



habarisoft
Enterprise Messaging Software for Delphi®

Getting started with Habari Client for RabbitMQ

Version 1.3

Trademarks

Habari is a registered trademark of Michael Justin and is protected by the laws of Germany and other countries. RabbitMQ™ is a Trademark of Rabbit Technologies Ltd. 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	4
About Habari Client for RabbitMQ	4
Quick Start	7
Download and Installation	7
Dependencies	10
Requirements	10
TCP/IP Communication Libraries	10
Communication Adapter Configuration	11
Introduction	11
The JMS API Programming Model	12
Tutorials	13
Habari Quick Start Tutorial	13
Connections and Sessions	16
Step by Step Example	16
Transacted Sessions	18
Failover Support	19
Destinations	20
Introduction	20
Create a new Destination	21
Producer and Consumer	22
Message Producer	22
Message Consumer	22
Text Messages	24
Sending Text Messages	24
Receive Text Messages	25
Bytes Messages	27
Creation	27
Sending	27
Object Messages	28
Introduction	28
Message Transformers in Habari Client for RabbitMQ	28
Map Messages	31
Introduction	31
Durable Subscriptions	33
Description	33
Example Applications	34
Example Application Index	34
ConsumerTool	35
ProducerTool	36
Performance Test	37
Message Options	38
User Defined Properties	38
Useful Units	40
BTStreamHelper	40
BTJavaPlatform	40

Known Limitations	41
Limitations of the RabbitMQ STOMP adapter	41
Known Problems	41
Messages	41
Header Properties	42
References	43
Habari Client for RabbitMQ License	44
Third Party Library Licenses	46
Synapse	46
Indy BSD License	46
IkJSON	47
SuperObject	48
Log4D	48
NativeXml	48
Release Notes	50
Version 1.3	50
Version 1.2	50
Version 1.1	51
Version 1.0	52
Index	53

Introduction

About Habari Client for RabbitMQ

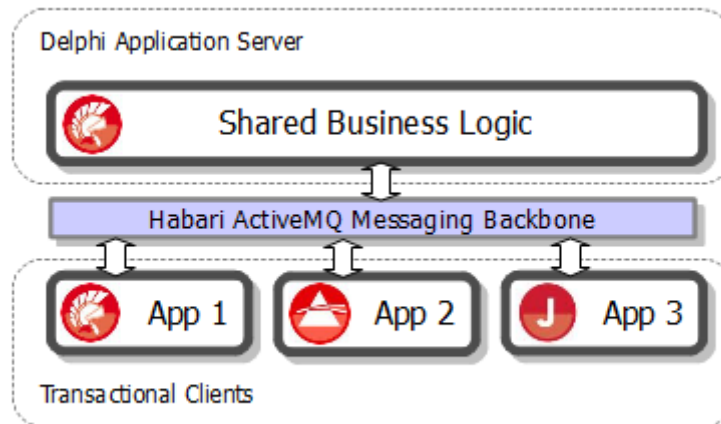
Habari Client for RabbitMQ is a Delphi library for RabbitMQ. With Habari Client for RabbitMQ, Delphi developers can build integrated solutions, connecting cross language clients and protocols from C, Delphi, and Java using the peer-to-peer or the publish and subscribe communication model. The library uses the Stomp message protocol and a plug-in architecture for communication libraries (including SSL). It supports RabbitMQ version 2.6.1 and higher, Delphi 2009 to XE2 and Free Pascal, and follows the specification of the JMS API for Message Oriented Middleware.

How Can I Use It?

