



sgcWebSockets .NET 2026.6

June 2026

Documentazione per .NET

Copyright © 2012-2026 eSeGeCe Software

info@esegece.com

www.esegece.com

Contenuto

Introduzione.....	19
Panoramica.....	21
Edizioni	21
Installazione	22
Guida rapida	23
Panoramica	23
QuickStart WebSockets	25
QuickStart HTTP	27
Flusso di threading.....	29
Build	30
Server ad alte prestazioni	31
OpenSSL	35
OpenSSL Windows	38
OpenSSL OSX	40
Certificati CA propri OpenSSL.....	42
Certificati OpenSSL P12	44
OpenSSL Verify Certificate	45
OpenSSL Load Additional Functions.....	46
Argomenti	47
Eventi WebSocket.....	47
WebSocket Parameters Connection	48
Utilizzo all'interno di una DLL	49
Test del browser Web.....	50
Autenticazione	51
Connessioni sicure	53

HeartBeat	55
WatchDog.....	56
Log.....	57
HTTP.....	58
Broadcast e canali	59
Bindings.....	60
Inviare file di grandi dimensioni tramite POST.....	61
Compressione.....	63
Flash.....	64
Gruppi.....	65
IOCP	67
ALPN	68
Inoltra richieste HTTP	69
Code.....	71
Transazioni.....	73
Connessioni TCP.....	75
SubProtocol	76
Throttle.....	77
Server-Sent Events	78
LoadBalancing	80
File.....	81
Proxy.....	82
Messaggi Frammentati	83
Componenti	84
TsgcWebSocketClient.....	84
Connetti il server WebSocket.....	86
Apertura connessione client.....	87
Client Chiudi Connessione	89
Client mantiene la connessione aperta.....	90
Disconnessioni Abbandonate.....	91

Connettere il server TCP.....	92
Connessioni TIME_WAIT	93
Reindirizzamenti WebSocket	94
Connetti al Server Sicuro.....	95
Certificati OpenSSL	96
Certificates SChannel.....	97
Invio messaggio di testo dal client	98
Il Client Invia un Messaggio Binario	99
Client Invia Messaggio di Testo e Binario.....	100
Ricezione messaggi di testo	101
Ricevi messaggi binari	102
Autenticazione client	103
Eccezioni del client	106
HandShake WebSocket del client.....	107
Protocollo di registrazione client.....	108
Proxy Client.....	109
TsgcTCPClient	110
TsgcTCPClient	111
TsgcWebSocketServer	112
Avvio server.....	114
Server Bindings.....	115
Avvio e Arresto del Server	116
Server Keep Active	117
Server SSL.....	118
Server SSL SChannel	120
Verifica del certificato del server	123
Mantenimento delle connessioni attive dal server.....	124
Server Plain TCP	125
Chiusura connessione da parte del server	126
Client Connections	127
Autenticazione Server.....	128

Server — Invio di un messaggio di testo	130
Invia messaggio binario dal server	131
Ricezione di un messaggio di testo dal server.....	132
Il server riceve un messaggio binario	133
Server Read Headers from Client.....	134
TsgcWebSocketHTTPServer	135
Richieste al server HTTP	137
HTTP Dispatch Files.....	138
Server HTTP/2.....	139
HTTP/2 Server Push	140
HTTP/2 Alternate Service.....	142
Thread del Server HTTP/2	143
Errore HTTP 404 senza corpo della risposta.....	145
HTTP Server Sessions	146
HTTP Server Stream Video	148
Server SSL SChannel	149
TsgcWebSocketServer_HTTPAPI	152
HTTPAPI URL Reservation	154
HTTPAPI FineTune	156
Server HTTPAPI SSL.....	158
Certificati autofirmati.....	159
HTTPAPI Disabilita HTTP/2	160
Intestazioni personalizzate HTTPAPI	161
HTTPAPI Invia risposta di testo.....	162
HTTPAPI Invia Risposta File	163
HTTPAPI OnDisconnect non viene generato	165
TsgcWebSocketClient_WinHTTP	166
TsgcWebSocketFirewall	167
Firewall: Blacklist e Whitelist.....	174
Firewall: Protezione contro la forza bruta.....	176
Firewall: Rilevamento SQL Injection e XSS	178

Firewall: Limitazione della velocità e protezione contro il flooding	181
TsgcWSRateLimiter	184
TsgcWSCircuitBreaker.....	196
TsgcWSAPIKeyManager	207
TsgcWebSocketLoadBalancerServer.....	220
TsgcWebSocketProxyServer.....	221
TsgcIWWebSocketClient	222
TsgcWSConnection.....	224
Protocolli	226
Protocolli Javascript.....	228
Protocollo MQTT	231
TsgcWSPClient_MQTT	233
Connessione Client MQTT.....	234
Connessione ai server MQTT Mosquitto	235
Client MQTT Sessions	236
Versione MQTT del Client.....	237
MQTT Publish Subscribe	238
MQTT Topics	239
MQTT Subscribe	241
Messaggio di pubblicazione MQTT	242
MQTT Ricevi messaggi	243
MQTT - Pubblicazione e attesa risposta	244
MQTT Cancella Messaggi Conservati.....	245
Protocol AMQP	246
TsgcWSPClient_AMQP.....	247
Client AMQP Connect	250
Disconnessione client AMQP	251
Canali AMQP	252
AMQP Exchange	254
Code AMQP.....	256
Pubblicazione Messaggi AMQP	259

AMQP Consuma Messaggi	260
Recupero Messaggi AMQP	262
AMQP QoS.....	263
AMQP Transactions.....	264
Protocol AMQP1	266
Client AMQP1 Connect	268
Disconnessione client AMQP1	269
Connessione Client AMQP1 con Timeout di Inattività	270
Client AMQP1 Connection State	271
Autenticazione Client AMQP1	272
Client AMQP1 Azure MessageBus.....	273
AMQP1 Sessions.....	276
AMQP1 Links.....	278
Link Sender AMQP1	279
AMQP1 Receiver Links	282
AMQP1 Invio Messaggio.....	284
AMQP1 Lettura messaggio	286
Protocollo STOMP	287
TsgcWSPClient_STOMP	288
TsgcWSPClient_STOMP_RabbitMQ.....	289
TsgcWSPClient_STOMP_ActiveMQ	290
Protocol AppRTC	291
TsgcWSPServer_AppRTC.....	292
Protocollo WebRTC	293
TsgcWSPServer_WebRTC.....	294
Protocollo WebRTC Javascript.....	295
Protocollo WAMP	296
TsgcWSPServer_WAMP	298
TsgcWSPClient_WAMP	299
Protocollo WAMP Javascript.....	300
Sottoscrittori	303

Publisher	304
RPC semplice	305
Risultati avanzamento RPC	306
Protocollo WAMP2	308
TsgcWSPClient_WAMP2	309
Protocollo predefinito.....	312
TsgcWSPServer_sgc.....	314
TsgcWSPClient_sgc	315
TsgcIWWSPClient_sgc	316
Protocol Default Javascript.....	317
TsgcWSPServer_Dataset	321
TsgcWSPClient_Dataset	322
TsgcIWWSPClient_Dataset.....	323
File di protocollo	324
TsgcWSPServer_Files.....	325
TsgcWSPClient_Files	326
TsgcWSMessageFile	327
Come inviare file al server.....	328
Come inviare file ai client	329
Come inviare file di grandi dimensioni.....	330
Protocol Presence	331
TsgcWSPServer_Presence	332
TsgcWSPPresenceMessage	333
TsgcWSPClient_Presence.....	334
Protocollo Presence Javascript	335
Protocollo E2EE.....	338
TsgcWSPServer_E2EE	340
TsgcWSPClient_E2EE	341
API	342
API Binance	344
Binance Connect WebSocket API	352

