SGCWEBSOCKETS · TECHNICAL DOCUMENT

STUN Client

TsgcSTUNClient — RFC 8489 Session Traversal Utilities for NAT client; binding requests for peer-public-address discovery.

eSeGeCe

WWW.ESEGECE.COM

Overview

TsgcSTUNClient is the client that implements the STUN protocol and allows you to send binding requests to STUN servers.

At a glance

COMPONENT CLASS
`TsgcSTUNClient`

STANDARDS / SPEC
STUN — RFC 8489

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	STUN — RFC 8489
Component class	`TsgcSTUNClient` (unit `sgcSTUN_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 STUN server to which the client sends Binding Requests.
`Port`	Listening port of the STUN server; defaults to 3478 as defined by RFC 5389.
`Transport`	Transport used to connect to the STUN server: UDP, TCP or TLS. Default is UDP.
`STUNOptions`	STUN-specific options: FINGERPRINT, SOFTWARE attribute and Authentication credentials.
`RetransmissionOptions`	UDP retransmission settings (RTO and MaxRetries) used to resend Binding Requests when no response arrives.
`LogFile`	Saves every STUN 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 STUN server; defaults to IPv4.
`Version`	Read-only build version of the sgcWebSockets component library.

Main methods

The principal public methods exposed by the component.

`SendRequest()`	Sends a STUN Binding Request to the configured server and returns True if the request was dispatched.
`WriteData()`	Advanced: sends raw bytes over the STUN client socket, either to the connected peer or to an explicit IP/port pair.
`Clear()`	Clears the client internal state, cancelling any in-flight STUN transactions and freeing buffered messages.

Public events

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

`OnSTUNBeforeSend`	Fires just before a STUN message is sent to the server, allowing last-minute inspection or modification of the outgoing message.
`OnSTUNException`	Fires when an unhandled exception is raised while processing STUN protocol messages.
`OnSTUNResponseError`	Fires when the STUN server returns an error response; exposes the numeric Code and human-readable Reason.
`OnSTUNResponseSuccess`	Fires when the STUN server returns a successful Binding Response with the client mapped address.

Quick Start

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

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

Delphi (VCL / FireMonkey)

```
oSTUN := TsgcSTUNClient.Create(nil);
    oSTUN.Host := 'stun.sgcwebsockets.com';
    oSTUN.Port := 3478;
    oSTUN.SendRequest;

    procedure OnSTUNResponseSuccess(Sender: TObject; const aSocket: TsgcSocketConnection;
     const aMessage: TsgcSTUN_Message; const aBinding: TsgcSTUN_ResponseBinding);
    begin
    DoLog('Remote IP: ' + aBinding.RemoteIP + '. Remote Port: ' + IntToStr(aBinding.RemotePort))
    end;

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

C++ Builder

```
TsgcSTUNClient oSTUN = new TsgcSTUNClient(this);
    oSTUN→Host = "stun.sgcwebsockets.com";
    oSTUN→Port = 3478;
    oSTUN→SendRequest();

    private void OnSTUNResponseSuccess(TObject *Sender, const TsgcSocketConnection *aSocket,
     const TsgcSTUN_Message *aMessage, const TsgcSTUN_ResponseBinding *aBinding)
    {
    DoLog("Remote IP: " + aBinding→RemoteIP + ". Remote Port: " + IntToStr(aBinding→RemotePort
    }

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

.NET (C#)

```
TsgcSTUNClient oSTUN = new TsgcSTUNClient();
    oSTUN.Host = "stun.sgcwebsockets.com";
    oSTUN.Port = 3478;
    oSTUN.SendRequest();

    private void OnSTUNResponseSuccess(Component Sender, TsgcSocketConnection aSocket,
     TsgcSTUN_Message aMessage, TsgcSTUN_ResponseBinding aBinding)
    {
    DoLog("Remote IP: " + aBinding.RemoteIP + ". Remote Port: " + IntToStr(aBinding.RemotePort))
    }

    private void OnSTUNResponseError(Component Sender, TsgcSocketConnection aSocket,
      TsgcSTUN_Message aMessage, TsgcSTUN_ResponseError aError)
    {
    DoLog("Error: " + Int32.Parse(aError.Code) + " " + 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 · STUN Client | Attributes

Every time a server sends a message to client, as a response message to a request binding, the STUN message contains a list of attributes with information about the response.

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

```
procedure OnSTUNResponseSuccess(Sender: TObject; const aSocket: TsgcSocketConnection;
 const aMessage: TsgcSTUN_Message; const aBinding: TsgcSTUN_ResponseBinding);
var
i: Integer;
begin
DoLog('#binding: ' + aBinding.RemoteIP + ':' + IntToStr(aBinding.RemotePort));
for i := 0 to aMessage.Attributes.Count - 1 do
begin
case TsgcSTUN_Attribute(aMessage.Attributes.Items[i]).AttributeType of
stmaFingerprint:
DoLog('#fingerprint: ' + IntToStr(TsgcSTUN_Attribute_FINGERPRINT
(aMessage.Attributes.Items[i]).Fingerprint));
stmaSoftware:
DoLog('#software: ' + TsgcSTUN_Attribute_SOFTWARE
(aMessage.Attributes.Items[i]).Software);
stmaResponse_Origin:
DoLog('#response_origin: ' + TsgcSTUN_Attribute_RESPONSE_ORIGIN
(aMessage.Attributes.Items[i]).Address + ':' +
IntToStr(TsgcSTUN_Attribute_RESPONSE_ORIGIN(aMessage.Attributes.Items
[i]).Port));
stmaOther_Address:
DoLog('#other_address: ' + TsgcSTUN_Attribute_OTHER_ADDRESS
(aMessage.Attributes.Items[i]).Address + ':' +
IntToStr(TsgcSTUN_Attribute_OTHER_ADDRESS(aMessage.Attributes.Items
[i]).Port));
stmaSource_Address:
DoLog('#source_address: ' + TsgcSTUN_Attribute_SOURCE_ADDRESS
(aMessage.Attributes.Items[i]).Address + ':' +
IntToStr(TsgcSTUN_Attribute_SOURCE_ADDRESS(aMessage.Attributes.Items
[i]).Port));
stmaChanged_Address:
DoLog('#changed_address: ' + TsgcSTUN_Attribute_CHANGED_ADDRESS
(aMessage.Attributes.Items[i]).Address + ':' +
IntToStr(TsgcSTUN_Attribute_CHANGED_ADDRESS(aMessage.Attributes.Items
[i]).Port));
end;
end;
end;
```

C++ Builder

