SGCWEBSOCKETS · TECHNICAL DOCUMENT

MQTT Client

Lightweight publish/subscribe messaging for Delphi, C++ Builder and .NET — full MQTT 3.1.1 and 5.0 support over TCP or WebSocket transports.

eSeGeCe

WWW.ESEGECE.COM

Overview

MQTT (Message Queuing Telemetry Transport) is a lightweight publish/subscribe messaging protocol originally designed for constrained devices and low-bandwidth networks. It is the protocol of choice for IoT, telemetry, mobile messaging and machine-to-machine communication, and is supported natively by every major IoT cloud (AWS IoT Core, Azure IoT Hub, Google IoT Core), broker (Mosquitto, HiveMQ, EMQX, RabbitMQ) and SDK.

The sgcWebSockets `TsgcWSPClient_MQTT` component delivers a full MQTT client implementation written in native Delphi, supporting versions **3.1.1** and **5.0**, plain-text and TLS-secured transports, and both standard TCP and MQTT-over-WebSocket connections. It plugs into any sgcWebSockets server stack via `TsgcWebSocketClient` and ships with the WatchDog auto-reconnect, three-level QoS and Last-Will-Testament semantics defined by the OASIS specifications.

At a glance

COMPONENT CLASS
`TsgcWSPClient_MQTT`

PROTOCOL VERSIONS
MQTT 3.1.1 & MQTT 5.0

TRANSPORTS
TCP, TLS, WebSocket, WebSocket Secure

QOS LEVELS
0 (at most once) / 1 (at least once) / 2 (exactly once)

PLATFORMS
Windows, macOS, Linux, iOS, Android

FRAMEWORKS
VCL, FireMonkey, Lazarus / FPC, .NET

Features

- **Versions 3.1.1 and 5.0** — full coverage of both specifications, including MQTT 5.0 reason codes, properties, topic aliases, session expiry, request/response and shared subscriptions.
- **Publish / Subscribe** — one-to-many message distribution with topic filters, wildcards (`+` , `#`), retained messages and queued deliveries.

- **Three QoS levels** — at-most-once (fire and forget), at-least-once (PUBACK), exactly-once (4-step PUBREC / PUBREL / PUBCOMP handshake).
- **Last Will and Testament** — the broker publishes a configured payload to a configured topic when the client disconnects abnormally.
- **Authentication** — username / password, client certificates, enhanced SASL-style challenge/response (MQTT 5.0).
- **Secure connections** — TLS 1.2 / 1.3 via OpenSSL or SChannel, mutual TLS, ALPN.
- **HeartBeat and WatchDog** — keep-alive PING / PINGRESP and automatic reconnect with subscription replay.
- **WebSocket transport** — MQTT over WebSockets, the standard transport for browser-hosted MQTT and JavaScript-paired clients.

Technical specification

Protocol	MQTT 3.1.1 (OASIS), MQTT 5.0 (OASIS) — v3.1.1 spec, v5.0 spec
Transport	TCP (port 1883 default), TLS (port 8883 default), WebSocket, WebSocket Secure
QoS levels	0 / 1 / 2 (At-most-once, At-least-once, Exactly-once)
Authentication	Username / Password, X.509 client certificates, Enhanced SASL (MQTT 5.0)
TLS	1.2 and 1.3 via OpenSSL or SChannel
Topic structure	Hierarchical (`level1/level2/level3`), wildcards `+` (single) and `#` (multi)
Max payload	256 MB per message
Component class	`TsgcWSPClient_MQTT` (unit `sgcWebSocket_Protocol_MQTT_Client`)
Companion class	`TsgcWebSocketClient` (for transport) or `TsgcTCPClient` (raw TCP)

Main properties

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

`Authentication`	Sends a UserName and Password in the MQTT CONNECT packet to authenticate the client against the broker.
`LastWillTestament`	Registers a Will message the broker publishes on behalf of this client if the connection is lost ungracefully.
`Client`	References the TsgcWebSocketClient that carries MQTT frames when connecting over WebSockets.
`Broker`	References a TsgcWSMQTTBroker component so the MQTT protocol runs over raw TCP instead of WebSockets.
`QoS`	Default Quality of Service level and retry/timeout behavior for outbound QoS 1 and QoS 2 messages.
`HeartBeat`	Sends MQTT PINGREQ packets periodically to keep the session alive and to detect silent broker drops.

`MQTTVersion`	Selects the MQTT protocol level advertised in the CONNECT packet (3.1.1 or 5.0).
`ConnectProperties`	MQTT 5.0 properties sent with the CONNECT packet: session expiry, receive maximum, packet size, topic aliases and extended authentication.
`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 MQTT subprotocol component.

Main methods

The principal public methods exposed by the component.