Binance Subscribe WebSocket Channel	353
Binance Ottieni dati di mercato.....	354
Binance Private REST API.....	355
Binance Trade Spot.....	356
Orario delle richieste private Binance	358
Binance Withdraw	359
API Binance Futures	360
API Binance Futures Trade	367
API SocketIO.....	368
API Coinbase.....	370
Coinbase Connetti WebSocket API.....	375
Coinbase Iscriviti al canale WebSocket.....	376
Coinbase Ottieni dati di mercato	377
Coinbase API REST privata	378
Coinbase Tempo delle richieste private	379
Coinbase Inserisci ordini	380
Coinbase Account Sandbox	381
API SignalRCore	382
API SignalR	388
API Kraken.....	391
API Kraken WebSockets Public	393
API Kraken WebSockets Private.....	398
API Kraken REST Public.....	401
API Kraken REST Private	403
API Kraken Futures.....	406
API Kraken Futures WebSockets Public.....	408
API Kraken Futures WebSockets Private	415
API Kraken Futures REST Public	421
API Kraken Futures REST Private	423
API Pusher	428
API Bitmex.....	436

Bitmex Connect WebSocket API	440
Bitmex Sottoscrivi Canale WebSocket	441
Come effettuare un ordine su Bitmex.....	442
API Bitfinex.....	445
API Kucoin	449
Kucoin Connect WebSocket API	456
Kucoin Subscribe WebSocket Channel.....	457
Kucoin Get Market Data	458
Kucoin Private REST API.....	459
Kucoin Trade Spot.....	460
Tempo richieste private Kucoin.....	462
API Kucoin Futures	463
Kucoin Futures Connessione WebSocket API	469
Kucoin Futures Subscribe WebSocket Channel.....	470
Kucoin Futures Recupero dei dati di mercato	471
Kucoin Futures REST API Privata	472
Kucoin Futures Trade.....	473
Kucoin Futures Private Requests Time.....	476
API 3Commas	477
API OKX.....	482
API XTb	488
API Bybit	492
API Forex	497
Forex Connect REST API	499
Ordini di trading Forex	500
Forex Account Queries	501
Dati di mercato Forex	502
Forex Connect WebSocket API	503
Forex Sottoscrizione Stream.....	504
API Cex.....	506
API Cex Plus	514

API Discord.....	519
API OpenAI	522
API MEXC	524
API MEXC Futures.....	528
API Bitget.....	532
API GateIO	534
API Deribit	536
API Crypto.com.....	539
API HTX	541
API Whatsapp	543
Creazione App WhatsApp	547
WhatsApp Phone Number Id.....	549
Token WhatsApp	550
WhatsApp Webhook	551
Sicurezza WhatsApp.....	552
WhatsApp Invia Messaggi	553
WhatsApp - Invio di messaggi interattivi	556
WhatsApp Invio Messaggi Template.....	561
WhatsApp Ricezione Messaggi e Notifiche di Stato	563
Invio di file tramite WhatsApp	565
WhatsApp Download Media	567
API Telegram.....	568
Invia messaggio Telegram con pulsanti inline.....	576
Inviare un messaggio Telegram con pulsanti	577
Invia messaggio Telegram in grassetto	578
Chat Telegram non trovata come Bot	579
Messaggi sponsorizzati di Telegram.....	580
Invia il messaggio fattura Telegram.....	581
Telegram - Ottieni membri del supergruppo	582
Aggiungere un proxy Telegram	583
Registra utente Telegram.....	584

RCON	585
CryptoHopper	586
RTCMultiConnection	591
WebPush	593
TsgcWSAPIServer_WebPush	594
TsgcWebPush_Client.....	595
Estensioni	596
PerMessage-Deflate	597
Deflate-Frame	598
MCP	599
Sessioni server MCP	600
MCP Server Tools	601
Prompt del server MCP	604
MCP Server Resources.....	607
Radici del server MCP	610
MCP Server Sampling	611
Elicitation del server MCP.....	612
OpenAI.....	614
OpenAI Completion	621
OpenAI Chat.....	622
OpenAI Edit	623
OpenAI Audio.....	624
Moderazione OpenAI.....	625
OpenAI RealTime.....	626
OpenAI Responses	627
OpenAI Speech	628
OpenAI Fine-Tuning	629
OpenAI Batch.....	630
OpenAI Uploads	631
OpenAI Audio.....	632
Anthropic.....	633

Anthropic Messages.....	637
Anthropic Vision	639
Anthropic Tool Use	640
Anthropic Models.....	641
Anthropic Batches.....	642
Anthropic Extended Thinking	643
Anthropic Documents.....	645
Anthropic Prompt Caching.....	647
Anthropic Citations	649
Ricerca Web Anthropic	650
Anthropic Structured Outputs.....	651
Anthropic Files	653
Connettore MCP Anthropic.....	654
Gemini	656
DeepSeek	658
DeepSeek Messaggi	660
DeepSeek Vision.....	661
Modelli DeepSeek	662
Gemini	663
Messaggi Gemini	665
Gemini Vision.....	667
Gemini Models	668
Gemini Structured Outputs	669
Gemini Token Counting.....	670
Gemini Embeddings.....	671
Gemini Tool Use	672
Ollama	674
Messaggi Ollama	676
Modelli Ollama	677
Ollama Embeddings.....	678
Grok	679

Messaggi Grok	680
Grok Vision.....	681
Modelli Grok	682
Mistral AI	683
Mistral Messages.....	685
Visione con Mistral	686
Modelli Mistral	687
Mistral Embeddings	688
IoT	689
IoT Amazon MQTT Client.....	690
IoT Azure MQTT Client	697
HTTP.....	701
HTTP2	702
TsgcHTTP2Client.....	703
Metodo HTTP/2 Request	705
HTTP/2 Server Push	706
HTTP/2 Download File	707
Risposte parziali HTTP/2.....	708
Intestazioni HTTP/2	709
Client Chiudi Connessione	710
Mantenimento della connessione attiva del client	711
HTTP/2 Motivo della disconnessione.....	712
Richieste client in sospeso	713
Autenticazione client	714
HTTP/2 e OAuth2.....	715
TsgcHTTP2ConnectionClient.....	716
TsgcHTTP2RequestProperty	717
TsgcHTTP2ResponseProperty.....	718
Apple Push Notifications	719
Generare una notifica remota APNs.....	720
Invio di richieste di notifica ad APNs.....	721

Connessione basata su token ad APNs	722
Connessione ad APNs basata su certificato.....	723
HTTP1	725
OAuth2	729
TsgcHTTP_OAuth2_Client	730
OAuth2 Client for Web Applications	734
Client OAuth2 per applicazioni desktop.....	735
TsgcHTTP_OAuth2_Client_Google	736
TsgcHTTP_OAuth2_Client_Microsoft.....	737
Authorization Code Grant (RFC 6749).....	738
Authorization Code con PKCE (RFC 7636)	740
Client Credentials Grant (RFC 6749).....	742
Resource Owner Password Credentials Grant (RFC 6749).....	744
Device Authorization Grant (RFC 8628)	746
DPoP - Demonstrating Proof of Possession (RFC 9449).....	748
TsgcHTTP_OAuth2_Server	752
Esempio Server OAuth2	755
OAuth2 Personalizza HTML di Accesso	759
Endpoint server OAuth2.....	760
OAuth2 Registrazione App.....	761
OAuth2 Recover Access Tokens	762
Autenticazione server OAuth2.....	763
URL di autenticazione OAuth2 None	764
Authorization Code Grant	765
Client Credentials Grant	767
Resource Owner Password Credentials Grant	769
Refresh Token Grant.....	771
Device Authorization Grant (RFC 8628)	773
Revoca Token (RFC 7009)	775
Token Introspection (RFC 7662)	777
Validazione DPoP (RFC 9449).....	779

