SGCWEBSOCKETS · TECHNICAL DOCUMENT

WAMP Protocol

Routed RPC and publish-subscribe over WebSocket for Delphi, C++ Builder and .NET — WAMP v1 client and server, both raw JSON and MsgPack serializers.

eSeGeCe

WWW.ESEGECE.COM

Overview

WAMP is an open WebSocket subprotocol that provides two asynchronous messaging patterns: RPC and PubSub.

At a glance

COMPONENT CLASS
`TsgcWSPClient_WAMP`

STANDARDS / SPEC
WAMP specification (latest IETF draft)

TRANSPORTS
TCP, TLS

PLATFORMS
Windows, macOS, Linux, iOS, Android

FRAMEWORKS
VCL, FireMonkey, Lazarus / FPC, .NET

EDITION
Standard / Professional / Enterprise

Features

- Native Delphi implementation with full ANSI/Unicode support.

Technical specification

Standards & specs	WAMP specification (latest IETF draft) · WAMP specifications index
Component class	`TsgcWSPClient_WAMP` (unit `sgcWebSocket_Protocol_WAMP_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`	References the TsgcWebSocketClient that carries WAMP v1 frames over a WebSocket connection.
`Broker`	References a raw-TCP broker component so the WAMP subprotocol travels over a plain socket instead of a WebSocket.
`Version`	Read-only string with the sgcWebSockets build version of the WAMP subprotocol component.

Main methods

The principal public methods exposed by the component.

`CancelCall()`	Requests cancellation of an in-flight RPC previously started with Call.
`Subscribe()`	Registers interest in a PubSub topic so that matching events are delivered to the OnEvent handler.
`UnSubscribe()`	Cancels a prior subscription so that further events on the topic no longer reach this client.
`Publish()`	Broadcasts an event payload to every subscriber of the given topic, with optional exclude/eligible session lists.
`Call()`	Invokes a remote procedure identified by its URI and correlates the eventual result or error to the supplied call id.

`WriteData()`	Sends a pre-built text or binary WAMP frame directly over the underlying WebSocket transport.
`Prefix()`	Registers a short label that expands to a full URI, letting later Call, Subscribe and Publish frames use a compact notation.

Public events

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

`OnBinary`	Fires when the server sends a binary WebSocket frame that is not part of the standard WAMP v1 text protocol.
`OnCallError`	Fires when a remote procedure invoked by Call fails on the server or is rejected.
`OnCallProgressResult`	Fires for each interim chunk of a streaming RPC before the final result arrives via OnCallResult.
`OnCallResult`	Fires once per successful RPC to deliver the final result of a Call invocation.
`OnConnect`	Fires when the underlying WebSocket (or raw TCP) transport has successfully connected to the server.
`OnDisconnect`	Fires when the underlying transport closes, ending the current WAMP session.
`OnError`	Fires when the component detects a transport or protocol-level error condition.
`OnEvent`	Fires when a published event arrives on a topic this client has previously subscribed to.
`OnException`	Fires when a Delphi exception is raised inside one of the component's worker threads or event handlers.
`OnFragmented`	Fires for each fragment of a multi-frame WebSocket message before reassembly.
`OnMessage`	Fires for incoming text frames that the WAMP decoder did not route to a higher-level RPC or PubSub handler.
`OnRawMessage`	Fires before WAMP decoding, giving the application first look at every incoming text frame with an option to suppress further processing.

`OnWelcome`	Fires when the server's WELCOME frame has been received, signalling that the WAMP session is fully open and ready for RPC and PubSub calls.

Quick Start

Drop the component on a form, configure the properties below and activate it. The snippet that follows shows the typical **WAMP | Publishers** configuration sourced from the online help.

> **About this scenario.** A publisher sends (publishes) an event by providing a topic (aka channel) as the abstract address, not a specific peer. Just call Publish method and pass as arguments the name of the topic and the message you want to send. This message will be delivered to all subscribers of this topic. As a note, there is no need to subscribe to a topic to publish messages on that topic.

