SGCWEBSOCKETS · TECHNICAL DOCUMENT

TURN Client

TsgcTURNClient — RFC 8656 Traversal Using Relays around NAT client; allocates a relayed address and relays traffic when direct paths fail.

Overview

TsgcTURNClient is the client that implements the TURN protocol and allows you to send allocation requests to TURN servers. The client inherits from STUN Client, so all methods supported by STUN client are already supported by TURN Client.

At a glance

COMPONENT CLASS `TsgcTURNClient`	STANDARDS / SPEC **TURN — RFC 8656**
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

Standards & specs	TURN — RFC 8656
Component class	`TsgcTURNClient` (unit `sgcTURN_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.

`Host`	IP address or DNS name of the TURN server to which the client sends Allocate and related requests.
`Port`	Listening port of the TURN server; defaults to 3478 as defined by RFC 5766.
`Transport`	Transport used to connect to the TURN server: UDP, TCP or TLS. Default is UDP.
`STUNOptions`	Inherited STUN options: FINGERPRINT, SOFTWARE attribute and Authentication credentials.
`TURNOptions`	TURN-specific options: Allocation lifetime, long-term credentials, AutoRefresh and data-channel behaviour.
`RetransmissionOptions`	UDP retransmission settings (RTO and MaxRetries) used to resend TURN requests when no response arrives.
`LogFile`	Saves every STUN/TURN message sent or received to a file for debugging.
`NotifyEvents`	Selects how threaded events are synchronized with the main VCL/FMX thread.
`IPVersion`	Address family used to resolve the TURN server; defaults to IPv4.
`Version`	Read-only build version of the sgcWebSockets component library.

Main methods

The principal public methods exposed by the component.

`SendIndication()`	Sends a datagram to a peer wrapped in a SEND Indication (requires an active permission).
`SendChannelData()`	Sends a datagram to a peer using a bound channel, producing a smaller ChannelData message.
`SendRequest()`	Inherited generic STUN request used to send a Binding Request over the TURN transport.
`WriteData()`	Writes raw bytes directly on the underlying TURN transport; used for custom payloads and diagnostics.
`Refresh()`	Sends a REFRESH request to extend (or end) the current allocation lifetime.
`CreatePermission()`	Installs a 5-minute permission that allows a specific peer IP to reach the relayed transport address.
`Allocate()`	Sends an ALLOCATE request to the TURN server to reserve a relayed transport address.
`ChannelBind()`	Binds a 16-bit channel number (0x4000-0x7FFF) to a peer for low-overhead ChannelData exchange.
`Clear()`	Resets the client state, releasing cached allocation, permissions and channel bindings.

Public events

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

`OnSTUNBeforeSend`	Fires just before a STUN/TURN message is sent to the server, allowing last-minute inspection or modification.
`OnSTUNException`	Fires when a transport, parsing or timeout exception occurs while processing a STUN/TURN message.
`OnSTUNResponseError`	Fires when the server returns an error response (for example 401 Unauthorized or 437 Allocation Mismatch).
`OnSTUNResponseSuccess`	Fires when the server returns a success response to any STUN or TURN request.

`OnTURNAllocate`	Fires after a successful ALLOCATE response; exposes the relayed IP address and port assigned by the server.
`OnTURNChannelBind`	Fires after a successful ChannelBind response; exposes the channel number assigned to the peer.
`OnTURNChannelData`	Fires when the client receives a ChannelData message relayed from a peer bound to a channel.
`OnTURNCreatePermission`	Fires after a successful CreatePermission response for the requested peer IP.
`OnTURNDataIndication`	Fires when the client receives a DATA Indication relayed from a peer that has no bound channel.
`OnTURNICMPIndication`	Fires when the server forwards an ICMP error received from a peer (for example unreachable or TTL exceeded).
`OnTURNRefresh`	Fires after a successful REFRESH response, reporting the new allocation lifetime.

Quick Start

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

> **About this scenario.** Usually TURN servers run on UDP port 3478 and don't require authentication, so in order to send a TURN request, fill the server properties to allow the client know where connect and Handle the events where the component will receive the response from server.