TsgcHTTP_OAuth2_Server_Provider	781
Provider OAuth2 Azure AD	783
OAuth2 Provider Private Endpoints	784
Autenticazione provider OAuth2.....	785
Richieste del provider OAuth2.....	787
JWT	788
TsgcHTTP_JWT_Client.....	790
TsgcHTTP_JWT_Server.....	793
WebAuthn	794
TsgcWSAPIServer_WebAuthn	795
Registrazione WebAuthn.....	799
Richiesta di Registrazione WebAuthn.....	802
Risposta di registrazione WebAuthn.....	803
Risultato della registrazione WebAuthn.....	805
Autenticazione WebAuthn	807
WebAuthn Authentication Request	809
Risposta di autenticazione WebAuthn.....	810
Risultato autenticazione WebAuthn	811
WebAuthn MDS	812
Autorizzazione WebAuthn.....	813
WebAuthn Authorization HTTP	814
WebAuthn Authorization WebSocket	815
Client Javascript Webauthn.....	816
Amazon SQS	820
Chiavi Google OAuth2.....	823
Google Cloud Pub/Sub	829
Google Calendar.....	838
Google Calendar Sync Calendars	844
Google Calendar Sincronizzazione eventi.....	845
Google Calendar RefreshToken.....	846
Google Cloud FCM.....	847

TsgcWebView2	850
Navigazione WebView2	853
JavaScript WebView2.....	855
Cookie WebView2.....	857
WebView2 Downloads	859
Impostazioni WebView2	861
Funzionalità avanzate WebView2.....	863
TsgcUDPClient	867
TsgcUDPServer	868
STUN	869
TsgcSTUNClient	870
Ritrasmissioni UDP Client STUN	872
STUN Client Long Term Credentials.....	873
STUN Client Attributes	874
TsgcSTUNServer	875
Credenziali a lungo termine del server STUN.....	876
Server STUN Alternate Server.....	877
TURN.....	878
TsgcTURNClient	879
TURN Client Allocate IP Address.....	881
Client TURN - Crea permessi.....	882
TURN Client Send Indication.....	883
TURN Client Channels.....	884
TsgcTURNServer	885
Credenziali a lungo termine del server TURN	887
Allocazioni del TURN Server	888
TsgcICEClient.....	889
TsgcRTCPeerConnection	890
Esempi	892
Chat server	892

Chat client	894
Client.....	896
Client MQTT	897
Client SocketIO	899
Monitor del server	900
Snapshot del server	903
Snapshot Client	904
Carica file.....	905
Autenticazione Server.....	907
KendoUI_Grid.....	908
ServerSentEvents	910
Server WebRTC	911
Server AppRTC	912
Telegram Client	914
Terze parti.....	916
Coturn.....	916
Licenza	918
Licenza	918
Indice.....	920
PDF-Back-Cover	927

Introduzione

sgcWebSockets è una libreria di componenti di livello professionale per la comunicazione in tempo reale, la messaggistica e l'integrazione con l'AI. Fornisce implementazioni pronte per la produzione di WebSockets, HTTP/2, MQTT, AMQP, WebRTC, Server-Sent Events e oltre 30 connettori API di terze parti, tutti accessibili tramite un'architettura a componenti coerente e basata sugli eventi. Che Lei stia realizzando piattaforme di trading, gateway IoT, applicazioni basate sull'AI o sistemi di messaggistica aziendali, sgcWebSockets fornisce le fondamenta di rete di cui ha bisogno.

La libreria supporta **da Delphi 7 a Delphi 13, C++Builder, Lazarus/FreePascal e .NET** (Framework 2.0+, .NET Core 1.0+, .NET 5–9, .NET Standard). Le applicazioni possono avere come destinazione Windows, macOS, Linux, iOS e Android da un'unica base di codice utilizzando VCL e FireMonkey.

Funzionalità .NET

- Server WebSocket **multithread completamente funzionante** che implementa **RFC 6455**.
- Assembly per **.NET Framework (2.0+), .NET Standard (1.6+), .NET Core (1.0+)**.
- Supporta **Windows 32/64, macOS 64 e Linux 64**.
- Modello di threading **IOCP** per server ad alte prestazioni.
- **Compressione dei messaggi** tramite l'estensione **PerMessage_Deflate** (RFC 7692).
- Supporto dei messaggi **di testo e binari**.
- **Autenticazione del server e del client**.
- Connessioni WebSocket e HTTP attraverso la **stessa porta**.
- **Server-Sent Events** (notifiche push) su HTTP.
- Supporto integrato per **WatchDog** e **HeartBeat**.
- Connessioni client **Socket.IO**.
- Client **Telegram**.
- **Binance Spot e Futures** (API WebSocket, User Stream e REST).
- Componenti client e server **STUN** e **TURN**.
- Connessioni client tramite server **HTTP Proxy** e **SOCKS Proxy**.
- Eventi: **OnConnect**, **OnDisconnect**, **OnMessage**, **OnError**, **OnHandshake**.
- **Librerie JavaScript** integrate per i client browser.
- **SSL/TLS** per i componenti server e client. **OpenSSL 1.1.1 e 3.0** supportati. Il client supporta **SChannel** su Windows.

I seguenti componenti sono inclusi nella libreria sgcWebSockets:

1 sgcWebSockets

- **TsgcWebSocketClient**: Client WebSocket basato sulla libreria Indy.
- **TsgcWebSocketServer**: Server WebSocket basato sulla libreria Indy
- **TsgcWebSocketHTTPServer**: Server WebSocket + HTTP basato sulla libreria Indy.
- **TsgcWebSocketServer_HTTPAPI**: Server WebSocket + HTTP ad alte prestazioni basato sulla Microsoft HTTP API HTTP.SYS.

2 sgcWebSocket APIs

- **TsgcWSAPI_Binance**: Client Binance Spot, supporta le API WebSocket + REST.
- **TsgcWSAPI_Binance_Futures**: Client Binance Futures, supporta le API WebSocket + REST.
- **TsgcWSAPI_SocketIO**: Client Socket.IO.

3 sgcWebSocket Libs

- **TsgcTDLib_Telegram**: Client API Telegram.

4 sgcWebSocket Protocols

- **TsgcWSPClient_MQTT**: Client MQTT (3.1.1 e 5.0). Supporta connessioni WebSocket e Plain TCP.
- **TsgcWSPServer_AppRTC**: Server WebRTC basato sul progetto Google AppRTC.
- **TsgcWSPServer_WebRTC**: Protocollo Server WebRTC.
- **TsgcWSPClient_Files**: Protocollo Client per il trasferimento di file WebSocket.
- **TsgcWSPServer_Files**: Protocollo Server per il trasferimento di file WebSocket.

5 sgcWebSockets HTTP

- [TsgcHTTP_JWT_Client](#): Client JWT (JSON WEB TOKEN).
- [TsgcHTTP_JWT_Server](#): Server JWT (JSON WEB TOKEN).
- [TsgcHTTP_OAuth2_Client](#): Client OAuth 2.0.
- [TsgcHTTP_OAuth2_Server](#): Server OAuth 2.0.

6 sgcWebSockets P2P

- [TsgcSTUNClient](#): Client STUN.
- [TsgcSTUNServer](#): Server STUN.
- [TsgcTURNClient](#): Client STUN / TURN.
- [TsgcTURNServer](#): Server STUN / TURN.