Delphi (VCL / FireMonkey)

```
oClient := TsgcWebSocketClient.Create(nil);
oClient.Host := '127.0.0.1';
oClient.Port := 80;
oClientWAMP := TsgcWSPClient_WAMP.Create(nil);
oClientWAMP.Client := oClient.
oClientWAMP.OnMessage := OnMessageEvent;
oClient.Active := True;

// Publish a message to all subscribers
oClient.Publish('myTopic', 'Hello subscribers myTopic');
```

C++ Builder

```
oClient = new TsgcWebSocketClient();
oClient->Host = "127.0.0.1";
oClient->Port = 80;
oClientWAMP = new TsgcWSPClient_WAMP();
oClientWAMP->Client = oClient.
oClientWAMP->OnMessage = OnMessageEvent;
oClient->Active = true;

// Publish a message to all subscribers
oClient->Publish("myTopic", "Hello subscribers myTopic");
```

.NET (C#)

```
oClient = new TsgcWebSocketClient();
oClient.Host = "127.0.0.1";
oClient.Port = 80;
oClientWAMP = new TsgcWSPClient_WAMP();
oClientWAMP.Client = oClient.
oClientWAMP.OnMessage = OnMessageEvent;
oClient.Active = true;

// Publish a message to all subscribers
oClient.Publish("myTopic", "Hello subscribers myTopic");
```

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 · WAMP | Subscribers

A subscriber receives events by first providing topics (aka channels) it is interested in. Subsequently, the subscriber will receive any events published to that topic.

Delphi (VCL / FireMonkey)

```
procedure OnMessageEvent(Connection: TsgcWSConnection; const Text: string);
begin
ShowMessage(Text);
end;

oClient := TsgcWebSocketClient.Create(nil);
oClient.Host := '127.0.0.1';
oClient.Port := 80;
oClientWAMP := TsgcWSPClient_WAMP.Create(nil);
oClientWAMP.Client := oClient.
oClientWAMP.OnMessage := OnMessageEvent;
oClient.Active := True;

// Subscribe to topic after successful connect
oClient.Subscribe('myTopic');
```

C++ Builder

```
void OnMessageEvent(TsgcWSConnection *Connection, string Text)
{
ShowMessage(Text);
}

oClient = new TsgcWebSocketClient();
oClient→Host = "127.0.0.1";
oClient→Port = 80;
oClientWAMP = new TsgcWSPClient_WAMP();
oClientWAMP→Client = oClient.
oClientWAMP→OnMessage = OnMessageEvent;
oClient→Active = true;

// Subscribe to topic after successful connect
oClient→Subscribe("myTopic");
```

.NET (C#)

```
void OnMessageEvent(TsgcWSConnection Connection, string Text)
{
MessageBox.Show(Text);
}

oClient = new TsgcWebSocketClient();
oClient.Host = "127.0.0.1";
oClient.Port = 80;
oClientWAMP = new TsgcWSPClient_WAMP();
oClientWAMP.Client = oClient.
oClientWAMP.OnMessage = OnMessageEvent;
oClient.Active = true;

// Subscribe to topic after successful connect
oClient.Subscribe("myTopic");
```

2 · WAMP | RPC Progress Results

Sometimes, Remote Procedure Calls require more than one result to finish requests, by default WAMP 1.0 protocol doesn't allow Partial results in a call, this is a feature only for sgcWebSockets library.

Delphi (VCL / FireMonkey)

```
procedure OnServerCall(Connection: TsgcWSConnection; const CallId, ProcUri, Arguments: string);
var
vNum: Integer;
begin
if ProcUri = 'GetProgressiveTime' then
begin
vNum := StrToInt(Arguments);
for i := 1 to vNum do
        begin
         if i = 20 then
oServerWAMP.CallResult(CallId, FormatDateTime('yyyymmdd hh:nn:ss', Now))
         else
         oServerWAMP.CallProgressiveResult(CallId, FormatDateTime('yyyymmdd hh:nn:ss', Now));
        end
end
else
oServer.WAMP.CallError(CallId, 'Unknown method');
end;</code><code class="delphi">
oServer := TsgcWebSocketServer.Create(nil);
oServer.Port := 80;
oServerWAMP := TsgcWSPServer_WAMP.Create(nil);
oServerWAMP.OnCall := OnServerCallEvent;
oServerWAMP.Server := oServer;
oServer.Active := True;
```

C++ Builder

```
void OnServerCall(TsgcWSConnection *Connection, const string CallId, const string ProcUri, const
{
if (ProcUri == "GetProgressiveTime")
{
int vNum = StrToInt(Arguments);
        for (int i = 1; i = vNum; i++)
        {
         if (i == 20)
         {
oServerWAMP->CallResult(CallId, FormatDateTime("yyyymmdd hh:nn:ss", Now));
         }
         else
         {
                oServerWAMP->CallProgressiveResult(CallId, FormatDateTime("yyyymmdd hh:nn:ss", N
         }
        }
}
else
{
oServer->WAMP->CallError(CallId, "Unknown method");
}
}</code><code class="cpp">
oServer = new TsgcWebSocketServer();
oServer->Port = 80;
oServerWAMP = new TsgcWSPServer_WAMP();
oServerWAMP->OnCall = OnServerCallEvent();
oServerWAMP->Server = oServer;
oServer->Active = true;
```