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

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

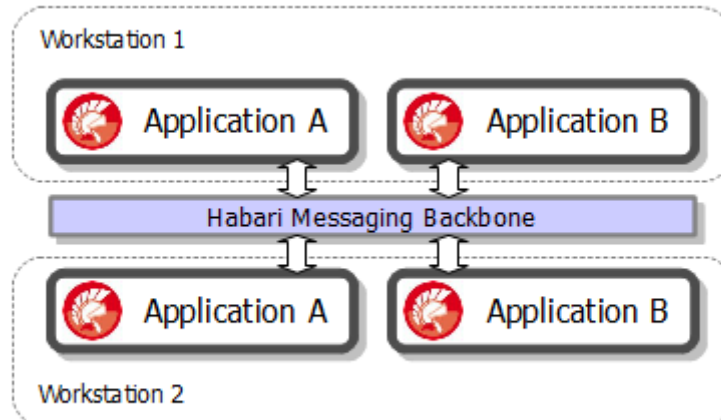
Example Illustrations



Habari for shared business logic

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

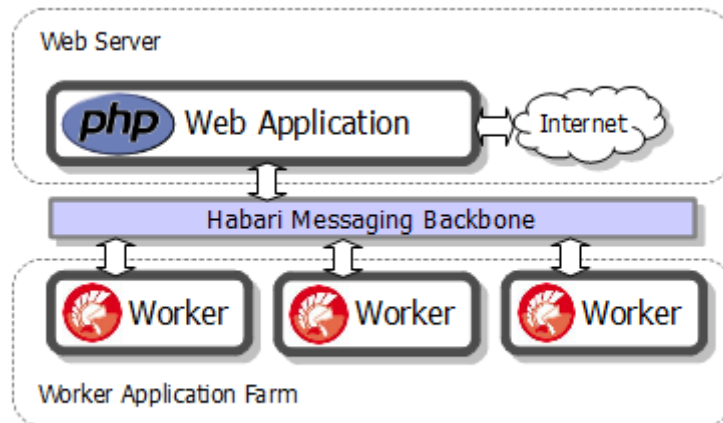
Documents and messages (including objects, serialized using JSON or XML) can be exchanged and secured by **client-side acknowledgment** and **transactional sessions**.



Habari in a network of Delphi applications

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

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



Habari in a load balancing solution

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

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

Quick Start

Download and Installation

Main Page

The main RabbitMQ download page is located at

<http://www.rabbitmq.com/download.html>

Server

The server is available in several compiled forms, including an installer for Windows systems, or as source.

The Installation guide for Erlang and the Windows installer can be found at

<http://www.rabbitmq.com/install-windows.html>

Stomp Plugin (RabbitMQ 2.7.0 and newer)

To enable the Stomp plugin and its dependencies, follow the instructions on

<http://www.rabbitmq.com/plugins.html>

The common syntax for plugin activation is

```
rabbitmq-plugins enable plugin-name
```

Note:

*Releases prior to 2.7.0 did not include plugins with the server.
For details, please visit the plugin page.*

Start

To launch the message broker on Windows simply run the rabbitmq-server.bat script file

```
<path-to-rabbitmq>\sbin\rabbitmq-server.bat
```

Broker log messages

```

Activating RabbitMQ plugins ...
2 plugins activated:
* amqp_client-2.7.1
* rabbitmq_stomp-2.7.1

+----+ +----+
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  +----+ +-----+
|  RabbitMQ +----+ |
|  v2.7.1 +----+ |
|  |  |  |  |
+-----+
AMQP 0-9-1 / 0-9 / 0-8
Copyright (C) 2007-2011 VMware, Inc.
Licensed under the MPL. See http://www.rabbitmq.com/

node           : rabbit@...
app descriptor : ../sbin/../ebin/rabbit.app
home dir       : C:\...
config file(s) : (none)
cookie hash    : WjOr7YEHESuUkR1mMuR+g==
log            : C:/.../RabbitMQ/log/rabbit@MJ-PC.log
sasl log       : C:/.../Roaming/RabbitMQ/log/rabbit@MJ-PC-sasl.log
database dir   : c:/.../Roaming/RabbitMQ/db/rabbit@MJ-PC-mnesia
erlang version : 5.8.3

-- rabbit boot start
starting file handle cache server      ...done
starting worker pool                   ...done
starting database                       ...done
starting codec correctness check        ...done
-- external infrastructure ready
starting plugin registry                ...done
starting auth mechanism cr-demo         ...done
starting auth mechanism amqplain        ...done
starting auth mechanism plain           ...done
starting statistics event manager       ...done
starting logging server                 ...done
starting exchange type direct           ...done
starting exchange type fanout           ...done
starting exchange type headers          ...done
starting exchange type topic            ...done
-- kernel ready
starting alarm handler                  ...done
starting node monitor                   ...done
starting cluster delegate               ...done
starting guid generator                 ...done
starting memory monitor                 ...done
-- core initialized
starting empty DB check                 ...done
starting exchange, queue and binding recovery ...done
starting mirror queue slave sup         ...done
starting adding mirrors to queues       ...done

