SGCWEBSOCKETS · TECHNICAL DOCUMENT

Custom Protocol — Presence

Presence custom subprotocol — connection-state, channels and member metadata broadcast across the room, in the spirit of Pusher Channels.

eSeGeCe

WWW.ESEGECE.COM

Overview

Presence protocol allows you to know who is subscribed to a channel, this makes it easier to create chat applications and know who is online, example: game users, chat rooms, users viewing the same document...

At a glance

COMPONENT CLASS `TsgcWSPClient_Presence`	STANDARDS / SPEC —
TRANSPORTS **TCP, TLS**	PLATFORMS **Windows, macOS, Linux, iOS, Android**
FRAMEWORKS **VCL, FireMonkey, Lazarus / FPC, .NET**	EDITION **Standard / Professional / Enterprise**

Features

- By default user is identified by a name, but this can be customized passing more data: email, company, twitter...
- Events to Authorize if a Channel can be created, if a member is allowed...
- Every time a new member joins a channel, all members are notified.
- Publish messages to all channel subscribers.
- Low memory usage.

Technical specification

Component class	`TsgcWSPClient_Presence` (unit `sgcWebSocket_Protocol_Presence_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`	WebSocket client component used as transport for the Presence subprotocol.
`Broker`	Optional broker component that relays Presence messages between peers.
`Presence`	Local member identity (name and arbitrary info) published to the server on join.
`Acknowledgment`	Enables per-message acknowledgments for Presence traffic and configures retry behavior.
`EncodeBase64`	Base64-encodes the message payload before sending to keep binary-safe transport.
`Guid`	Unique identifier used to route messages to a specific Presence protocol instance.
`Version`	Read-only Presence subprotocol version string.

Main methods

The principal public methods exposed by the component.

`Subscribe()`	Joins the local member to the given channel.
`UnSubscribe()`	Removes the local member from the given channel.
`Publish()`	Publishes a text message to every member of a channel.

`WriteData()`	Low-level raw write hook; disabled on the Presence client and kept only for API compatibility.
`Invite()`	Invites another member by id to join a channel.

Public events

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

`OnChannelInvitation`	Fires when this member is invited to a channel; set Accept to join.
`OnChannelInvitationResponse`	TsgcWSPClient_Presence › Events › OnChannelInvitationResponse
`OnConnect`	Fires when the underlying WebSocket transport has connected to the server.
`OnDisconnect`	property OnDisconnect: TsgcWSDisconnectEvent; // TsgcWSDisconnectEvent = procedure(Connection: TsgcWSConnection; Code: Integer) of object __property TsgcWSDisconnectEvent OnDisconnect; // typedef void...
`OnError`	Fires when the transport reports a protocol-level error string.
`OnErrorMemberChannel`	Fires when a subscription or member-related operation fails on the server.
`OnErrorPublishMsg`	property OnErrorPublishMsg: TsgcWSPresencePublishMsgErrorEvent; // TsgcWSPresencePublishMsgErrorEvent = procedure(Connection: TsgcWSConnection; const aError: TsgcWSPresenceError; const aMsg: TsgcWSPre...
`OnException`	Fires when an unhandled exception is raised while processing a Presence message.
`OnGetMembers`	Fires with the list of members returned by a GetMembers request.
`OnNewChannelMember`	property OnNewChannelMember: TsgcWSPresenceNewMemberChannelEvent; // TsgcWSPresenceNewMemberChannelEvent = procedure(Connection: TsgcWSConnection; const aChannel: TsgcWSPresenceChannel; const aMember:...
`OnNewMember`	property OnNewMember: TsgcWSPresenceNewMemberEvent; // TsgcWSPresenceNewMemberEvent = procedure(aConnection:

	TsgcWSConnection; const aMember: TsgcWSPresenceMember) of object __property TsgcWSPresenceN...
`OnPublishMsg`	property OnPublishMsg: TsgcWSPresencePublishMsgEvent; // TsgcWSPresencePublishMsgEvent = procedure(Connection: TsgcWSConnection; const aMsg: TsgcWSPresenceMsg; const aChannel: TsgcWSPresenceChannel; c...
`OnRawMessage`	Fires for every incoming frame before Presence parsing; set Handled to skip default handling.
`OnRemoveChannelMember`	property OnRemoveChannelMember: TsgcWSPresenceRemoveMemberChannelEvent; // TsgcWSPresenceRemoveMemberChannelEvent = procedure(Connection: TsgcWSConnection; const aChannel: TsgcWSPresenceChannel; const...
`OnRemoveMember`	property OnRemoveMember: TsgcWSPresenceRemoveMemberEvent; // TsgcWSPresenceRemoveMemberEvent = procedure(aConnection: TsgcWSConnection; aMember: TsgcWSPresenceMember) of object __property TsgcWSPresen...
`OnSession`	Fires when the server returns the session id assigned to this member.

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.esegece.com/help/sgcWebSockets/Components/Protocols/Subprotocols/Presence/Protocol_Presence.htm
Online help — class reference (TsgcWSPClient_Presence)	www.esegece.com/help/sgcWebSockets/Components/Protocols/Subprotocols/Presence/TsgcWSPClient_Presence.htm
Delphi demo project (in the sgcWebSockets package)	`Demos\02.WebSocket_Protocols\03.Presence_Protocol`
.NET demo project (in the sgcWebSockets package)	`.net\demos\02.WebSocket_Protocols\03.Presence_Protocol`
Component page	www.esegece.com/products/websockets/protocols/custom-presence/
Product page	www.esegece.com/products/websockets/

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