SGCWEBSOCKETS · TECHNICAL DOCUMENT

AMQP 0.9.1

AMQP 0.9.1 client for Delphi, C++ Builder and .NET — speak natively to RabbitMQ and other 0.9.1-compatible brokers over TCP or WebSocket.

eSeGeCe

WWW.ESEGECE.COM

Overview

The Advanced Message Queuing Protocol (AMQP) is an open standard application layer protocol for message-oriented middleware. The defining features of AMQP are message orientation, queuing, routing (including point-to-point and publish-and-subscribe), reliability and security.

At a glance

COMPONENT CLASS

`TsgcWSPClient_AMQP`

STANDARDS / SPEC

AMQP 0.9.1 specification (RabbitMQ PDF)

TRANSPORTS

TCP, TLS

PLATFORMS

Windows, macOS, Linux, iOS, Android

FRAMEWORKS

VCL, FireMonkey, Lazarus / FPC, .NET

EDITION

Standard / Professional / Enterprise

Features

- Monitoring and sharing updates.
- Ensuring quick response of the server to requests and transmission of time-consuming tasks for further processing.
- Distribute messages to multiple recipients.
- Connection offline clients for further data retrieval.
- Increase the reliability and smooth operation of applications.
- Reliability of message delivery.
- High speed message delivery.
- Message Acceptance.

Technical specification

Standards & specs	AMQP 0.9.1 specification (RabbitMQ PDF) · RabbitMQ AMQP 0-9-1 reference
Component class	`TsgcWSPClient_AMQP` (unit `sgcWebSocket_Protocol_AMQP_Client`)
Frameworks	VCL, FireMonkey, Lazarus / FPC, .NET
Platforms	Windows, macOS, Linux, iOS, Android

Main properties

The principal published / public properties used to configure and drive the component. Consult the online help for the full list.

`Client`	References the TsgcWebSocketClient that carries AMQP 0-9-1 frames when connecting over WebSockets.
`Broker`	References a TsgcWSAMQPBroker component so the AMQP protocol runs over raw TCP instead of WebSockets.
`AMQPOptions`	Negotiated AMQP 0-9-1 connection-tune parameters sent to the broker in Connection.StartOk / Connection.TuneOk.
`HeartBeat`	Sends AMQP heartbeat frames periodically to keep the connection alive and to detect silent broker drops.
`Guid`	Unique identifier that binds this subprotocol instance to its WebSocket or broker connection.
`Version`	Read-only string with the sgcWebSockets build version of the AMQP subprotocol component.

Main methods

The principal public methods exposed by the component.

`Close()`	Closes the AMQP connection. Overloaded: clean close with no arguments, or explicit close passing reply-code/text/class/method.

`CloseEx()`	Closes the AMQP connection synchronously and returns True when the broker acknowledged the close.
`OpenChannel()`	Opens a new AMQP channel with the specified name (channel.open).
`OpenChannelEx()`	Opens a new AMQP channel synchronously, waiting for channel.open-ok.
`CloseChannel()`	Closes an AMQP channel (channel.close).
`CloseChannelEx()`	Closes an AMQP channel synchronously and returns True when the broker acknowledged the close.
`DeleteExchange()`	Deletes an exchange (exchange.delete).
`DeleteExchangeEx()`	Deletes an exchange synchronously and returns True on exchange.delete-ok.
`DeleteQueue()`	Deletes a queue (queue.delete).
`DeleteQueueEx()`	Deletes a queue synchronously and returns True on queue.delete-ok.

Public events

The component exposes the following published events; consult the online help for full event-handler signatures.

`OnAMQPAuthentication`	Lets the application choose the SASL mechanism and supply the credentials during the AMQP login.
`OnAMQPBasicCancelConsume`	Fires when the server confirms that a consumer has been cancelled (basic.cancel-ok).
`OnAMQPBasicConsume`	Fires when the server confirms that a consumer has been registered (basic.consume-ok).
`OnAMQPBasicDeliver`	Fires when the broker pushes a message to an active consumer (basic.deliver).
`OnAMQPBasicGetEmpty`	Fires when a synchronous basic.get finds the queue empty (basic.get-empty).
`OnAMQPBasicGetOk`	Fires when a synchronous basic.get call returns a message (basic.get-ok).
`OnAMQPBasicQoS`	Fires when the server confirms that a QoS prefetch setting has been applied (basic.qos-ok).