```

```
-- message delivery logic ready
starting error log relay           ...done
starting networking               ...done
starting direct_client            ...done
starting notify_cluster nodes    ...done

broker running
starting STOMP Adapter (binding to {[61613],[ ]}) ...done
```

Dependencies

Requirements

Development Environment

- Delphi 2009 or higher
- Free Pascal 2.4.4 or higher

Message Broker

- RabbitMQ 2.7.1 or higher with Stomp adapter plugin installed

TCP/IP Communication Library

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

Internet Direct (Indy)

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

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

Synapse

<http://www.ararat.cz/synapse/doku.php/download>

TCP/IP Communication Libraries

Supported libraries

Internet Direct (Indy) 10

The communication adapter for Indy supports both GUI-based and console mode applications.

Synapse

The communication adapter for Synapse supports both GUI-based and console mode applications.

Communication Adapter Configuration

Introduction

Habari uses communication adapters as an abstraction layer between the internal library and the TCP/IP library.

These adapters are implemented using a common API, which allows to exchange them easily, even at run time.

Installation of Communication Adapter classes

A communication adapter implementation can be prepared for usage by simply adding its Delphi unit to the project.

Behind the scenes, the communication adapter will add itself to the communication adapter list in the BTAdapterRegistry unit.

If more than one communication adapter is in the project, the first adapter class in the list will be the default adapter. (The methods of the adapter registry performs some checks, for example to prevent duplicate entries in the adapter list, and raise exceptions in case of errors)

