 5, 2008

Index

Reference

| | | | |
|------------------------------|---------------------------|----------------------------|-------------------|
| ActiveMQ..... | 9, 86 | JMSDeliveryMode..... | 73 |
| Authentication..... | 13 | JMSExpiration..... | 73 |
| Binary Message..... | 33 | JMSMessageId..... | 74 |
| Broker Statistics..... | 62 | JMSPriority..... | 73 |
| BTJMSConnection..... | 37 | JMSReplyTo..... | 62, 73 |
| BTSerialIntf..... | 38 | JMSTimestamp..... | 74 |
| BTStreamHelper..... | 33 | JMSXGroupID..... | 74 |
| Communication Adapter..... | 19 | JMSXGroupSeq..... | 74 |
| Connection..... | 19 | JSON..... | 86 |
| ConnectionFactory..... | 19 | LkJSON..... | 36, 86, 92 |
| ConsumerTool..... | 49 | LoadBytesFromStream..... | 33 |
| CreateDurableSubscriber..... | 42 | Log4D..... | 82, 87, 93 |
| CreateObjectMessage..... | 38 | Map Message..... | 40 |
| Cron..... | 71 | Message Consumer..... | 27 |
| Delayed Delivery..... | 71 | Message Producer..... | 27 |
| Destination..... | 25 | MessageTransformer..... | 37 |
| DUnit..... | 80 | NativeXml..... | 36, 87, 94 |
| Failover Support..... | 22 | Object Message..... | 34 |
| HABARI_LOGGING..... | 82 | OmniXML..... | 36, 87 |
| HABARI_RAW_TRACE..... | 82 | OnMessage..... | 31 |
| HABARI_STOMP_11..... | 82 | Point-to-point..... | 24 |
| HABARI_USE_RTTI..... | 82 | ProducerTool..... | 56 |
| IConnection..... | 19 | Publish and subscribe..... | 24 |
| IDestination..... | 31 | Queue..... | 25 |
| IMapMessage..... | 62 | Receive..... | 32 |
| IMessage..... | 31 | ReceiveNoWait..... | 32 |
| IMessageConsumer..... | 31 | SamplePojo..... | 37 |
| IMessageListener..... | 31 | Scheduler..... | 71 |
| IMessageProducer..... | 38 | Session..... | 20 |
| Internet Direct (Indy)..... | 9, 10 , 83, 87, 92 | SetTransformer..... | 37 |
| ISession..... | 38 | Stomp..... | 86 |
| JMS..... | 7 , 86 | SuperObject..... | 36, 86, 93 |
| JMS Selector..... | 28 | Synapse..... | 9, 10 , 87 |
| JMSCorrelationID..... | 73 | Text Message..... | 30 |

| | | | |
|---------------------------|----|----------------------------|----|
| Throughput Test Tool..... | 61 | Transformation-custom..... | 38 |
| Topic..... | 25 | XML..... | 87 |
| TopicSubscriber..... | 42 | XPath..... | 29 |
| Transacted Sessions..... | 21 | XStream..... | 87 |
| Transformation..... | 37 | | |

Table Index

| | |
|--|----|
| Communication Adapters..... | 17 |
| Failover Transport Options..... | 22 |
| Message Transformer Implementations..... | 36 |
| Basic Demo Applications..... | 47 |
| Advanced Demo Applications..... | 48 |
| ConsumerTool Command Line Options..... | 49 |
| ProducerTool Command Line Options..... | 56 |
| Throughput Test Tool Command Line Options..... | 61 |

Illustration Index

| | |
|---|----|
| Illustration 1: Shared Business Logic..... | 6 |
| Illustration 2: Peer to Peer Communication..... | 6 |
| Illustration 3: Load Balancing..... | 7 |
| Illustration 4: ActiveMQ 5.3.0 running with Stomp and Stomp+SSL connectors..... | 12 |
| Illustration 5: Programming Model..... | 18 |
| Illustration 6: Performance Test Application..... | 66 |
| Illustration 7: PHP Producer Demo – Web Interface..... | 67 |
| Illustration 8: PHP Producer Demo - Incoming Messages..... | 67 |
| Illustration 9: DUnit GUI Testrunner..... | 81 |