`OnAMQPBasicRecoverOk`	Fires when the server confirms that unacknowledged messages have been redelivered (basic.recover-ok).
`OnAMQPBasicReturn`	Fires when a published message is returned by the broker because it could not be routed (basic.return).
`OnAMQPChallenge`	Fires when the broker sends a SASL challenge that the client must answer.
`OnAMQPChannelClose`	Fires when a channel is closed by either peer (channel.close).
`OnAMQPChannelFlow`	Fires when the peer asks to pause or resume content traffic on a channel (channel.flow).
`OnAMQPChannelOpen`	Fires when the server confirms that a channel has been opened (channel.open-ok).
`OnAMQPConnect`	Fires after the AMQP connection handshake completes successfully (connection.open-ok).
`OnAMQPDisconnect`	Fires when the server or the client closes the AMQP connection (connection.close).
`OnAMQPException`	Fires when an unhandled exception is raised inside the AMQP protocol or reader thread.
`OnAMQPExchangeDeclare`	Fires when the server confirms that an exchange has been declared (exchange.declare-ok).
`OnAMQPExchangeDelete`	Fires when the server confirms that an exchange has been deleted (exchange.delete-ok).
`OnAMQPHeartBeat`	Fires every time a heartbeat frame is exchanged with the server.
`OnAMQPQueueBind`	Fires when the server confirms that a queue has been bound to an exchange (queue.bind-ok).
`OnAMQPQueueDeclare`	Fires when the server confirms that a queue has been declared (queue.declare-ok).
`OnAMQPQueueDelete`	Fires when the server confirms that a queue has been deleted (queue.delete-ok).
`OnAMQPQueuePurge`	Fires when the server confirms that a queue has been purged (queue.purge-ok).

`OnAMQPQueueUnBind`	Fires when the server confirms that a queue has been unbound from an exchange (queue.unbind-ok).
`OnAMQPTransactionOk`	Fires when the server acknowledges a transaction method: tx.select-ok, tx.commit-ok or tx.rollback-ok.

Quick Start

Drop the component on a form, configure the properties below and activate it. The snippet that follows shows the typical **TsgcWSPClient_AMQP — Connection** configuration sourced from the online help.

> **About this scenario.** AMQP 0.9.1 protocol defines the concept of channels, which allows you to share a single socket connection with several virtual channels, the client implements an internal thread which reads the bytes received and dispatch every message to the correct channel (which already runs in its own thread), so, if you are running an AMQP connection with 5 channels, the client will run 6 threads (5 threads which handle the data of every channel and 1 thread which handles the data of the connection).

Delphi (VCL / FireMonkey)

```
oAMQP := TsgcWSPClient_AMQP.Create(nil);
    oAMQP.AMQPOptions.Locale := 'en_US';
    oAMQP.AMQPOptions.MaxChannels := 100;
    oAMQP.AMQPOptions.MaxFrameSize := 16384;
    oAMQP.AMQPOptions.VirtualHost := '/';
    oAMQP.HeartBeat.Enabled := true;
    oAMQP.HeartBeat.Interval := 60;

    oClient := TsgcWebSocketClient.Create(nil);
    oAMQP.Client := oClient;
    oClient.Specifications.RFC6455 := false;
    oClient.Host := 'www.esegece.com';
    oClient.Port := 5672;
    oClient.Active := True;
```

C++ Builder

```
TsgcWSPClient_AMQP *oAMQP = new TsgcWSPClient_AMQP();
    oAMQP->AMQPOptions->Locale = "en_US";
    oAMQP->AMQPOptions->MaxChannels = 100;
    oAMQP->AMQPOptions->MaxFrameSize = 16384;
    oAMQP->AMQPOptions->VirtualHost = "/";
    oAMQP->HeartBeat->Enabled = true;
    oAMQP->HeartBeat->Interval = 60;

    TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
    oAMQP->Client = oClient;
    oClient->Specifications->RFC6455 = false;
    oClient->Host = "www.esegece.com";
    oClient->Port = 5672;
    oClient->Active = true;
```

.NET (C#)

```
oAMQP = new TsgcWSPClient_AMQP();
    oAMQP.AMQPOptions.Locale = "en_US";
    oAMQP.AMQPOptions.MaxChannels = 100;
    oAMQP.AMQPOptions.MaxFrameSize = 16384;
    oAMQP.AMQPOptions.VirtualHost = "/";
    oAMQP.HeartBeat.Enabled = true;
    oAMQP.HeartBeat.Interval = 60;

    oClient = new TsgcWebSocketClient();
    oAMQP.Client = oClient;
    oClient.Specifications.RFC6455 = false;
    oClient.Host = "www.esegece.com";
    oClient.Port = 5672;
    oClient.Active = true;
```

Common scenarios

The following scenarios are lifted verbatim from the online help. Each shows the configuration and method calls needed to drive the component through a specific real-world flow.

1 · Client AMQP Connect — Basic Usage

Connect to AMQP server without authentication. Define the AMQPOptions property values, virtual host and then set in the TsgcWebSocketClient the Host and Port of the server.