.NET (C#)

```
void OnServerCall(TsgcWSConnection Connection, string CallId, string ProcUri, string Arguments)
{
if (ProcUri == "GetProgressiveTime")
{
int vNum = Int32.Parse(Arguments);
        for (int i = 1; i = vNum; i++)
        {
         if (i == 20)
         {
oServerWAMP.CallResult(CallId, FormatDateTime("yyyymmdd hh:nn:ss", Now));
         }
         else
         {
                oServerWAMP.CallProgressiveResult(CallId, FormatDateTime("yyyymmdd hh:nn:ss", No
         }
        }
}
else
{
oServer.WAMP.CallError(CallId, "Unknown method");
}
}</code><code class="csharp">
oServer = new TsgcWebSocketServer();
oServer.Port = 80;
oServerWAMP = new TsgcWSPServer_WAMP();
oServerWAMP.OnCall = OnServerCallEvent();
oServerWAMP.Server = oServer;
oServer.Active = true;
```

3 · WAMP | Simple RPC

The most common use of the WAMP component is for a client to request a method from the server, and the server sends a response to the client. The client can send only the name of the method and/or can pass some parameters required by server to calculate the result. Server processes requests and if successful sends a response to client with the result. If there is any error, server sends an error response to client.

```
Delphi (VCL / FireMonkey)
```

```
procedure OnServerCall(Connection: TsgcWSConnection; const CallId, ProcUri, Arguments: string);
begin
if ProcUri = 'GetTime' then
oServerWAMP.CallResult(CallId, FormatDateTime('yyyymmdd hh:nn:ss', Now))
else
oServer.WAMP.CallError(CallId, 'Unknown method');
end;
oServer := TsgcWebSocketServer.Create(nil);
oServer.Port := 80;
oServerWAMP := TsgcWSPServer_WAMP.Create(nil);
oServerWAMP.OnCall := OnServerCallEvent;
oServerWAMP.Server := oServer;
oServer.Active := True;
```

C++ Builder

```
void OnServerCall(TsgcWSConnection *Connection, const string CallId, const string ProcUri, const
{
if (ProcUri == "GetTime")
{
oServerWAMP->CallResult(CallId, FormatDateTime("yyyymmdd hh:nn:ss", Now));
}
else
{
oServer->WAMP->CallError(CallId, "Unknown method");
}
}
oServer = new TsgcWebSocketServer();
oServer->Port = 80;
oServerWAMP = new TsgcWSPServer_WAMP();
oServerWAMP->OnCall = OnServerCallEvent();
oServerWAMP->Server = oServer;
oServer->Active = true;
```

.NET (C#)

```
void OnServerCall(TsgcWSConnection Connection, string CallId, string ProcUri, string Arguments)
{
if (ProcUri == "GetTime")
{
oServerWAMP.CallResult(CallId, DateTime.Now.ToString("yyyyMMdd HH:mm:ss"));
}
else
{
oServer.WAMP.CallError(CallId, "Unknown method");
}
}
oServer = new TsgcWebSocketServer();
oServer.Port = 80;
oServerWAMP = new TsgcWSPServer_WAMP();
oServerWAMP.OnCall = OnServerCallEvent();
oServerWAMP.Server = oServer;
oServer.Active = true;
```

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.

Primary standard / spec — WAMP specification (latest IETF draft)	wamp-proto.org/wamp_latest_ietf.html
Primary standard / spec — WAMP specifications index	wamp-proto.org/spec
Online help — component page	www.esegece.com/help/sgcWebSockets/Components/Protocols/Subprotocols/WAMP/Protocol_WAMP.htm
Online help — class reference (TsgcWSPClient_WAMP)	www.esegece.com/help/sgcWebSockets/Components/Protocols/Subprotocols/WAMP/TsgcWSPClient_WAMP.htm
Delphi demo project (in the sgcWebSockets package)	`Demos\02.WebSocket_Protocols\04.WAMP_Protocol`
.NET demo project (in the sgcWebSockets package)	`.net\demos\02.WebSocket_Protocols\04.WAMP_Protocol`
Component page	www.esegece.com/products/websockets/protocols/wamp/
Product page	www.esegece.com/products/websockets/

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