7 sgcWebSockets AI

Versioni supportate

Versioni .NET supportate

- .NET Framework 2.0+
- .NET Standard 1.6+
- .NET Core 1.0+
- Il pacchetto VSIX richiede .NET Framework 4.5
- Supporta Windows 32 / Windows 64 / OSX64

Installazione

Pacchetto Nuget

Può installare l'edizione Community di sgcWebSockets .NET dal seguente url nuget:

<https://www.nuget.org/packages/esegece.sgcWebSockets/>

Guardi i seguenti video che mostrano come installare il pacchetto NuGet e utilizzare i componenti sgcWebSockets

[Visual Studio Windows](#)

[Visual Studio Mac OS](#)

Riferimento all'assembly

Può lavorare con il pacchetto sgcWebSockets .NET senza utilizzare il pacchetto NuGet. Basta aprire il Suo progetto, selezionare **Aggiungi riferimento** dal menu contestuale del progetto e selezionare l'assembly che desidera aggiungere come riferimento dalla cartella Assemblies.

Assembly disponibili:

- .NET Framework 2.0
- .NET Framework 3.5
- .NET Framework 4.0
- .NET Framework 4.5
- .NET Framework 5.0
- .NET 6.0
- .NET 7.0
- .NET 8.0
- .NET Standard 1.6
- .NET Standard 2.0
- .NET Core 1.1
- .NET Core 2.0
- .NET Core 3.0

Si ricordi di copiare sgcWebSockets.dll (su Windows) o libsgcWebSockets.dylib (su OSX64)

QuickStart

Componenti WebSockets

Creare un nuovo server WebSocket o un nuovo client WebSocket è molto semplice: basta creare una nuova istanza della classe, configurare Host / Port e impostare la proprietà Active = true per avviare il processo.

[QuickStart WebSockets](#)

Componenti HTTP

Il protocollo HTTP/2 Le consente di creare server / client HTTP molto più veloci rispetto all'uso del protocollo HTTP/1. Il server HTTP/2 è incluso nel server WebSocket, mentre il client HTTP/2 è un componente dedicato che implementa il protocollo HTTP/2.

[QuickStart HTTP](#)

Flusso di threading

I componenti sgcWebSockets sono threaded, il che significa che **le connessioni vengono eseguite in thread secondari**. Per **impostazione predefinita**, i principali **eventi vengono distribuiti sul thread principale**; ciò è utile quando il numero di eventi da distribuire è basso, ma per **prestazioni migliori** è possibile configurare i componenti in modo che gli **eventi vengano distribuiti nel contesto del thread di connessione**. Legga il seguente articolo che spiega come configurare il flusso di threading:

[Come configurare NotifyEvents](#)

Come creare le applicazioni

Creare applicazioni con la libreria sgcWebSockets è molto facile. Basta seguire i suggerimenti seguenti, che La aiuteranno a **creare con successo la Sua applicazione**.

[Build](#)

OpenSSL

Quando la Sua applicazione richiede connessioni sicure, di solito sono necessarie le **librerie openssl** per **crittografare le comunicazioni**; segua i passaggi successivi per configurare correttamente la Sua applicazione con le librerie openssl.

[Configurare OpenSSL](#)

ASP.NET

Può utilizzare sgcWebSockets nella Sua applicazione ASP.NET, tenga solo presente che sgcWebSockets richiede alcune dll non gestite (come sgcWebSockets.dll) e, per impostazione predefinita, i Suoi progetti ASP.NET non tro-

veranno queste librerie. Per impostare il percorso della cartella bin del Suo progetto, imposti il PATH con un codice come questo nel Suo file Global.asax

```
protected void Application_Start()
{
    var path = string.Concat(Environment.GetEnvironmentVariable("PATH"), ";",
        AppDomain.CurrentDomain.RelativeSearchPath);
    Environment.SetEnvironmentVariable("PATH", path, EnvironmentVariableTarget.Process);
    .....
}
```

QuickStart | WebSockets

Iniziamo con un esempio di base in cui è necessario creare un Server WebSocket e 2 tipi di client WebSocket: Client Applicazione e Client Browser Web.

Server WebSocket

1. Create a new Window Forms Application
2. Rilasciare un TsgcWebSocketServer su un Form.
3. Nella scheda Eventi, fare doppio clic sull'evento OnMessage e digitare il seguente codice:

```
private void OnMessage(TsgcWSConnection Connection, const string Text)
{
    MessageBox.Show("Message Received From Client: " + Text);
}
```

4. Trascinare un pulsante nel form, fare doppio clic e digitare questo codice:

```
TsgcWebSocketServer1.Active = True;
```

WebSocket Client

1. Create a new Window Forms Application
2. Posizionare un TsgcWebSocketClient su un Form e configurare le proprietà Host e Port per connettersi al server.
3. Inserire un TButton in un Form, fare doppio clic e digitare questo codice:

```
TsgcWebSocketClient1.Active = true;
```

4. Trascinare un pulsante nel form, fare doppio clic e digitare questo codice:

```
TsgcWebSocketClient1.WriteData("Hello Server From VCL Client");
```

Client Browser Web

1. Creare un nuovo file HTML
2. Aprire il file con un editor di testo e copiare il seguente codice:

```
<html>
<head>
<script type="text/javascript" src="http://host:port/sgcWebSockets.js"></script>
</head>
<body>
<a href="javascript:var socket = new sgcWebSocket('ws://host:port');">Open</a>
<a href="javascript:socket.send('Hello Server From Web Browser');">Send</a>
</body>
</html>
```

È necessario sostituire host e porta in questo file con il proprio Host e Porta personalizzati!

3. Salvare il file e il gioco è fatto: è stato configurato un client WebSocket di base per il browser Web.

Come Utilizzare

1. Avviare l'applicazione server e premere il pulsante per avviare il server WebSocket in ascolto di nuove connessioni.
2. Avviare l'applicazione client e premere il pulsante button1 per connettersi al server, quindi premere button2 per inviare un messaggio. Sul lato server verrà visualizzato un messaggio con il testo inviato dal client.
3. Aprire il file HTML con il proprio browser Web (Chrome, Firefox, Safari o Internet Explorer 10+), premere Apri per aprire una connessione e premere Invia per inviare un messaggio al server. Sul lato server, verrà visualizzato un messaggio con il testo inviato dal client del browser Web.

ASP.NET

Può utilizzare sgcWebSockets nella Sua applicazione ASP.NET, tenga solo presente che sgcWebSockets richiede alcune dll non gestite (come sgcWebSockets.dll) e, per impostazione predefinita, i Suoi progetti ASP.NET non troveranno queste librerie. Per impostare il percorso della cartella bin del Suo progetto, imposti il PATH con un codice come questo nel Suo file Global.asax

```
protected void Application_Start()
{
    var path = string.Concat(Environment.GetEnvironmentVariable("PATH"), ";",
        AppDomain.CurrentDomain.RelativeSearchPath);
    Environment.SetEnvironmentVariable("PATH", path, EnvironmentVariableTarget.Process);
    .....
}
```

QuickStart | HTTP

Iniziamo con un esempio di base in cui è necessario creare un server HTTP/2 e un client HTTP/2.

