

Custom Protocol — sgc (Generic PubSub)

sgc generic publish-subscribe custom subprotocol — channels, queued messages and broker-style fan-out built into sgcWebSockets.

Overview

This is default sub-protocol implemented using "JSONRPC 2.0" messages, every time you send a message using this protocol, a JSON object is created with the following properties:

At a glance

COMPONENT CLASS

`TsgcWSPClient_sgc`

STANDARDS / SPEC

—

TRANSPORTS

TCP, TLS

PLATFORMS

Windows, macOS, Linux, iOS, Android

FRAMEWORKS

VCL, FireMonkey, Lazarus / FPC, .NET

EDITION

Standard / Professional / Enterprise

Features

- A messaging transport that is agnostic to the content of the payload
- Acknowledgment of messages sent.
- Implements QoS (Quality of Service) for message delivery.

Technical specification

Component class	<code>TsgcWSPClient_sgc</code> (unit <code>sgcWebSocket_Protocol_sgc_Client</code>)
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.

<code>Client</code>	WebSocket client component used as transport for the sgc subprotocol.
<code>Broker</code>	Optional broker component that relays subprotocol messages between peers.
<code>QoS</code>	Quality of Service options (Level, Interval, Timeout) for acknowledged delivery.
<code>Guid</code>	Unique identifier used to route messages to a specific protocol instance.
<code>Version</code>	Read-only subprotocol version string.

Main methods

The principal public methods exposed by the component.

<code>Subscribe()</code>	Subscribes the client to a custom channel.
<code>UnSubscribe()</code>	Unsubscribes the client from a custom channel.
<code>UnSubscribeAll()</code>	Unsubscribes the client from all active channel subscriptions.
<code>Publish()</code>	Publishes a message to all clients subscribed to a channel.
<code>StartTransaction()</code>	Begins a new transaction; subsequent messages are queued until Commit or RollBack.
<code>WriteData()</code>	Sends a plain text message to the server using the sgc message envelope.

Broadcast()	Broadcasts a message to all connected clients, optionally filtered by channel.
RPC()	Sends a remote procedure call request and awaits a Result or Error response.
Notify()	Sends a one-way notification to the server that does not expect a response.
Commit()	Commits the current transaction so the server processes all queued messages.

Public events

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

OnAcknowledge	Fires when the server acknowledges receipt of a QoS 1 or 2 message.
OnBinary	Fires when a binary frame arrives; payload is delivered as a TMemoryStream.
OnConnect	Fires after the WebSocket handshake completes and the sgc subprotocol is initialized.
OnDisconnect	Fires when the connection is closed, reporting the close code.
OnError	property OnError: TsgcWSErrorEvent; // TsgcWSErrorEvent = procedure(Connection: TsgcWSConnection; const Error: string) of object __property TsgcWSErrorEvent OnError; // typedef void __fastcall (__clos...
OnEvent	property OnEvent: TsgcWSCustomEvent; // TsgcWSCustomEvent = procedure(Connection: TsgcWSConnection; const Channel, Text: string) of object __property TsgcWSCustomEvent OnEvent; // typedef void __fastc...
OnException	property OnException: TsgcExceptionEvent; // TsgcExceptionEvent = procedure(Connection: TsgcWSConnection; E: Exception) of object __property TsgcExceptionEvent OnException; // typedef void __fastcall ...
OnFragmented	Fires for fragmented WebSocket frames, exposing OpCode and continuation flag.
OnMessage	property OnMessage: TsgcWSMessageEvent; // TsgcWSMessageEvent = procedure(Connection: TsgcWSConnection; const Text: string) of object __property TsgcWSMessageEvent OnMessage; // typedef void __fastcal...
OnRPCError	Fires when the server returns an error response to an RPC request.
OnRPCResult	property OnRPCResult: TsgcWSRPCResultEvent; // TsgcWSRPCResultEvent = procedure(Connection: TsgcWSConnection; Id, Result: string) of object

__property TsgcWSRPCResultEvent OnRPCResult; // typedef void...

OnRawMessage

Fires before the component parses a message; set Handled to True to suppress default processing.