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

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

Available Communication Adapters

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

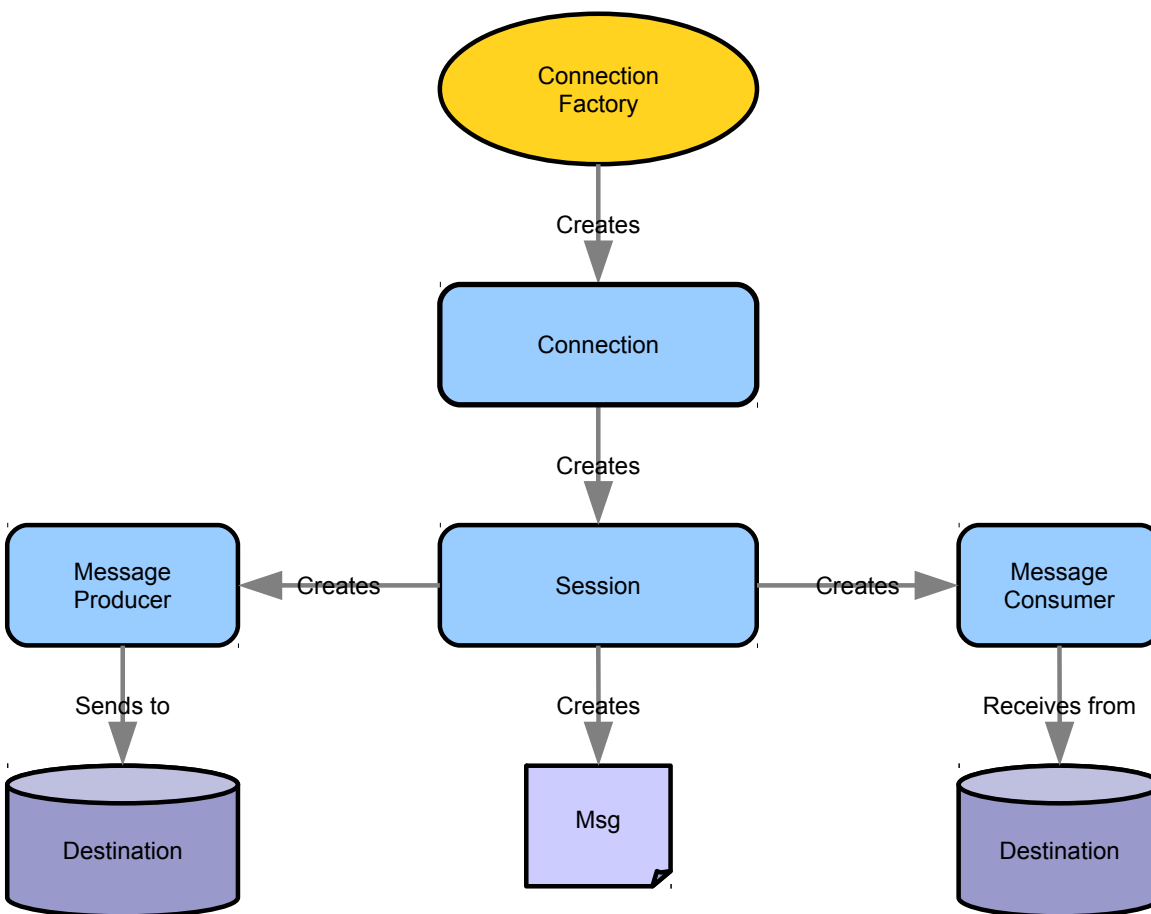
Adapter classes and units

Adapter	Unit name
TBCommAdapterIndy	BTCommAdapterIndy
TBCommAdapterSynapse	BTCommAdapterSynapse

The JMS API Programming Model

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

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



The JMS API Programming Model: Overview

Tutorials

Habari Quick Start Tutorial

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

This tutorial takes less than 10 minutes to complete.

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

<u>Software or Resource</u>	<u>Version required</u>
Delphi	2009 ¹
Synapse	rev. 144
RabbitMQ	Version 2.6.1

Setting up the project

To create a new project:

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

Now the project is created and saved.

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

Adding code to the project

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

8. Switch to Code view (F12)

1 Delphi 6 to XE are supported, only the IDE steps are different

9. Add the required units to the interface uses list:

```
uses
  BTJMSConnection,
  BTJMSInterfaces,
  BTCommAdapterSynapse,
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs;
```

10. Compile and save the project.

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

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

13. Add the following code to send the message:

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

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

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

```
procedure TForm1.Button2Click(Sender: TObject);
var
  Connection: IConnection;
  Session: ISession;
  Destination: IDestination;
  Consumer: IMessageConsumer;
  Msg: ITextMessage;
begin
  Connection := TBTJMSConnection.MakeConnection;
  Connection.Start;
  Session := Connection.CreateSession(False, amAutoAcknowledge);
  Destination := Session.CreateQueue('TEST.DEFAULT');
  Consumer := Session.CreateConsumer(Destination);
  Msg := Consumer.Receive(1000) as ITextMessage;
```

```
if Assigned(Msg) then
  ShowMessage(Msg.Text);
Connection.Close;
end;
```

16. Compile and save the project

Run the demo

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

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

Next steps

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

Connections and Sessions

Step by Step Example

Add required units

Three units are required for this example

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

The SysUtils unit is necessary for the exception handling.

```
program SendOneMessage;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  BTCommAdapterIndy,
  BTJMSConnection,
  BTJMSInterfaces;
...
```

Creating a new Connection

To create a new connection,

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

or

- use an instance of TBTJMSConnectionFactory to create connections

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

```
var
```

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

Local connection

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

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

Creating a Session

To create the communication session,

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

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

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

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

Using the Session

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

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

Closing a Connection

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

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

```
end.
```

Transacted Sessions

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

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

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

Failover Support

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

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

Example for a failover URI:

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

Transport Options

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

Example URI:

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

Example code:

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

Destinations

Introduction

The JMS API supports two models:²

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.

Note

The RabbitMQ message broker does not support durable subscriptions

² 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).

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.

Note

The RabbitMQ message broker does not support message priorities

About Message Priorities

Though the AMQP protocol supports the concept of priority, RabbitMQ does not yet implement that feature.³

The AMQP protocol supports up to 10 levels of priority, starting at zero. 0 has the lowest priority and 9 has the highest. The priority of a message is set by the publisher using the priority header. The consumer will then have messages pushed to it in priority order.

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:

³ <http://dougbarth.github.com/2011/07/01/approximating-priority-with-rabbitmq.html>

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

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

Note

Message selectors are exposed in the library API but not supported by the RabbitMQ broker.