Server HTTP/2

1. Create a new Window Forms Application
2. Trascini un TsgcWebSocketHTTPServer su un Form.
3. Nella scheda Events, fare doppio clic sull'evento OnCommandGet e digitare il seguente codice:

```
void OnCommandGet(TsgcWSConnection Connection, TsgcWSHTTPRequestInfo RequestInfo,
    ref TsgcWSHTTPResponseInfo ResponseInfo)
{
    if (RequestInfo.Document == "/")
    {
        ResponseInfo.ContentText = "<html><head><title>Test Page</title></head><body></body></html>";
        ResponseInfo.ContentType = "text/html";
        ResponseInfo.ResponseNo = 200;
    }
}
```

4. Per impostazione predefinita, il server abilita solo le connessioni HTTP/1; pertanto abilitare le opzioni HTTP/2 nella proprietà **HTTP2Options.Enabled = true**, quindi configurare le opzioni SSL. Le connessioni sicure richiedono le [librerie OpenSSL](#).

```
TsgcWebSocketHTTPServer1.Port := 443;
TsgcWebSocketHTTPServer1.SSL := true;
TsgcWebSocketHTTPServer1.SSLOptions.CertFile = "server cert file";
TsgcWebSocketHTTPServer1.SSLOptions.KeyFile = "server private key file";
TsgcWebSocketHTTPServer1.SSLOptions.RootCertFile = "server root cert file";
TsgcWebSocketHTTPServer1.SSLOptions.OpenSSL_Options.APIVersion = TwsOpenSSLAPI.oslAPI_1_1;
TsgcWebSocketHTTPServer1.SSLOptions.Port = 443;
TsgcWebSocketHTTPServer1.SSLOptions.Version = TwsTLSVersions.tls1_3;
```

5. Trascinare un Button sul Form, fare doppio clic e digitare questo codice:

```
TsgcWebSocketHTTPServer1.Active = True;
```

Client HTTP/1

1. Create a new Window Forms Application
2. Trascini un TButton in un Form, faccia doppio clic e digiti questo codice:

```
TsgcHTTP1Client oHTTP1 = new TsgcHTTP1Client();
MessageBox.Show(oHTTP1.Get("https://127.0.0.1"));
```

HTTP/2 Client

1. Create a new Window Forms Application

2. Trascini un TButton in un Form, faccia doppio clic e digiti questo codice:

```
TsgcHTTP2Client oHTTP2 = new TsgcHTTP2Client();  
MessageBox.Show(oHTTP2.Get("https://127.0.0.1"));
```


QuickStart | Flusso di threading

I componenti `sgcWebSockets` sono threaded; ad esempio, `TsgcWebSocketHTTPServer` (basato sulla libreria `Indy`) crea un thread per ogni connessione, mentre `TsgcWebSocketServer_HTTPAPI` (basato su `Microsoft HTTP.SYS`) esegue un pool di thread e le connessioni vengono gestite da questo pool di thread (massimo 64 thread) e `TsgcWebSocketClient` esegue il proprio thread per elaborare in modo asincrono le risposte dal server `WebSocket`.

Per impostazione predefinita, esiste una proprietà chiamata **NotifyEvents**, che ha il valore `neAsynchronous`. Ciò significa che quando un client `WebSocket` riceve un messaggio, questo viene messo in coda e inviato al thread principale dal sistema operativo in un secondo momento. Questo funziona bene per i client che non ricevono molti messaggi e per semplicità d'uso, perché non richiede la sincronizzazione con il thread principale quando si desidera, ad esempio, aggiornare un controllo nel proprio form.

Ma quando il server/client deve elaborare diversi messaggi in un breve periodo di tempo, è preferibile modificare questo flusso di threading in uno in cui gli eventi vengono inviati nel contesto del thread di connessione. Per farlo, è sufficiente impostare la proprietà `NotifyEvents` su **neNoSync**, in questo modo, quando ad esempio un client riceve un messaggio dal server, questo messaggio verrà inviato nel contesto di un thread secondario; pertanto, se è necessario aggiornare un controllo del modulo, occorre prima sincronizzarsi con il thread principale e poi aggiornare il controllo del modulo (poiché i controlli dei moduli non sono thread-safe). Lo stesso vale se si desidera accedere a un oggetto condiviso: è necessario implementare i propri metodi di sincronizzazione.

Flusso di threading Modalità semplice (`NotifyEvents = neAsynchronous`) e basse prestazioni

Questo è il flusso di threading predefinito ed è solitamente utilizzato negli esempi dimostrativi. Selezionare questa modalità se non si prevede di gestire diversi messaggi al secondo e si ha la necessità di aggiornare i controlli del form o accedere a oggetti condivisi.

`NotifyEvents = neAsynchronous`

Flusso di threading Prestazioni migliori (`NotifyEvents = neNoSync`)

Impostare questo flusso di threading per i componenti server e per i client che necessitano di alte prestazioni perché si prevede di gestire diversi messaggi. Con questa configurazione, gli eventi vengono inviati nel contesto del thread di connessione; pertanto, per aggiornare un controllo di un Form, è necessario prima sincronizzarsi con il thread principale.

`NotifyEvents = neNoSync`

```
void OnClientMessage(TsgcWSConnection Connection, string Text)
{
    Invoke((MethodInvoker)delegate
    {
        memo1.AppendText(Text + Environment.NewLine);
    });
}
```

QuickStart | Build

Costruire un'applicazione con la libreria `sgcWebSockets` è molto semplice. È sufficiente tenere a mente se i componenti utilizzati richiedono o meno le librerie OpenSSL. Se le applicazioni richiedono connessioni sicure, è necessario distribuire le librerie OpenSSL (a meno che non si utilizzi [SChannel per Windows](#) sui componenti client).

Per le **applicazioni Windows**, è sufficiente distribuire le librerie OpenSSL nella stessa cartella in cui si trova l'applicazione.

Server ad alte prestazioni

Server basati sulla libreria Indy

[TsgcWebsocketServer](#) e [TsgcWebsocketHTTPServer](#) sono basati sulla libreria Indy, quindi ogni connessione è gestita da un thread; con 1000 connessioni simultanee si avranno almeno 1000 thread. Quando le prestazioni sono importanti, è necessario apportare alcune "ottimizzazioni" per migliorare il lavoro del server. **A partire da sgcWebSockets 4.3.3 i server Indy supportano anche IOCP**; è possibile [approfondire](#).

Utilizzare i seguenti suggerimenti per aumentare le prestazioni del server.

1. Impostare nella proprietà del componente Server **NotifyEvents := neNoSync**. Ciò significa che gli eventi vengono generati nel contesto del thread della connessione, quindi non è presente alcun meccanismo di sincronizzazione. Se è necessario accedere a controlli VCL o oggetti condivisi, utilizzare i propri meccanismi di sincronizzazione.

2. Impostare nella proprietà del componente Server **Optimizations.Connections.Enabled := True**. Se si prevede di avere più di 1000 connessioni simultanee nel server e si chiama frequentemente il metodo `Server.WriteData`, abilitare questa proprietà. In sostanza, salva le connessioni in una lista cache in cui le ricerche sono più rapide rispetto all'accesso alla lista delle connessioni Indy.

2.1 CacheSize: è il numero di connessioni memorizzate in una cache veloce. Valore predefinito = 100.

2.2 GroupLevel: crea internamente diversi elenchi suddivisi per il primo carattere, quindi se si dispone di molte connessioni, le ricerche sono più veloci. Valore predefinito = 1.

3. Impostare nella proprietà del componente Server **Optimizations.Channels.Enabled := True**. Abilitando questa proprietà, i canali vengono salvati in un elenco in cui le ricerche sono più veloci rispetto al metodo precedente.

4. Impostare nella proprietà del componente Server **Optimizations.ConnectionsFree.Enabled := True**. Se questa proprietà è abilitata, ogni volta che si verifica una disconnessione, invece di distruggere `TsgcWSCConnection`, l'oggetto viene memorizzato in un elenco e ogni X secondi tutti gli oggetti memorizzati in questo elenco vengono distrutti. L'abilitazione di questa proprietà aumenterà il consumo di memoria, quindi se si hanno molte disconnessioni in un breve periodo di tempo, impostare questa proprietà su false.

4.1 Interval: numero di secondi entro cui vengono distrutte tutte le connessioni disconnesse memorizzate in un elenco. Per impostazione predefinita è 60.