Delphi (VCL / FireMonkey)

```
oTURN := TsgcTURNClient.Create(nil);
    oTURN.Host := 'turn.sgcwebsockets.com';
    oTURN.Port := 3478;
    oTURN.Allocate;

    procedure OnTURNAllocate(Sender: TObject; const aSocket: TsgcSocketConnection;
     const aMessage: TsgcSTUN_Message; const aAllocation: TsgcTURN_ResponseAllocation);
    begin
    DoLog('Relayed IP: ' + aAllocation.RelayedIP + '. Relayed Port: ' + IntToStr(aAllocation.Rel
    end;

    procedure OnSTUNResponseError(Sender: TObject; const aMessage: TsgcSTUN_Message;
    const aError: TsgcSTUN_ResponseError);
    begin
    DoLog('Error: ' + IntToStr(aError.Code) + ' ' + aError.Reason);
    end;
```

C++ Builder

```
TsgcTURNClient oTURN = new TsgcTURNClient(this);
    oTURN→Host = "turn.sgcwebsockets.com";
    oTURN→Port = 3478;
    oTURN→Allocate();

    private void OnTURNAllocate(TObject *Sender, const TsgcSocketConnection *aSocket,
     const TsgcSTUN_Message *aMessage, const TsgcTURN_ResponseAllocation *aAllocation)
    {
    DoLog("Relayed IP: " + aAllocation→RelayedIP + ". Relayed Port: " + IntToStr(aAllocation→R
    }

    private void OnSTUNResponseError(TObject *Sender, const TsgcSTUN_Message *aMessage,
    const TsgcSTUN_ResponseError *aError)
    {
    DoLog("Error: " + IntToStr(aError→Code) + " " + aError→Reason);
    }
```

.NET (C#)

```
TsgcTURNClient oTURN = new TsgcTURNClient();
    oTURN.Host = "turn.sgcwebsockets.com";
    oTURN.Port = 3478;
    oTURN.Allocate();

    private void OnTURNAllocate(Component Sender, const TsgcSocketConnection aSocket,
      const TsgcSTUN_Message aMessage, const TsgcTURN_ResponseAllocation aAllocation)
    {
    DoLog("Relayed IP: " + aAllocation.RelayedIP + ". Relayed Port: " + aAllocation.RelayedPort.
    }

    private void OnSTUNResponseError(Component Sender, const TsgcSTUN_Message aMessage,
    const TsgcSTUN_ResponseError aError)
    {
    DoLog("Error: " + aError.Code.ToString() + " " + aError.Reason);
    }
```

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 · TURN Client | Send Indication

TURN Protocol supports 2 mechanisms for sending and receiving data from peers, one of them is Send and Data mechanisms.

Delphi (VCL / FireMonkey)

```
oTURN.SendIndication('80.147.23.157', 5000, 'random data');

procedure OnTURNDataIndication(Sender: TObject; const aSocket: TsgcSocketConnection;
 const aMessage: TsgcSTUN_Message; const aDataIndication: TsgcTURN_ResponseDataIndication);
begin
DoLog('#Data Indication: [' + aDataIndication.PeerIP + ':' + IntToStr(aDataIndication.PeerPort)
   sgcGetStringFromBytes(aDataIndication.Data));
end;
```

C++ Builder

```
oTURN->SendIndication("80.147.23.157", 5000, "random data");

void OnTURNDataIndication(TObject *Sender, const TsgcSocketConnection *aSocket,
 const TsgcSTUN_Message *aMessage, const TsgcTURN_ResponseDataIndication *aDataIndication)
{
DoLog("#Data Indication: [" + aDataIndication->PeerIP + ":" + IntToStr(aDataIndication->PeerPort
   sgcGetStringFromBytes(aDataIndication->Data));
}
```

.NET (C#)

```
oTURN.SendIndication("80.147.23.157", 5000, "random data");


void OnTURNDataIndication(Component Sender, TsgcSocketConnection aSocket,
 TsgcSTUN_Message aMessage, TsgcTURN_ResponseDataIndication aDataIndication)
{
DoLog("#Data Indication: [" + aDataIndication.PeerIP + ":" + IntToStr(aDataIndication.PeerPort) +
    aDataIndication.Data.ToString());
}
```

2 · TURN Client | Allocate IP Address

TURN Protocol allows you to use a Relayed IP Address to exchange data between peers that are behind NATs.

Delphi (VCL / FireMonkey)

```
oTURN := TsgcTURNClient.Create(nil);
oTURN.Host := 'turn.sgcwebsockets.com';
oTURN.Port := 3478;
oTURN.Allocate();

procedure OnTURNAllocate(Sender: TObject; const aSocket: TsgcSocketConnection; const
aMessage: TsgcSTUN_Message; const aAllocation: TsgcTURN_ResponseAllocation);
begin
DoLog('Relayed IP: ' + aAllocation.RelayedIP + '. Relayed Port: ' + IntToStr(aAllocation.Relayed
end;


procedure OnSTUNResponseError(Sender: TObject; const aMessage: TsgcSTUN_Message;
const aError: TsgcSTUN_ResponseError);
begin
DoLog('Error: ' + IntToStr(aError.Code) + ' ' + aError.Reason);
end;
```