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

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

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

Text Messages

Sending Text Messages

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

```
program SendOneMessage;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  BTCommAdapterIndy in '..\..\source\BTCommAdapterIndy.pas',
  BTJMSConnection in '..\..\source\BTJMSConnection.pas',
  BTJMSInterfaces in '..\..\source\BTJMSInterfaces.pas';

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

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

The unit `BTCommAdapterIndy` contains the Internet Direct (Indy) communication adapter class. By including this unit, it will register the adapter class with an internal list of all available communication adapters. By default, the first registered communication adapter will be used.

Receive Text Messages

Asynchronous receive

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

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

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

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

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

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

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

This interface only contains one procedure, OnMessage:

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

Synchronous Receive

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

Example:

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

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

Receive and ReceiveNowait

There are three different methods for synchronous receive:

- | | |
|-------------------------|--|
| 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. |

Bytes Messages

Creation

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

Sending

Reading Binary Content using BTStreamHelper

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

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

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

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

  Size := Length(Msg.Content);

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

finally
  FS.Free;
end;
```

Object Messages

Introduction

Object Serialization

Object serialization is the process of saving an object's state to a sequence of bytes, as well as the process of rebuilding those bytes into a live object at some future time.⁴ In messaging applications, object serialization is required to transfer objects between clients, but also to store objects on the broker if they are declared persistent.

Message Transformers in Habari Client for RabbitMQ

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

Table 1: Message Transformer Implementations

Memory Management

Outgoing Objects

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

Incoming Objects

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

⁴ <http://java.sun.com/developer/technicalArticles/Programming/serialization/>

Assign a Message Transformer

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

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

```
Connection: IConnection;
...

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

...
Connection.Start;
```

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

```
Connection: IConnection;
...

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

...
Connection.Start;
```

Create and Send an ObjectMessage

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

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

Complete Example using NativeXml

From ObjectExchangeTests.pas.

Send:

```
procedure TObExTestCase.TestXMLNative;
var
  ObjectMessage: IObjectMessage;
  Obj: SamplePojo;
begin
  // send
  Connection := TBTJMSSConnection.MakeConnection;
  try
```

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

Receive:

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

Map Messages

Introduction

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

Map message transformers take a nil parameter as argument.

Complete Example

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

Send:

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

Receive:

```
Connection := TBTJMSConnection.MakeConnection;
try
  SetTransformer(Connection, TBTMessageTransformerXMLMapNative.Create(nil));
  Connection.Start;
  Session := Connection.CreateSession(False, 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.⁵ The combination of the `clientId` and durable subscriber name uniquely identifies the durable topic subscription. After you restart your program and re-subscribe, the Broker will know which messages you need that were published while you were away.

Creation

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

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

Example

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

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

and receive them from a client:

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

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

Example Applications

Example Application Index

Basic Features

Directory	Description
common-chat	Simple chat client. (HabariChat.dpr, requires Delphi 2009+)
common-consumertool	Receives messages from broker.
common-delphigui	Sends and receives messages. (GUIDemo.dpr, requires Delphi 2009+)
common-performance	Multi-threaded performance test application.
common-ducertool	Sends messages to a broker.
common-throughput	Continuously produces and consumes messages to monitor the average message throughput over time.

Table 2: Basic Demo Applications

ConsumerTool

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

Parameter	Default Value	Description
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 3: ConsumerTool Command Line Options

Examples

Receive 1000 messages from local broker

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

Receive 10 messages from local broker and wait for any key

```
ConsumerTool --PauseBeforeShutDown
```

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

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

ProducerTool

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

Parameter	Default	Description
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 4: ProducerTool Command Line Options

Examples

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