5. Per impostazione predefinita, `sgcWebSockets` utilizza **sezioni critiche** per proteggere l'accesso agli oggetti condivisi. È tuttavia possibile utilizzare `TMonitor` o `SpinLock` al posto delle sezioni critiche. Basta compilare il progetto con una delle seguenti definizioni del compilatore

3.1 `{ $DEFINE SGC_SPINLOCKS }`

3.2 `{ $DEFINE SGC_TMONITOR }`

6. Utilizzare l'ultima versione di **FastMM4**, scaricabile da: <https://github.com/pleriche/FastMM4>

FastMM4 è un ottimo gestore della memoria, ma a volte non scala bene con le applicazioni multi-thread. Utilizzare la seguente direttiva del compilatore nell'applicazione:

`{ $DEFINE UseReleaseStack }`

Quindi aggiungere FastMM4 come prima unità nelle uses del progetto e ricompilare. Per un server ad alta concorrenza, si noterà un aumento delle prestazioni.

Questa modifica esegue le seguenti operazioni: se un blocco non può essere rilasciato immediatamente durante una chiamata `FreeMem`, il blocco verrà aggiunto a un elenco di blocchi che saranno liberati in seguito, o nel thread di pulizia in background o durante la successiva chiamata a `FreeMem`.

7. Meglio di FastMM4, utilizzare la versione più recente **FastMM5**, scaricabile da: <https://github.com/pleriche/FastMM5>

Questa è una nuova versione dello stesso sviluppatore di FastMM4, supporta Delphi XE3 e successivi e può essere utilizzata su Windows32 e Windows64.

FastMM5 è distribuito con doppia licenza: GPL e Commerciale. Pertanto, se si desidera utilizzarlo in progetti commerciali, è necessario acquistare una licenza.

Di seguito è riportata una griglia che confronta le prestazioni tra FastMM4 e FastMM5, eseguendo 100.000 richieste e risposte WebSocket con 1, 10, 100, 500 e 1000 client concorrenti. Le prestazioni con FastMM5 sono nettamente migliori, nelle applicazioni multithread, rispetto a FastMM4.

Client	Windows	FMM4	FMM5	Differenza
1	Win32	4135	4214	1,91%
	Win64	4052	4520	11,55%
10	Win32	4214	1729	-58,97%
	Win64	4104	1875	-54,31%
100	Win32	3958	1604	-59,47%
	Win64	3958	1614	-59,22%
500	Win32	4098	1723	-57,96%
	Win64	5333	1791	-66,42%
1000	Win32	5927	2208	-62,75%
	Win64	8166	2229	-72,70%

Indy Server Windows

sgcWebSockets Enterprise Edition supporta **IOCP** su Windows; ciò significa che invece di creare 1 thread per ogni connessione, un pool di thread gestisce tutte le connessioni. Per abilitare IOCP, impostare semplicemente `IOHandler` su `IOCP`.

```
IOHandlerOptions.IOHandlerType = iohIOCP
```

La proprietà `IOHandlerOptions.IOCP` consente di personalizzare le proprietà IOCP.

- **IOCPThreads:** questi sono i thread utilizzati per gestire le connessioni; per impostazione predefinita il valore è zero, il che significa che i thread verranno calcolati automaticamente in base al numero di processori (per Delphi 7 fino a Delphi 2007 questo valore è impostato a 32 perché la funzione di conteggio CPU non è supportata).
- **WorkOpThreads:** impostare un valore maggiore di zero se si desidera che le richieste per ogni connessione vengano sempre gestite dallo stesso thread. Per impostazione predefinita, le richieste IOCP sono gestite da thread casuali; se si vuole che le connessioni vengano sempre gestite dallo stesso thread, impostare un valore maggiore di zero. Esempio: se si imposta `WorkOpThreads = 32`, il server creerà 32 thread e ogni volta che arriva una nuova richiesta, se la connessione è già stata elaborata in precedenza verrà accodata nello stesso thread.

IOCP è consigliato quando si desidera gestire migliaia di connessioni simultanee.

Indy Server Linux

sgcWebSockets Enterprise Edition supporta **EPOLL** su Linux, il che significa che invece di creare 1 thread per ogni connessione, un pool di thread gestisce tutte le connessioni. Per abilitare EPOLL, è sufficiente impostare `IOHandler` su EPOLL.

```
IOHandlerOptions.IOHandlerType = iohEPOLL
```

La proprietà `IOHandlerOptions.EPOLL` consente di personalizzare le proprietà EPOLL.

- **EPOLLThreads:** questi sono i thread utilizzati per gestire le connessioni; per impostazione predefinita il valore è zero, il che significa che i thread verranno calcolati automaticamente in base al numero di processori.
- **WorkOpThreads:** impostare un valore maggiore di zero se si desidera che le richieste per ogni connessione vengano gestite sempre dallo stesso thread. Per impostazione predefinita, le richieste EPOLL vengono gestite da thread casuali; se si desidera che le connessioni vengano gestite sempre dallo stesso thread, impostare un valore maggiore di zero. Esempio: se si imposta `WorkOpThreads = 32`, il server creerà 32 thread e ogni volta che arriva una nuova richiesta, se la connessione è già stata elaborata in precedenza, verrà accodata nello stesso thread.

EPOLL è consigliato quando si desidera gestire migliaia di connessioni simultanee.

Server Basato su HTTP.SYS

Il componente `TsgcWebSocketServer_HTTPAPI` è basato sull'API HTTP di Microsoft ed è progettato per funzionare con IOCP, quindi è consigliato quando il server deve gestire migliaia di connessioni, ma ha il limite di poter funzionare solo su Windows.

Il server può gestire i protocolli **WebSocket** e **HTTP/2** sulla stessa porta e può funzionare con altre implementazioni perché può essere configurato per gestire solo alcuni endpoint.

Esempio: è possibile configurare questo server per gestire le connessioni WebSocket con la nostra libreria `sgcWebSockets` e lasciare che altre implementazioni / terze parti o quant'altro utilizzino altri endpoint.

- Endpoint: `https://server/ws` gestirà le connessioni che utilizzano il protocollo WebSocket con `sgcWebSockets`
- Endpoint: `https://server/other` gestirà le connessioni utilizzando un'altra libreria.

Utilizzare l'ultima versione di **FastMM5**, scaricabile da: <https://github.com/pleriche/FastMM5>

Questa è una nuova versione dello stesso sviluppatore di FastMM4, supporta Delphi XE3 e successivi e può essere utilizzata su Windows32 e Windows64.

FastMM5 è distribuito con doppia licenza: GPL e Commerciale. Pertanto, se si desidera utilizzarlo in progetti commerciali, è necessario acquistare una licenza.

Di seguito è riportata una griglia che confronta le prestazioni tra FastMM4 e FastMM5, eseguendo 100.000 richieste e risposte WebSocket con 1, 10, 100, 500 e 1000 client concorrenti. Le prestazioni con FastMM5 sono nettamente migliori, nelle applicazioni multithread, rispetto a FastMM4.

Client	Windows	FMM4	FMM5	Differenza
1	Win32	5364	5182	-3,39%
	Win64	5057	5026	-0,61%
10	Win32	4922	1744	-64,57%
	Win64	4958	1770	-64,30%
100	Win32	3359	1682	-49,93%
	Win64	3979	1536	-61,40%

500	Win32	2364	1890	-20,05%
	Win64	2901	1666	-42,57%
1000	Win32	3296	1968	-40,29%
	Win64	4469	1989	-55,49%

OpenSSL

OpenSSL è una libreria software per applicazioni che proteggono le comunicazioni su reti informatiche dalle intercettazioni o che devono identificare la controparte. È ampiamente utilizzata dai server Internet, inclusa la maggior parte dei siti HTTPS.