Delphi (VCL / FireMonkey)

```
oAMQP := TsgcWSPClient_AMQP.Create(nil);
oAMQP.AMQPOptions.Locale := 'en_US';
oAMQP.AMQPOptions.MaxChannels := 100;
oAMQP.AMQPOptions.MaxFrameSize := 16384;
oAMQP.AMQPOptions.VirtualHost := '/';
oAMQP.HeartBeat.Enabled := true;
oAMQP.HeartBeat.Interval := 60;

oClient := TsgcWebSocketClient.Create(nil);
oAMQP.Client := oClient;
oClient.Specifications.RFC6455 := false;
oClient.Host := 'www.esegece.com';
oClient.Port := 5672;
oClient.Active := True;
```

C++ Builder

```
oAMQP = new TsgcWSPClient_AMQP();
oAMQP->AMQPOptions->Locale = "en_US";
oAMQP->AMQPOptions->MaxChannels = 100;
oAMQP->AMQPOptions->MaxFrameSize = 16384;
oAMQP->AMQPOptions->VirtualHost = "/";
oAMQP->HeartBeat->Enabled = true;
oAMQP->HeartBeat->Interval = 60;

oClient = new TsgcWebSocketClient();
oAMQP->Client = oClient;
oClient->Specifications->RFC6455 = false;
oClient->Host = "www.esegece.com";
oClient->Port = 5672;
oClient->Active = true;
```

.NET (C#)

```
oAMQP = new TsgcWSPClient_AMQP();
oAMQP.AMQPOptions.Locale = "en_US";
oAMQP.AMQPOptions.MaxChannels = 100;
oAMQP.AMQPOptions.MaxFrameSize = 16384;
oAMQP.AMQPOptions.VirtualHost = "/";
oAMQP.HeartBeat.Enabled = true;
oAMQP.HeartBeat.Interval = 60;

oClient = new TsgcWebSocketClient();
oAMQP.Client = oClient;
oClient.Specifications.RFC6455 = false;
oClient.Host = "www.esegece.com";
oClient.Port = 5672;
oClient.Active = true;
```

2 · Publish Messages

The method PublishMessages is used to send a message to the AMQP server.

Delphi (VCL / FireMonkey)

```
AMQP.PublishMessage('channel_name', 'exchange_name', 'routing_key', 'Hello from sgcWebSockets!!!

procedure OnAMQPBasicReturn(Sender: TObject; const aChannel: string;
 const aReturn: TsgcAMQPFramePayload_Method_BasicReturn;
 const aContent: TsgcAMQPMessageContent);
begin
DoLog('#AMQP_basic_return: ' + aChannel + ' ' + IntToStr(aReturn.ReplyCode) + ' ' + aReturn.Repl
end;
```

C++ Builder

```
AMQP->PublishMessage("channel_name", "exchange_name", "routing_key", "Hello from sgcWebSockets!!

private void OnAMQPBasicReturn(TObject *Sender, const string aChannel,
 const TsgcAMQPFramePayload_Method_BasicReturn *aReturn,
 const TsgcAMQPMessageContent *aContent)
{
DoLog("#AMQP_basic_return: " + aChannel + " " + IntToStr(aReturn->ReplyCode) + " " + aReturn->Re
}
```

.NET (C#)

```
AMQP.PublishMessage("channel_name", "exchange_name", "routing_key", "Hello from sgcWebSockets!!!

private void OnAMQPBasicReturn(TObject Sender, const string aChannel,
  const TsgcAMQPFramePayload_Method_BasicReturn aReturn,
  const TsgcAMQPMessageContent aContent)
{
DoLog("#AMQP_basic_return: " + aChannel + " " + aReturn.ReplyCode.ToString() + " " + aReturn.Rep
}
```

3 · Sending a Close Reason

The AMQP client can inform the server that the connection will be closed and provide information about the reason why it is closing the connection. Use the method Close to request a connection close to the server.

```
Delphi (VCL / FireMonkey)
```

```
oAMQP.Close(541, 'Internal Error');
```

```
C++ Builder
```

```
oAMQP.Close(541, "Internal Error");
```

```
.NET (C#)
```

```
oAMQP.Close(541, "Internal Error");
```

4 · Consume

The method Consume creates a new consumer in the queue, and every time there is a new message this will be delivered automatically to the consumer client.

```
Delphi (VCL / FireMonkey)
```

```
AMQP.Consume('channel_name', 'queue_name', 'consumer_tag');

procedure OnAMQPBasicDeliver(Sender: TObject;
 const aChannel: string;
 const aDeliver: TsgcAMQPFramePayload_Method_BasicDeliver;
 const aContent: TsgcAMQPMessageContent);
begin
 DoLog('#AMQP_basic_deliver: ' + aChannel + ' ' + aDeliver.ConsumerTag + ' ' +
  ' ' + aContent.Body.AsString);
end;
```