`PublishAndWait()`	Publishes a message and blocks until the broker acknowledges it or the timeout elapses.
`Connect()`	Sends an MQTT CONNECT packet to the broker over the underlying WebSocket transport.
`Disconnect()`	Sends an MQTT DISCONNECT packet to end the session cleanly.
`Ping()`	Sends a PINGREQ control packet so the broker knows the session is still alive.
`Publish()`	Publishes a message to a topic using the requested QoS and retain flags.
`Subscribe()`	Subscribes the client to one or more topic filters at the requested QoS.
`UnSubscribe()`	Removes one or more active topic subscriptions from the current MQTT session.
`Auth()`	Sends an MQTT 5.0 AUTH packet to perform or continue enhanced authentication with the broker.
`WriteData()`	Writes a pre-built MQTT control packet directly over the WebSocket transport.

Public events

The component exposes the following published events; these mirror the protocol's packet types and let your application observe each step of the MQTT exchange.

`OnMQTTConnect`	Broker accepted the CONNECT and returned CONNACK with reason code and properties.

`OnMQTTDisconnect`	Broker (or local) closed the connection. MQTT 5.0 includes reason code and reason string.
`OnMQTTPublish`	An incoming PUBLISH for a topic this client is subscribed to. Provides topic, payload and properties.
`OnMQTTSubscribe`	Broker SUBACK acknowledgement of a subscription request, with per-topic reason codes.
`OnMQTTUnSubscribe`	Broker UNSUBACK acknowledgement of an unsubscribe request.
`OnMQTTPubAck` / `PubRec` / `PubRel` / `PubComp`	The QoS-1 and QoS-2 handshake steps; observe to track in-flight reliability.
`OnMQTTPing`	Keep-alive PING / PINGRESP exchange; useful for diagnostics.
`OnMQTTAuth`	MQTT 5.0 enhanced-authentication challenge/response round.

Quick Start

Drop a `TsgcWebSocketClient` (or `TsgcTCPClient`) and a `TsgcWSPClient_MQTT` on a form, set the transport's host / port, point the MQTT component's `Client` property at the transport, wire the `OnMQTTPublish` handler, then activate.

> **Tip.** For brokers like Mosquitto, HiveMQ, EMQX or AWS IoT, use port `1883` (plain) or `8883` (TLS). For MQTT-over-WebSocket use port `8083` or `8084` — check your broker's documentation.

Delphi (VCL / FireMonkey)

```
unit uMQTTDemo.pas
```

```
uses
  sgcWebSocket_Classes,
  sgcWebSocket_Client,
  sgcWebSocket_Protocol_MQTT_Client,
  sgcWebSocket_Protocol_MQTT_Message;

var
  Transport: TsgcWebSocketClient;
  MQTT:      TsgcWSPClient_MQTT;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Transport := TsgcWebSocketClient.Create(Self);
  Transport.Host := 'test.mosquitto.org';
  Transport.Port := 1883;

  MQTT := TsgcWSPClient_MQTT.Create(Self);
  MQTT.Client := Transport;
  MQTT.OnMQTTConnect := MQTTConnect;
  MQTT.OnMQTTPublish := MQTTPublish;

  Transport.Active := True;
end;

procedure TForm1.MQTTConnect(Connection: TsgcWSConnection;
  const Session: Boolean;
  const ReasonCode: Integer;
  const ReasonName: string;
  const ConnectProperties: TsgcWSMQTTCONNACKProperties);
begin
  // connected — subscribe to a topic with QoS 1
  MQTT.Subscribe('sensors/temperature/#', mtqsAtLeastOnce);
end;

procedure TForm1.MQTTPublish(Connection: TsgcWSConnection;
  aTopic, aText: string;
  PublishProperties: TsgcWSMQTTPublishProperties);
begin
  Memo1.Lines.Add('[' + aTopic + '] ' + aText);
end;

procedure TForm1.btnPublishClick(Sender: TObject);
begin
  MQTT.Publish('sensors/temperature/lab-1', '21.4', mtqsAtLeastOnce, False);
end;
```

C++ Builder

```
file: MQTTDemo.cpp
```

```
// Drop TsgcWebSocketClient (named Transport) and
// TsgcWSPClient_MQTT (named MQTT) on the form,
// set MQTT->Client = Transport at design time, then:

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Transport->Host   = "test.mosquitto.org";
  Transport->Port   = 1883;
  Transport->Active = true;
}

void __fastcall TForm1::MQTTMQTTConnect(TsgcWSConnection *Connection,
  const bool Session, const int ReasonCode,
  const UnicodeString ReasonName,
  const TsgcWSMQTTCONNACKProperties &ConnectProperties)
{
  MQTT->Subscribe("sensors/temperature/#", mtqsAtLeastOnce);
}

void __fastcall TForm1::MQTTMQTTPublish(TsgcWSConnection *Connection,
  UnicodeString aTopic, UnicodeString aText,
  TsgcWSMQTTPublishProperties &PublishProperties)
{
  Memo1->Lines->Add("[" + aTopic + "] " + aText);
}

void __fastcall TForm1::btnPublishClick(TObject *Sender)
{
  MQTT->Publish("sensors/temperature/lab-1", "21.4",
                mtqsAtLeastOnce, false);
}
```