Questa libreria è richiesta dai componenti basati sulla libreria Indy quando è necessaria una connessione sicura. Se l'applicazione richiede OpenSSL, è necessario che i file necessari siano presenti nel file system prima di distribuire l'applicazione:

Attualmente, sgcWebSockets supporta le versioni: **1.0.2, 1.1 e 3.0 a 3.3 openssl**.

Platform	API 1.0	API 1.1	API 3.*	Collegamento statico/dinamico
Windows (32-bit e 64-bit)	libeay32.dll and ssleay32.dll	libcrypto-1_1.dll and libssl-1_1.dll	libcrypto-3.dll e libssl-3.dll	Dinamico
OSX	libcrypto.dylib, libssl.dylib	libcrypto.1.1.dylib, libssl.1.1.dylib	libcrypto.3.dylib, libssl.3.dylib	Dinamico
iOS Device (32-bit and 64-bit)	libcrypto.a e libssl.a	libcrypto.a e libssl.a	libcrypto.a e libssl.a	Statico
iOS Simulator	libcrypto.dylib, libssl.dylib	libcrypto.1.1.dylib, libssl.1.1.dylib	libcrypto.3.dylib, libssl.3.dylib	Dinamico
Dispositivo Android	libcrypto.so, libssl.so	libcrypto.so, libssl.so	libcrypto.so, libssl.so	Dinamico

Trovi di seguito come **configurare OpenSSL** per ciascuna piattaforma:

- [Windows](#)
- [OSX](#)

Configurazioni openssl

I componenti di sgcWebSockets basati su Indy Le consentono di configurare alcune proprietà di OpenSSL. Accesso alle seguenti proprietà:

- **Componenti server:** SSLOptions.OpenSSL_Options.
- **Componenti client:** TLSOptions.OpenSSL_Options.

Versione API

La **libreria Indy standard** consente solo il caricamento delle librerie **OpenSSL 1.0.2**; queste librerie sono state deprecate e le ultime versioni di OpenSSL utilizzano l'API 1.1.1.

sgcWebSockets Enterprise consente di caricare le librerie **1.1.1 openssl**; in questa proprietà è possibile configurare quale versione dell'API openssl verrà caricata. Solo una versione dell'API può essere caricata per processo (quindi non è possibile combinare le librerie openssl 1.0.2 e 1.1.1 nella stessa applicazione).

LibPath

Questa proprietà consente di impostare la posizione delle librerie openssl. È utile per i progetti Android o OSX, dove è necessario impostare la posizione delle librerie openssl.

Accetta i seguenti valori:

- **oslpNone**: questo valore non imposta alcun valore di percorso della libreria (è il valore predefinito).
- **oslpDefaultFolder**: questo valore imposta la cartella predefinita delle librerie openssl. Questo percorso è diverso per ogni personalità (windows, osx...).

Carica funzioni OpenSSL aggiuntive

Utilizzare un callback per caricare funzioni OpenSSL aggiuntive non definite per impostazione predefinita. Per ulteriori informazioni consultare [OpenSSL: caricamento di funzioni aggiuntive](#).

Ciphers

Se si desidera fornire supporto per TLS 1.2 e 1.3 sul proprio server utilizzando la migliore sicurezza e prestazioni, utilizzare la seguente configurazione:

```
SSLOptions.Version := tls1_3;
SSLOptions.OpenSSL_Options.VersionMin := tls1_2;
SSLOptions.OpenSSL_Options.APIVersion := oslAPI_3_0;
```

E impostare il seguente elenco di cifrature.

```
AEAD-AES128-GCM-SHA256:AEAD-AES256-GCM-SHA384:AEAD-CHACHA20-POLY1305-SHA256:ECDHE-
ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-
GCM-SHA384
```

Certificati autofirmati

È possibile utilizzare certificati auto-firmati a scopo di test. È sufficiente eseguire il seguente comando per creare un certificato auto-firmato:

```
openssl req -newkey rsa:2048 -new -nodes -x509 -days 3650 -keyout key.pem -out cert.pem
```

Verranno creati 2 file: cert.pem (certificato) e key.pem (chiave privata). È possibile combinare entrambi i file in uno solo. È sufficiente creare un nuovo file e copiare il contenuto di entrambi i file in esso. Si otterrà così una struttura simile alla seguente:

```
-----BEGIN PRIVATE KEY-----
....
-----END PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
....
-----END CERTIFICATE-----
```

Errori Comuni

SSL_GET_RECORD: wrong version number

Questo errore indica che il server e il client stanno utilizzando versioni diverse del protocollo SSL/TLS. Per risolvere il problema, provare a impostare la versione corretta nel componente server e/o client.

```
Server.SSLOptions.Version
```


`Client.TLSOptions.Version`

SSL3_GET_RECORD: decryption failed or bad record mac

Solitamente questo errore si genera quando:

1. Verificare che si stia utilizzando la versione più recente di OpenSSL. Se è troppo vecchia, aggiornare alla versione supportata più recente.
2. Se questo errore compare in modo casuale, di solito è perché più di un thread sta accedendo alla connessione OpenSSL. Può provare a impostare `NotifyEvents = neNoSync`, il che significa che gli eventi: `OnConnect`, `OnDisconnect`, `OnMessage...` verranno generati nel contesto del thread di connessione, questo evita alcuni problemi di sincronizzazione e fornisce prestazioni migliori. Come aspetto negativo, se ad esempio sta aggiornando un controllo visivo in un form quando riceve un messaggio, deve implementare i Suoi metodi di sincronizzazione perché i controlli visivi non sono thread-safe.

OpenSSL | Windows

Esiste una versione per 32 bit e un'altra per 64 bit. Deve copiare queste librerie nella stessa cartella in cui si trova la sua applicazione oppure nel suo system path.

Se il sistema operativo è Windows a 32 bit, copiare semplicemente nella cartella System32.

Se il sistema operativo è Windows a 64 bit, copiare la versione a 64 bit nella cartella System32 e la versione a 32 bit nella cartella SysWow64.

API 1.0

Richiede le seguenti librerie:

- libeay32.dll
- ssleay32.dll

Se il sistema operativo è Windows a 32 bit, copiare semplicemente nella cartella System32.

Se il sistema operativo è Windows a 64 bit, copiare la versione a 64 bit nella cartella System32 e la versione a 32 bit nella cartella SysWow64.

Lei può scaricare le librerie più recenti dal Suo account (le librerie non hanno dipendenze esterne e sono firmate digitalmente).

API 1.1

Richiede le seguenti librerie:

Windows 32

- libcrypto-1_1.dll
- libssl-1_1.dll

Windows 64

- libcrypto-1_1-x64.dll
- libssl-1_1-x64.dll

Se il sistema operativo è Windows a 32 bit, copiare semplicemente nella cartella System32.

Se il sistema operativo è Windows a 64 bit, copiare la versione a 64 bit nella cartella System32 e la versione a 32 bit nella cartella SysWow64.

Lei può scaricare le librerie più recenti dal Suo account (le librerie non hanno dipendenze esterne e sono firmate digitalmente).

API 3.*

Richiede le seguenti librerie:

Windows 32

- libcrypto-3.dll
- libssl-3.dll

Windows 64

- libcrypto-3-x64.dll
- libssl-3-x64.dll

Se il sistema operativo è Windows a 32 bit, copiare semplicemente nella cartella System32.

Se il sistema operativo è Windows a 64 bit, copiare la versione a 64 bit nella cartella System32 e la versione a 32 bit nella cartella SysWow64.

Lei può scaricare le librerie più recenti dal Suo account (le librerie non hanno dipendenze esterne e sono firmate digitalmente).