C++ Builder

```
AMQP->Consume("channel_name", "queue_name", "consumer_tag");

void OnAMQPBasicDeliver(TObject *Sender, const string aChannel,
 const TsgcAMQPFramePayload_Method_BasicDeliver *aDeliver,
 const TsgcAMQPMessageContent *aContent)
{
 DoLog("#AMQP_basic_deliver: " + aChannel + " " + aDeliver->ConsumerTag + " " +
  " " + aContent->Body->AsString);
}
```

.NET (C#)

```
AMQP.Consume("channel_name", "queue_name", "consumer_tag");

private void OnAMQPBasicGetOk(TObject Sender, const string aChannel,
  const TsgcAMQPFramePayload_Method_BasicDeliver aDeliver,
  const TsgcAMQPMessageContent aContent)
{
DoLog("#AMQP_basic_deliver: " + aChannel + " " + aDeliver.ConsumerTag + " " + aContent.Body.AsSt
}
```

5 · Declare Exchange

This method creates a new exchange or verifies that an Exchange already exists. The method has the following arguments:

Delphi (VCL / FireMonkey)

```
AMQP.DeclareExchange('channel_name', 'exchange_name', 'direct');

procedure OnAMQPExchangeDeclare(Sender: TObject; const aChannel, aExchange: string);
begin
DoLog('#AMQP_exchange_declare: [' + aChannel + '] ' + aExchange);
end;
```

C++ Builder

```
AMQP->DeclareExchange("channel_name", "exchange_name", "direct");

private void OnAMQPExchangeDeclare(TObject *Sender, const string aChannel, const string aExchang
{
DoLog("#AMQP_exchange_declare: [" + aChannel + "] " + aExchange);
}
```

.NET (C#)

```
AMQP.DeclareExchange("channel_name", "exchange_name", "direct");

private void OnAMQPExchangeDeclare(TObject Sender, const string aChannel, const string aExchange
{
DoLog("#AMQP_exchange_declare: [" + aChannel + "] " + aExchange);
}
```

6 · Declare Queue

This method creates a new queue or verifies that a Queue already exists. The method has the following arguments:

Delphi (VCL / FireMonkey)

```
AMQP.DeclareQueue('channel_name', 'queue_name');

procedure OnAMQPQueueDeclare(Sender: TObject; const aChannel, aQueue: string;
 aMessageCount, aConsumerCount: Integer);
begin
DoLog('#AMQP_queue_declare: [' + aChannel + '] ' + aQueue);
end;
```

C++ Builder

```
AMQP→DeclareQueue("channel_name", "queue_name");

private void OnAMQPExchangeDeclare(TObject *Sender, const string aChannel, const string aQueue,
 int aMessageCount, int aConsumerCount)
{
DoLog("#AMQP_queue_declare: [" + aChannel + "] " + aQueue));
}
```

.NET (C#)

```
AMQP.DeclareQueue("channel_name", "queue_name");

private void OnAMQPExchangeDeclare(TObject Sender, const string aChannel, const string aQueue,
 int aMessageCount, int aConsumerCount)
{
DoLog("#AMQP_queue_declare: [" + aChannel + "] " + aQueue));
}
```

Sources used to build this document

Every external claim links back to a primary source. The online-help references decode the canonical deep-link the company maintains for this component.

Primary standard / spec — AMQP 0.9.1 specification (RabbitMQ PDF)	www.rabbitmq.com/resources/specs/amqp0-9-1.pdf
Primary standard / spec — RabbitMQ AMQP 0-9-1 reference	www.rabbitmq.com/amqp-0-9-1-reference
Online help — component page	www.esegece.com/help/sgcWebSockets/Components/Protocols/Subprotocols/AMQP/Protocol_AMQP.htm
Online help — class reference (TsgcWSPClient_AMQP)	www.esegece.com/help/sgcWebSockets/Components/Protocols/Subprotocols/AMQP/TsgcWSPClient_AMQP.htm
Delphi demo project (in the sgcWebSockets package)	`Demos\02.WebSocket_Protocols\10.AMQP_Client`
.NET demo project (in the sgcWebSockets package)	`.net\demos\02.WebSocket_Protocols\10.AMQP_Client`
Component page	www.esegece.com/products/websockets/protocols/amqp-091/
Product page	www.esegece.com/products/websockets/

> **Document scope.** This document covers the publicly-documented surface of the AMQP 0.9.1 component shipped with sgcWebSockets. For full property, method and event reference consult the online help linked above.