OnSession

Fires after a successful connection or GetSession request with the assigned session Guid.

OnSubscription

```
property OnSubscription: TsgcWSSubscriptionEvent; //
TsgcWSSubscriptionEvent = procedure(Connection: TsgcWSConnection; const
Subscription: String) of object __property TsgcWSSubscriptionEvent
OnSubscr...
```

OnUnSubscription

```
property OnUnSubscription: TsgcWSSubscriptionEvent; //
TsgcWSSubscriptionEvent = procedure(Connection: TsgcWSConnection; const
Subscription: String) of object __property TsgcWSSubscriptionEvent OnUnSu...
```

Quick Start

Drop the component on a form, configure the properties below and activate it. The snippet that follows shows the typical **Client AMQP Connect — Basic Usage** configuration sourced from the online help.

About this scenario. 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;
```

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 AMQP1 Connect — Basic Usage

Connect to an AMQP 1.0.0 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)

```
// Creating AMQP client
oAMQP := TsgcWSPClient_AMQP1.Create(nil);
// Creating WebSocket client
oClient := TsgcWebSocketClient.Create(nil);
// Setting WebSocket specifications
oClient.Specifications.RFC6455 := False;
// Setting WebSocket client properties
oClient.Host := 'amqp_host_address';
oClient.Port := 5672;
// Assigning WebSocket client to AMQP client
oAMQP.Client := oClient;
// Activating WebSocket client
oClient.Active := True;
```

C++ Builder

```
// Creating AMQP client
oAMQP = new TsgcWSPClient_AMQP1(this);
// Creating WebSocket client
oClient = new TsgcWebSocketClient(this);
// Setting WebSocket specifications
oClient->Specifications->RFC6455 = false;
// Setting WebSocket client properties
oClient->Host = L"amqp_host_address";
oClient->Port = 5672;
// Assigning WebSocket client to AMQP client
oAMQP->Client = oClient;
// Activating WebSocket client
oClient->Active = true;
```

.NET (C#)

```
oAMQP = new TsgcWSPClient_AMQP1(this);
// Creating WebSocket client
oClient = new TsgcWebSocketClient(this);
// Setting WebSocket specifications
oClient.Specifications.RFC6455 = false;
// Setting WebSocket client properties
oClient.Host = "amqp_host_address";
oClient.Port = 5672;
// Assigning WebSocket client to AMQP client
oAMQP.Client = oClient;
// Activating WebSocket client
oClient.Active = true;
```

2 · Client MQTT Connect — Basic Usage

Connect to Mosquitto MQTT server using websocket protocol. Subscribe to topic: "topic1" after connect.

Delphi (VCL / FireMonkey)

```
oClient := TsgcWebSocketClient.Create(nil);
oClient.Host := 'test.mosquitto.org';
oClient.Port := 8080;
oMQTT := TsgcWSPClient_MQTT.Create(nil);
oMQTT.Client := oClient;
oClient.Active := True;

procedure OnMQTTConnect(Connection: TsgcWSConnection; const Session: Boolean; const ReasonCode:
  const ReasonName: string; const ConnectProperties: TsgcWSMQTTCONNACKProperties);
begin
oMQTT.Subscribe('topic1');
end;
```

C++ Builder

```

oClient = new TsgcWebSocketClient();
oClient→Host = "test.mosquitto.org";
oClient→Port = 8080;
oMQTT = new TsgcWSPClient_MQTT();
oMQTT→Client = oClient;
oClient→Active = true;

void OnMQTTConnect(TsgcWSConnection *Connection, const bool Session, const int ReasonCode,
    const string ReasonName, const TsgcWSMQTTCONNACKProperties *ConnectProperties);
{
oMQTT→Subscribe("topic1");
}

```

.NET (C#)