```
ProducerTool --MessageCount 10000
```

Send 10 messages to the topic `ExampleTopic` on the local broker

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

Performance Test

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

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

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

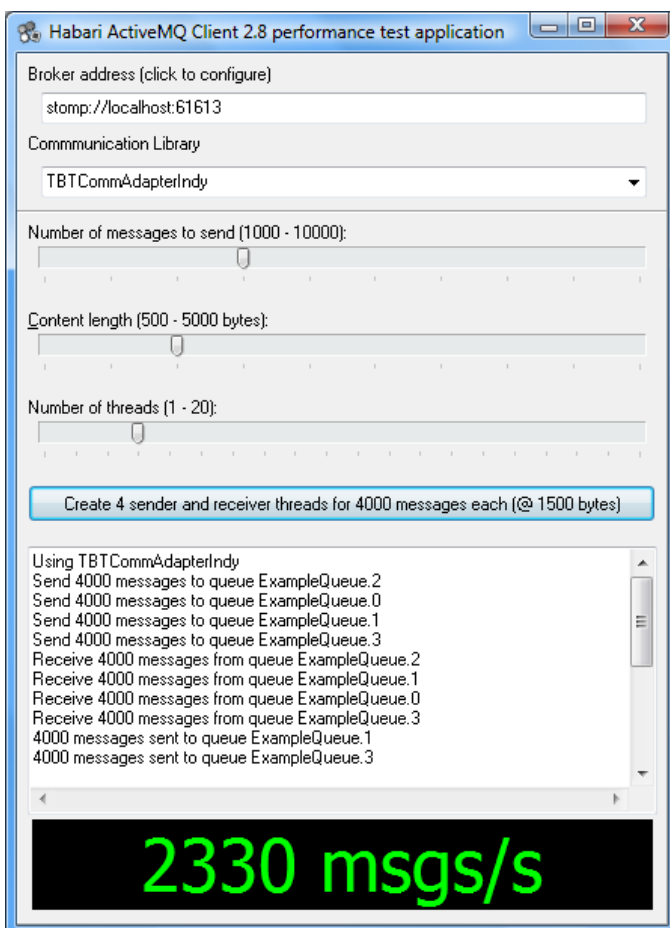


Illustration 1: Performance Test Application

Message Options

User Defined Properties

Supported Data Types

The Stomp protocol only supports string type properties.

Reserved Property Names

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

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

- redelivered

Useful Units

BTStreamHelper

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

Example:

```
Msg := Session.CreateBytesMessage;

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

Producer.Send(Msg);
```

BTJavaPlatform

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

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

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

Known Limitations

Limitations of the RabbitMQ STOMP adapter

The following JMS extensions of the Stomp protocol are exposed in the library API but not supported by the RabbitMQ message broker:

- JMS Selectors
- Message Expiration
- Message Priority
- Durable Consumers

For an overview of supported features please see also the **Feature Matrix**.

Known Problems

Message loss with Topic Destinations

Message loss has been discussed in the thread **[2.4.0] random message loss with Stomp and topics?**⁶

It seems that because messages and subscribe commands are processed asynchronously, messages are discarded until the destination is ready.

A simple workaround has been implemented in Habari Client for RabbitMQ as suggested in the newsgroup thread: when a destination is subscribed, the client requests a receipt confirmation message and then blocks until the broker has confirmed the subscription by sending a Stomp RECEIPT frame.

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 (e.g. SetInt or SetDate), all message properties will be interpreted as Java Strings by the Message Broker. As a side effect, the expressions in a Selector are limited to operations which are valid for strings. Timestamp properties are converted to an Unix time stamp value, which

⁶ <http://lists.rabbitmq.com/pipermail/rabbitmq-discuss/2011-April/012384.html>

is the internal representation in Java. But still, these values can not be used with date type expressions.

Header Properties

The client library does not process these RabbitMQ specific⁷ STOMP headers:

amqp-message-id	the message-id property
content-encoding	the content-encoding property

⁷ <http://www.rabbitmq.com/stomp.html>

References

Message Broker

Home page <http://www.rabbitmq.com/>

IDE

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

Stomp

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

Communication Libraries

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

Internet Direct (Indy) <http://www.indyproject.org>

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

Habari Client for RabbitMQ License

Habari Client for RabbitMQ (c) 2011-2012 Habarisoft - Michael Justin

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

GENERAL TERMS THAT APPLY TO COMPILED PROGRAMS AND REDISTRIBUTABLES

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

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

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

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

LIMITED WARRANTY

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

Third Party Library Licenses

Synapse

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

```

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

```

Indy BSD License

Copyright

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

License

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

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

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

lkJSON

```
LkJSON v1.07
```

```
06 november 2009
```

```
* Copyright (c) 2006,2007,2008,2009 Leonid Koninin
* leon_kon@users.sourceforge.net
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*   * Redistributions of source code must retain the above copyright
*     notice, this list of conditions and the following disclaimer.
*   * Redistributions in binary form must reproduce the above copyright
*     notice, this list of conditions and the following disclaimer in the
*     documentation and/or other materials provided with the distribution.
*   * Neither the name of the <organization> nor the
*     names of its contributors may be used to endorse or promote products
*     derived from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY Leonid Koninin ``AS IS'' AND ANY
* EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
```