C++ Builder

```
TsgcTURNClient oTURN = new TsgcTURNClient(this);
oTURN->Host = "turn.sgcwebsockets.com";
oTURN->Port = 3478;
oTURN->Allocate();


private void OnTURNAllocate(TObject *Sender, const TsgcSocketConnection *aSocket,
const TsgcSTUN_Message *aMessage, const TsgcTURN_ResponseAllocation *aAllocation)
{
DoLog("Relayed IP: " + aAllocation->RelayedIP + ". Relayed Port: " +
   IntToStr(aAllocation->RelayedPort));
}


private void OnSTUNResponseError(TObject *Sender, const TsgcSTUN_Message *aMessage,
const TsgcSTUN_ResponseError *aError)
{
DoLog("Error: " + IntToStr(aError->Code) + " " + aError->Reason);
}
```

.NET (C#)

```
TsgcTURNClient oTURN = new TsgcTURNClient();
oTURN.Host = "turn.sgcwebsockets.com";
oTURN.Port = 3478;
oTURN.Allocate();


private void OnTURNAllocate(Component Sender, TsgcSocketConnection aSocket, TsgcSTUN_Message aMe
 const TsgcTURN_ResponseAllocation aAllocation)
{
DoLog("Relayed IP: " + aAllocation.RelayedIP + ". Relayed Port: " +
    aAllocation.RelayedPort.ToString());
}

private void OnSTUNResponseError(Component Sender, TsgcSTUN_Message aMessage,
 TsgcSTUN_ResponseError aError)
{
DoLog("Error: " + aError.Code.ToString() + " " + aError.Reason);
}
```

3 · TURN Client | Channels

Channels provide a way for the TURN Client and Server to send application data using ChannelData messages, which have less overhead than Send and Data Indications.

Delphi (VCL / FireMonkey)

```
oTURN.ChannelBind('80.147.23.157', 5000);

procedure OnTURNChannelBind(Sender: TObject; const aSocket: TsgcSocketConnection;
 const aMessage: TsgcSTUN_Message; const aChannelBind: TsgcTURN_ResponseChannelBind);
begin
DoLog('#Channel Bind: ' + IntToStr(aChannelBind.Channel));
end;
```

```
C++ Builder
```

```
oTURN->ChannelBind("80.147.23.157", 5000);

void OnTURNChannelBind(TObject *Sender, const TsgcSocketConnection *aSocket,
 const TsgcSTUN_Message *aMessage, const TsgcTURN_ResponseChannelBind *aChannelBind)
{
DoLog("#Channel Bind: " + IntToStr(aChannelBind->Channel));
}
```

```
.NET (C#)
```

```
oTURN.ChannelBind("80.147.23.157", 5000);


void OnTURNChannelBind(Component Sender, TsgcSocketConnection aSocket,
 TsgcSTUN_Message aMessage, TsgcTURN_ResponseChannelBind aChannelBind)
{
DoLog("#Channel Bind: " + aChannelBind.Channel.ToString());
}
```

4 · TURN Client | Create Permissions

When a new Allocation is created in a TURN server, this allocation cannot process any incoming packet from other peers if has no permissions. So, in order to allow other peers to communicate using a Relayed IP Address, first the TURN Client must create permissions for the IP Addresses that are allowed to exchange Data.

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

```
oTURN.CreatePermission('80.147.23.157');

procedure OnTURNCreatePermission(Sender: TObject; const aSocket: TsgcSocketConnection;
 const aMessage: TsgcSTUN_Message; const aCreatePermission: TsgcTURN_ResponseCreatePermission);
begin
DoLog('#Create Permission: ' + aCreatePermission.IPAddresses.Text);
end;
```

C++ Builder

```
oTURN->CreatePermission("80.147.23.157");


void OnTURNCreatePermission(TObject *Sender, const TsgcSocketConnection *aSocket,
 const TsgcSTUN_Message *aMessage, const TsgcTURN_ResponseCreatePermission *aCreatePermission)
{
DoLog("#Create Permission: " + aCreatePermission->IPAddresses->Text);
}
```

.NET (C#)

```
oTURN.CreatePermission("80.147.23.157");


void OnTURNCreatePermission(Component Sender, TsgcSocketConnection aSocket,
  TsgcSTUN_Message aMessage, TsgcTURN_ResponseCreatePermission aCreatePermission)
{
DoLog("#Create Permission: " + aCreatePermission.IPAddresses);
}
```

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 — TURN — RFC 8656	datatracker.ietf.org/doc/html/rfc8656
Online help — component page	www.esegece.com/help/sgcWebSockets/Components/P2P/TURN/Client/TsgcTURNClient.htm
Delphi demo project (in the sgcWebSockets package)	`Demos\35.P2P\03.TURN`
Component page	www.esegece.com/products/websockets/p2p/turn-client/
Product page	www.esegece.com/products/websockets/

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