```
private void OnSTUNResponseSuccess(TObject * Sender, const TsgcSocketConnection *aSocket,
  const TsgcSTUN_Message *aMessage, const TsgcSTUN_ResponseBinding *aBinding)
{
DoLog("#binding: " + aBinding→RemoteIP + ":" + IntToStr(aBinding→RemotePort));

for (int i = 0; i < aMessage→Attributes→Count; i++)
{
switch (TsgcSTUN_Attribute(aMessage→Attributes→Items[i])→AttributeType)
stmaFingerprint:
DoLog("#fingerprint: " + IntToStr(dynamic_cast<TsgcSTUN_Attribute_FINGERPRINT*>
(aMessage→Attributes→Items[i])→Fingerprint));
stmaSoftware:
DoLog("#software: " + dynamic_cast<TsgcSTUN_Attribute_SOFTWARE*>
(aMessage→Attributes→Items[i])→Software);
stmaResponse_Origin:
DoLog("#response_origin: " + dynamic_cast<TsgcSTUN_Attribute_RESPONSE_ORIGIN*>
(aMessage→Attributes→Items[i])→Address + ":" +
IntToStr(dynamic_cast<TsgcSTUN_Attribute_RESPONSE_ORIGIN*>(aMessage→Attributes→Items
[i])→Port));
}
}
}
```

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

```
```

2 · STUN Client | Long Term Credentials

The long-term credential mechanism relies on a long-term credential, in the form of a username and password that are shared between client and server. The credential is considered long-term since it is assumed that it is provisioned for a user and remains in effect until the user is no longer a subscriber of the system or until it is changed.

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

```
oSTUN := TsgcSTUNClient.Create(nil);
oSTUN.Host := 'stun.sgcwebsockets.com';
oSTUN.Port := 3478;
oSTUN.STUNOptions.Authentication.Credentials := stauLongTermCredential;
oSTUN.STUNOptions.Authentication.Username := 'user_name';
oSTUN.STUNOptions.Authentication.Password := 'secret';
oSTUN.SendRequest;
```

```
C++ Builder
```

```
TsgcSTUNClient oSTUN = new TsgcSTUNClient(this);
oSTUN->Host = "stun.sgcwebsockets.com";
oSTUN->Port = 3478;
oSTUN->STUNOptions->Authentication->Credentials = stauLongTermCredential;
oSTUN->STUNOptions->Authentication->Username = "user_name";
oSTUN->STUNOptions->Authentication->Password = "secret";
oSTUN->SendRequest();
```

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

```
TsgcSTUNClient oSTUN = new TsgcSTUNClient();
oSTUN.Host = "stun.sgcwebsockets.com";
oSTUN.Port = 3478;
oSTUN.STUNOptions.Authentication.Credentials = TsgcStunCredentials.stauLongTermCredential;
oSTUN.STUNOptions.Authentication.Username = "user_name";
oSTUN.STUNOptions.Authentication.Password = "secret";
oSTUN.SendRequest();
```

3 · STUN Client | UDP Retransmissions

When running STUN over UDP, it's possible that the STUN message might be dropped by the network. Reliability of STUN request/response transactions is accomplished through retransmissions of the request message by the client application itself.

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

```
oSTUN := TsgcSTUNClient.Create(nil);
oSTUN.Host := 'stun.sgcwebsockets.com';
oSTUN.Port := 3478;
oSTUN.RetransmissionOptions.Enabled := true;
oSTUN.RetransmissionOptions.RTO := 500;
oSTUN.RetransmissionOptions.MaxRetries := 7;
oSTUN.SendRequest;
```

```
C++ Builder
```

```
TsgcSTUNClient oSTUN = new TsgcSTUNClient(this);
oSTUN->Host = "stun.sgcwebsockets.com";
oSTUN->Port = 3478;
oSTUN->RetransmissionOptions->Enabled = true;
oSTUN->RetransmissionOptions->RTO = 500;
oSTUN->RetransmissionOptions->MaxRetries = 7;
oSTUN->SendRequest();
```

.NET (C#)

```
TsgcSTUNClient oSTUN = new TsgcSTUNClient();
oSTUN.Host = "stun.sgcwebsockets.com";
oSTUN.Port = 3478;
oSTUN.RetransmissionOptions.Enabled = true;
oSTUN.RetransmissionOptions.RTO = 500;
oSTUN.RetransmissionOptions.MaxRetries = 7;
oSTUN.SendRequest();
```

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 — STUN — RFC 8489	datatracker.ietf.org/doc/html/rfc8489
Online help — component page	www.esegece.com/help/sgcWebSockets/Components/P2P/STUN/Client/TsgcSTUNClient.htm
Delphi demo project (in the sgcWebSockets package)	`Demos\35.P2P\02.STUN`
Component page	www.esegece.com/products/websockets/p2p/stun-client/
Product page	www.esegece.com/products/websockets/

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