```

oClient = new TsgcWebSocketClient();
oClient.Host = "test.mosquitto.org";
oClient.Port = 8080;
oMQTT = TsgcWSPClient_MQTT.Create(nil);
oMQTT.Client = oClient;
oClient.Active = true;

void OnMQTTConnect(TsgcWSConnection Connection, bool Session, int ReasonCode,
    string ReasonName, TsgcWSMQTTCONNACKProperties ConnectProperties);
{
oMQTT.Subscribe("topic1");
}

```

3 · Subscribe QoS = At Least Once

You can Subscribe to a Topic using method Subscribe from TsgcWSPClient_MQTT. This method has the following parameters:

Delphi (VCL / FireMonkey)

```
MQTT.Subscribe('topic1', mtqsAtLeastOnce);
```

C++ Builder

```
MQTT→Subscribe("topic1", mtqsAtLeastOnce);
```

.NET (C#)

```
MQTT.Subscribe("topic1", TmqttQoS.mtqsAtLeastOnce);
```

4 · Subscribe Topic

Subscribe to Topic "topic1" after a successful connection.

Delphi (VCL / FireMonkey)

```
oClient := TsgcWebSocketClient.Create(nil);
oClient.Host := 'test.mosquitto.org';
oClient.Port := 8080;
oMQTT := TsgcWSPClient_MQTT.Create(nil);
oMQTT.Client := oClient;
oClient.Active := True;

procedure OnMQTTConnect(Connection: TsgcWSConnection; const Session: Boolean; const ReasonCode:
  const ReasonName: string; const ConnectProperties: TsgcWSMQTTCONNACKProperties);
begin
oMQTT.Subscribe('topic1');
end;
```

C++ Builder

```
oClient = new TsgcWebSocketClient();
oClient->Host = "test.mosquitto.org";
oClient->Port = 8080;
oMQTT = new TsgcWSPClient_MQTT();
oMQTT->Client = oClient;
oClient->Active = true;

void OnMQTTConnect(TsgcWSConnection *Connection, const bool Session, const int ReasonCode,
  const string ReasonName, const TsgcWSMQTTCONNACKProperties *ConnectProperties);
{
oMQTT->Subscribe("topic1");
}
```

.NET (C#)

```

oClient = new TsgcWebSocketClient();
oClient.Host = "test.mosquitto.org";
oClient.Port = 8080;
oMQTT = TsgcWSPClient_MQTT.Create(nil);
oMQTT.Client = oClient;
oClient.Active = true;

void OnMQTTConnect(TsgcWSConnection Connection, bool Session, int ReasonCode, string ReasonName,
    TsgcWSMQTTCONNACKProperties ConnectProperties);
{
oMQTT->Subscribe("topic1");
}

```

5 · 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  
}
```

6 · Publish a simple message

You can publish messages to all subscribers of a Topic using Publish method, which has the following parameters:

Delphi (VCL / FireMonkey)

```
MQTT.Publish('topic1', 'Hello Subscribers topic1');
```

C++ Builder

```
MQTT->Publish("topic1", "Hello Subscribers topic1");
```

.NET (C#)

```
MQTT.Publish("topic1", "Hello Subscribers topic1");
```

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.

Online help — component page www.egegece.com/help/sgcWebSockets/Components/Protocols/Subprotocols/Default/Protocol_Default.htm

Online help — class reference (TsgcWSPClient_sgc) www.egegece.com/help/sgcWebSockets/Components/Protocols/Subprotocols/Default/TsgcWSPClient_sgc.htm

Delphi demo project (in the sgcWebSockets package) `Demos\02.WebSocket_Protocols\01.SGC_Generic_PubSub_Protocol`

.NET demo project (in the sgcWebSockets package) `.net\demos\02.WebSocket_Protocols\01.SGC_Generic_PubSub_Protocol`

Component page www.egegece.com/products/websockets/protocols/custom-sgc/

Product page www.egegece.com/products/websockets/

Document scope. This document covers the publicly-documented surface of the Custom Protocol — sgc (Generic PubSub) component shipped with sgcWebSockets. For full property, method and event reference consult the online help linked above.