SGCWEBSOCKETS · TECHNICAL DOCUMENT

Custom Protocol — Dataset

sgcWebSockets custom subprotocol that streams TDataset changes over WebSocket — server publishes updates, clients apply them automatically.

eSeGeCe

WWW.ESEGECE.COM

Overview

This protocol inherits from Protocol Default and it's useful if you want to broadcast dataset changes over clients connected to this protocol. It can be used in 2 modes:

At a glance

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

Features

- Native Delphi implementation with full ANSI/Unicode support.

Technical specification

Component class	`TsgcWSPClient_Dataset` (unit `sgcWebSocket_Protocol_Dataset_Client`)
Frameworks	VCL, FireMonkey, Lazarus / FPC
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 Dataset subprotocol.
`Broker`	Optional broker component that relays subprotocol messages between peers.
`AutoSubscribe`	Subscribe automatically to the internal Dataset channels after connecting.
`QoS`	Quality of Service options (Level, Interval, Timeout) for acknowledged delivery.
`DataSet`	TDataSet instance that the component keeps in sync with the server.
`NotifyUpdates`	Send a message to the server each time the local Dataset changes.
`ApplyUpdates`	Apply Dataset changes received from the server to the local Dataset.
`UpdateMode`	Selects which fields are sent on update: all, changed, or full refresh.
`FormatSettings`	Shared format settings used to serialize numeric and date/time fields.
`AutoEscapeText`	Automatically escape/unescape reserved characters inside string field values.

Main methods

The principal public methods exposed by the component.

`Subscribe()`	Subscribes the client to a custom channel.

`UnSubscribe()`	Unsubscribes the client from a custom channel.
`UnSubscribeAll()`	Unsubscribes the client from all active channel subscriptions.
`Subscribe_all()`	Subscribes to the three internal Dataset channels (new, update, delete).
`UnSubscribe_all()`	Unsubscribes from the three internal Dataset channels (new, update, delete).
`Publish()`	Publishes a message to all clients subscribed to a channel.
`StartTransaction()`	Begins a new transaction; subsequent messages are queued until Commit or RollBack.
`WriteData()`	Sends a plain text message to the server using the sgc message envelope.
`Synchronize()`	Requests the full Dataset content from the server to refresh the local copy.
`Broadcast()`	Broadcasts a message to all connected clients, optionally filtered by channel.

Public events

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

`OnAcknowledgment`	Fires when the server acknowledges receipt of a QoS 1 or 2 message.
`OnAfterDeleteRecord`	Fires after a delete received from the server has been applied to the local Dataset.
`OnAfterNewRecord`	Fires after a new record received from the server has been inserted in the local Dataset.
`OnAfterSynchronize`	Fires when the server signals that the Synchronize batch has ended.
`OnAfterUpdateRecord`	Fires after an update received from the server has been applied to the local Dataset.
`OnBeforeDatasetUpdate`	Fires before any Dataset message from the server is applied locally; set Handled to skip it.
`OnBeforeDeleteRecord`	Fires before a delete received from the server is applied to the local Dataset.
`OnBeforeNewRecord`	Fires before a new record received from the server is inserted in the local Dataset.
`OnBeforeSynchronize`	Fires when the server announces the start of a Synchronize batch.

`OnBeforeUpdateRecord`	Fires before an update received from the server is applied to the local Dataset.
`OnBinary`	Fires when a binary frame arrives; payload is delivered as a TMemoryStream.
`OnConnect`	Fires after the WebSocket handshake completes and the Dataset 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...
`OnMetaData`	Fires when the server answers a GetMetaData request with the Dataset field definitions.
`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.esegece.com/help/sgcWebSockets/Components/Protocols/Subprotocols/Dataset/Protocol_Dataset.htm
Online help — class reference (TsgcWSPClient_Dataset)	www.esegece.com/help/sgcWebSockets/Components/Protocols/Subprotocols/Dataset/TsgcWSPClient_Dataset.htm
Delphi demo project (in the sgcWebSockets package)	`Demos\02.WebSocket_Protocols\05.DataSet_Quotes_Protocol`
Component page	www.esegece.com/products/websockets/protocols/custom-dataset/
Product page	www.esegece.com/products/websockets/

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