```
* DISCLAIMED. IN NO EVENT SHALL Leonid Koninin BE LIABLE FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

SuperObject

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

Log4D

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

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

NativeXml

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

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

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

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

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

Release Notes

Version 1.3

Released May 8, 2012

New

- amClientIndividual** RabbitMQ 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)`
- 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`
- RabbitMQ 2.8.2** Tested with RabbitMQ version 2.8.2

Changed

- Throughput test** The utility displays more statistics about message count and transfer speed.
- Persistency** A workaround fixes persistent messages, by requesting for broker receipt messages and waiting (blocking) until the receipt arrives or a time limit expired (set to 5000 milliseconds)
- ReceiveNoWait** ReceiveNoWait has been improved in the Synapse communication adapter

Version 1.2

Released January 31, 2012

New

- RabbitMQ 2.7.1** Tested with RabbitMQ version 2.6.1 and 2.7.1
More than 12500 outgoing messages per second (less than 80 microseconds per message)

Changed

Indy rev. 4733	Tested with revision 4733 of Indy 10.5.8
FPC 2.6.0	Tested with Free Pascal 2.6.0
Throughput Demo	The throughput tool displays more information about message transfer rates and can be stopped with Ctrl+C and the close box
Chat Demo	The chat demo starts with a connection configuration dialog and has an improved user interface
GUI Demo	The GUI demo allows to save the file which is in an incoming message
Packages	Removed packages for unsupported demo component HabariExpress
Fixes	Improvements in the Indy and Synapse communication adapter, and warnings in object message transformers

Version 1.1

Released October 18, 2011

New

Durable Subscriber	If a client needs to receive all the messages published on a topic, including the ones published while the subscriber is inactive, it now can use a durable TopicSubscriber and the method TBTJMSSession.CreateDurableSubscriber.
Throughput Demo	The throughput tool continuously produces and consumes messages to monitor the average message throughput.
RabbitMQ 2.6.1	Tested with RabbitMQ version 2.6.1
Framedecoder v2	Switched to a new Stomp frame decoder

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 4.01	Upgraded for XE2 support (included)
Indy rev. 4686	Tested with revision 4686 of Indy 10.5.8
Unicode Tests	DUnit tests now include checks of Unicode string properties in object message exchanges, this revealed a problem with Delphi

6 - it is recommended to use Delphi 2009 or newer for object message exchange

Version 1.0

Released August 9, 2011

Index

Reference

BTJMSConnection.....	29	message selector.....	23
BTStreamHelper.....	27	Message selector.....	23
Connection.....	16	MessageListener.....	23, 25
connection factory.....	16	MessageTransformer.....	29
ConsumerTool.....	35	NativeXml.....	28, 48
CreateDurableSubscriber.....	33	Object Message.....	28
CreateObjectMessage.....	29	OmniXML.....	28
Destination.....	21	OnMessage.....	23, 25
Failover Support.....	19	point-to-point.....	20
IConnection.....	16	ProducerTool.....	36
IDestination.....	25	publish and subscribe.....	20
IMessage.....	25	Queue.....	21
IMessageConsumer.....	25	Receive.....	26
IMessageListener.....	25	ReceiveNoWait.....	26
IMessageProducer.....	29	SamplePojo.....	29
Internet Direct (Indy).....	10, 43	Session.....	17
ISession.....	29	SetTransformer.....	29
JMS.....	43	Stomp.....	43
IkJSON.....	47	SuperObject.....	28, 48
LoadBytesFromStream.....	27	Synapse.....	10, 43
Log4D.....	48	Text Message.....	24
Map Message.....	31	Topic.....	21
Message Consumer.....	22	TopicSubscriber.....	33
Message Producer.....	22	Transacted Sessions.....	18

Table Index

Table 1: Message Transformer Implementations.....	28
Table 2: Basic Demo Applications.....	34
Table 3: ConsumerTool Command Line Options.....	35
Table 4: ProducerTool Command Line Options.....	36

Illustration Index

Illustration 1: Performance Test Application.....	37
---	----