.NET (C#)

file: FRMMQTT.cs

```
using esegece.sgcWebSockets;

public partial class FRMMQTT : Form
{
    private TsgcWebSocketClient Transport;
    private TsgcWSPClient_MQTT  MQTT;

    public FRMMQTT()
    {
        InitializeComponent();

        Transport      = new TsgcWebSocketClient();
        Transport.Host = "test.mosquitto.org";
        Transport.Port = 1883;

        MQTT                       = new TsgcWSPClient_MQTT();
        MQTT.Client                = Transport;
        MQTT.HeartBeat.Enabled     = true;
        MQTT.HeartBeat.Interval    = 5;

        MQTT.OnMQTTConnect += (conn, session, code, name, props) =>
        {
            // connected — subscribe with QoS 1
            MQTT.Subscribe("sensors/temperature/#",
                TmqttQoS.mtqsAtLeastOnce);
        };

        MQTT.OnMQTTPublish += (conn, topic, text, props) =>
        {
            memoLog.AppendText("[" + topic + "] " + text + "\r\n");
        };

        Transport.Active = true;
    }

    private void btnPublishClick(object sender, EventArgs e)
    {
        MQTT.Publish("sensors/temperature/lab-1", "21.4",
                     TmqttQoS.mtqsAtLeastOnce, false);
    }
}
```

Common scenarios

1 · Subscribe with a topic-filter wildcard

Topic filters use `+` for a single-level wildcard and `#` for a multi-level wildcard. The expression `devices/+/status` matches `devices/lab-1/status` and `devices/lab-2/status` but not `devices/lab-1/health/cpu`; the expression `logs/#` matches everything starting with `logs/`.

```
MQTT.Subscribe('devices/+/status', mtqsAtLeastOnce);
MQTT.Subscribe('logs/#',           mtqsAtMostOnce);
```

2 · Publish with retained flag and QoS 2

A retained message is stored by the broker and delivered to any new subscriber whose filter matches the topic. QoS 2 guarantees exactly-once delivery via the four-step PUBREC / PUBREL / PUBCOMP handshake.

```
MQTT.Publish('config/firmware', '1.4.2',
  mtqsExactlyOnce,
  True);  // retained
```

3 · Last Will and Testament

The broker publishes the configured payload to the configured topic when the connection is closed abnormally (network drop, crash, etc.). Subscribers see the message just as if the publisher had explicitly sent it.

```
MQTT.LastWill.Enabled := True;
MQTT.LastWill.Topic   := 'devices/lab-1/status';
MQTT.LastWill.Message := 'offline';
MQTT.LastWill.QoS     := mtqsAtLeastOnce;
MQTT.LastWill.Retain  := True;
```

4 · MQTT over WebSocket / WebSocket-Secure

For browser-paired clients or networks that block port 1883/8883, MQTT runs natively over WebSocket. Use the standard WebSocket subprotocol `mqtt`.

```
Transport.Host    := 'broker.hivemq.com';
Transport.Port    := 8884;
Transport.TLS     := True;
Transport.URL     := '/mqtt';
Transport.Subprotocol := 'mqtt';
```

Sources used to build this document

Every claim in this document maps back to a primary source. Use the links below to drill into the official references, the bundled help and the demo project that the code samples were reduced from.

Protocol specification — MQTT 3.1.1	docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html
Protocol specification — MQTT 5.0	docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html
Component online help	esegece.com/help/sgcWebSockets — Protocol MQTT
Component class reference	esegece.com/help/sgcWebSockets — TsgcWSPClient_MQTT
Bundled demo (Delphi)	`Demos\02.WebSocket_Protocols\08.MQTT_Client\ClientMQTT.dpr`
Bundled demo (.NET)	`.net\demos\02.WebSocket_Protocols\08.MQTT_Client\MQTT`
Sample MQTT broker (public)	test.mosquitto.org · HiveMQ public broker
Component page	esegece.com/products/websockets/protocols/mqtt
Product page	esegece.com/products/websockets

> **Document scope.** This document covers the publicly-documented surface of the MQTT client component shipped with sgcWebSockets. For complete property, method and event reference, including every MQTT 5.0 property and reason code, consult the online help linked above.