

Custom Protocol — Files

File-transfer custom subprotocol — chunked, resumable upload and download over WebSocket with progress notifications.

Overview

This protocol allows sending files using binary WebSocket transport. It can handle big files with a low memory usage.

At a glance

COMPONENT CLASS

`TsgcWSPClient_Files`

STANDARDS / SPEC

—

TRANSPORTS

TCP, TLS

PLATFORMS

Windows, macOS, Linux, iOS, Android

FRAMEWORKS

VCL, FireMonkey, Lazarus / FPC, .NET

EDITION

Standard / Professional / Enterprise

Features

- Publish/subscribe message pattern to provide one-to-many message distribution and decoupling of applications.
- Acknowledgment of messages sent.
- Implements QoS (Quality of Service) for file delivery.
- Optionally can request Authorization for files received.
- Low memory usage.

Technical specification

Component class	<code>TsgcWSPClient_Files</code> (unit <code>sgcWebSocket_Protocol_Files_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 Files subprotocol.
<code>Broker</code>	Optional broker component that forwards file-protocol traffic through a shared client transport.
<code>Files</code>	File-transfer settings collection covering buffer size, save directory and QoS.
<code>Guid</code>	Identifier used by the server to route file-protocol traffic to this client instance.
<code>Version</code>	Read-only <code>sgcWebSockets</code> library version string.

Main methods

The principal public methods exposed by the component.

<code>Subscribe()</code>	Subscribes this client to a channel so it receives file-protocol notifications published on it.
<code>UnSubscribe()</code>	Cancels a previous <code>Subscribe</code> so the client no longer receives messages from the channel.
<code>SendFile()</code>	Uploads a file to the server in fragmented chunks with configurable QoS.
<code>WriteData()</code>	Sends a plain text message to the server over the Files subprotocol.

Public events

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

OnBinary	Fires when a raw binary frame arrives through the Files subprotocol transport.
OnConnect	Fires after the WebSocket handshake completes and the Files subprotocol is ready to transfer files.
OnDisconnect	Fires when the transport connection drops; the subprotocol stops its QoS timer and optionally clears the pending streams.
OnError	Fires for transport-level errors reported by the underlying sgcWebSocket client.
OnException	Fires when an unhandled exception is raised inside the subprotocol worker or QoS timer thread.
OnFileBeforeSent	Fires just before the file header leaves the client, letting you inspect or tweak the outgoing TsgcWSMessageFile.
OnFileReceived	Fires when an incoming file has been fully received and persisted to disk.
OnFileReceivedAuthorization	TsgcWSPClient_Files > Events > OnFileReceivedAuthorization
OnFileReceivedError	<pre>property OnFileReceivedError: TsgcWSFileErrorEvent; // TsgcWSFileErrorEvent = procedure(Connection: TsgcWSCConnection; const aMessage: TsgcWSMessageFile; const Error: String) of object __property TsgcW...</pre>
OnFileReceivedFragment	Fires every time a new fragment of an incoming file has been written to disk; ideal for progress indicators.
OnFileSent	Fires when the server confirms that a file has been fully received.
OnFileSentAcknowledgment	Fires when the server acknowledges processing of an outbound file fragment under qosLevel1 or qosLevel2.
OnFileSentError	Fires when the upload of a file fails either locally or after a rejection from the server.
OnFileSentFragmentRequest	Fires each time the client is about to send a new outbound file fragment; ideal for upload progress and cancellation.

OnFragmented	Fires for every fragmented WebSocket frame passing through the Files subprotocol transport.
OnMessage	Fires when a plain text message arrives inside a Files subprotocol envelope.
OnRawMessage	Fires with the untouched incoming text before the subprotocol parses it; set Handled to stop default processing.
OnSubscription	Fires when the server confirms a channel subscription requested with Subscribe.
OnUnSubscription	Fires when the server confirms that a channel subscription has been cancelled.

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/Files/Protocol_Files.htm

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

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

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

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

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

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