Se si utilizza un certificato p12, è necessario distribuire la libreria legacy.dll. Per ulteriori informazioni, consultare [Certificati OpenSSL p12](#).

OpenSSL | OSX

Le versioni più recenti di OSX non includono le librerie OpenSSL o le includono in versioni obsolete, pertanto è necessario distribuirle insieme all'applicazione. Distribuire queste librerie seguendo i passaggi indicati:

- Aprire Progetto/Deployment nel proprio progetto.
- Aggiungere le librerie richieste.
- Impostare RemotePath = 'Contents\Macos\'.
 • Configurare il LibPath di openssl sulla cartella predefinita:
 - Client.TLSOptions.OpenSSL_Options.LibPath = oslpDefaultFolder.
 - Server.SSLOptions.OpenSSL_Options.LibPath = oslpDefaultFolder.

API 1.0

Richiede le seguenti librerie:

- libcrypto.dylib
- libssl.dylib

È possibile scaricare le librerie più recenti dal proprio account.

API 1.1

Richiede le seguenti librerie:

- libcrypto.1.1.dylib
- libssl.1.1.dylib

Esiste una versione per 32 bit e un'altra per 64 bit. È necessario copiare queste librerie nella stessa cartella in cui si trova l'applicazione.

È possibile scaricare le librerie più recenti dal proprio account.

API 3.0

Richiede le seguenti librerie:

- libcrypto.3.dylib
- libssl.3.dylib

Viene fornita solo la versione a 64 bit. È necessario copiare queste librerie nella stessa cartella in cui si trova l'applicazione.

È possibile scaricare le librerie più recenti dal proprio account.

Se si includono le librerie OpenSSL in un'applicazione OSX, dopo che l'applicazione è stata notarizzata, le librerie saranno firmate; è possibile verificarlo con il seguente comando:

```
codesign -dv --verbose=4 libcrypto.1.1.dylib
```

Guardare il seguente video che mostra come creare un'applicazione MacOSX64 con le librerie OpenSSL.

<https://www.esegece.com/websockets/videos/delphi/quickstart/275-build-macosx64-application/file>

Errori

I client non dovrebbero caricare la dylib libcrypto senza versione poiché non dispone di un'ABI stabile.

Su MacOS Monterey e versioni successive, potrebbe verificarsi questo errore durante il tentativo di caricare le librerie OpenSSL; l'errore si verifica quando si tenta di caricare prima le librerie OpenSSL senza versione (ad esempio libcrypto.dylib).

Per correggere questo errore, impostare nella proprietà **OpenSSL_Options.UnixSymLinks** il valore **oslsSymLinksDontLoad**. Questo evita il caricamento delle librerie openSSL senza versione.

OpenSSL | Certificati CA propri

[Post su Github](#)

Per creare un certificato firmato dalla propria CA e che può essere considerato attendibile dai browser web (come Chrome) dopo aver aggiunto il certificato CA alla macchina locale.

1. Preparare i file di configurazione per la creazione di certificati senza prompt

CA.cnf

```
[ req ]
prompt = no
distinguished_name = req_distinguished_name
[ req_distinguished_name ]
C = US
ST = Localzone
L = localhost
O = Certificate Authority Local Center
OU = Develop
CN = develop.localhost.localdomain
emailAddress = root@localhost.localdomain
```

localhost.cnf

```
[req]
default_bits = 2048
distinguished_name = req_distinguished_name
req_extensions = req_ext
x509_extensions = v3_req
prompt = no
[req_distinguished_name]
countryName = US
stateOrProvinceName = Localzone
localityName = Localhost
organizationName = Certificate signed by my CA
commonName = localhost.localdomain
[req_ext]
subjectAltName = @alt_names
[v3_req]
subjectAltName = @alt_names
[alt_names]
IP.1 = 127.0.0.1
IP.2 = 127.0.0.2
IP.3 = 127.0.0.3
IP.4 = 192.168.0.1
IP.5 = 192.168.0.2
IP.6 = 192.168.0.3
DNS.1 = localhost
DNS.2 = localhost.localdomain
DNS.3 = dev.local
```

2. Generare una chiave privata CA e un Certificato (valido per 5 anni)

```
openssl req -nodes -new -x509 -keyout CA_key.pem -out CA_cert.pem -days 1825 -config CA.cnf
```

3. Genera la chiave segreta del server web e il CSR

```
openssl req -sha256 -nodes -newkey rsa:2048 -keyout localhost_key.pem -out localhost.csr -config localhost.cnf
```

4. Creare il certificato e firmarlo con la propria autorità di certificazione (validità 1 anno)

```
openssl x509 -req -days 398 -in localhost.csr -CA CA_cert.pem -CAkey CA_key.pem -CAcreateserial -out localhost_certificate.pem
```

5. I file di output saranno:

- `CA.cnf` → file di configurazione della CA OpenSSL. Può essere eliminato dopo il processo di creazione del certificato.
- `CA_cert.pem` → Certificato [Autorità di certificazione]. Questo certificato deve essere aggiunto all'archivio dell'autorità locale del browser per rendere attendibili tutti i certificati creati con questa CA.
- `CA_cert.srl` → Numero seriale casuale. Può essere eliminato dopo il processo di creazione del certificato.
- `CA_key.pem` → Deve essere utilizzato durante la creazione di un nuovo certificato [localhost]. Può essere eliminato dopo il processo di creazione del certificato (se non si prevede di riutilizzarlo insieme a `CA_cert.pem`).
- `localhost.cnf` → File di configurazione del certificato SSL OpenSSL. Può essere eliminato dopo il processo di creazione del certificato.
- `localhost.csr` → Richiesta di firma del certificato. Può essere eliminato dopo il processo di creazione del certificato.
- `localhost_cert.pem` → certificato SSL. **Deve essere configurato nella proprietà `SSLOptions.CertFile` del server.**
- `localhost_key.pem` → Chiave segreta. **Deve essere installata nella proprietà `SSLOptions.KeyFile` del server.**

OpenSSL | Certificati P12

OpenSSL 3.0 ha spostato diversi algoritmi deprecati o non sicuri in un modulo di libreria interno denominato legacy provider. Non viene caricato per impostazione predefinita, quindi le applicazioni (o i relativi runtime di linguaggio) che utilizzano OpenSSL per operazioni crittografiche non possono utilizzare tali algoritmi durante il caricamento dei certificati, la creazione di message digest...

Gli algoritmi nel provider legacy includono MD2, MD4, MDC2, RMD160, CAST5, BF (Blowfish), IDEA, SEED, RC2, RC4, RC5 e DES (ma non 3DES).

Per motivi di sicurezza, si raccomanda vivamente di abbandonare l'utilizzo di questi algoritmi legacy.

Se l'applicazione utilizza certificati client memorizzati in un file crittografato con un cifrario legacy come RC2-40-CBC, è possibile "modernizzare" il file del certificato ri-crittografandolo con il programma openssl.

Ad esempio, se si dispone di un file certificato client.p12 (o client.pfx) sul computer locale:

```
$ openssl pkcs12 -legacy -in client.p12 -nodes -out cert-decrypted.tmp
(inserire le passphrase se richiesto)
```

```
$ openssl pkcs12 -in cert-decrypted.tmp -export -out client-new.p12
(inserire le passphrase se richieste)
```

```
$ rm cert-decrypted.tmp
```

Il file di certificato client-new.p12 esportato contiene ora le stesse chiavi, ma cifrate con AES-256-CBC.

Verificare di seguito la configurazione per i pacchetti sgcWebSockets e sgclndy:

sgcWebSockets

- Impostare la proprietà **OpenSSL_Options.Legacy.Enabled** su True.
- Imposta la posizione della libreria Legacy.
 - **OpenSSL_Options.Legacy.LibPath**: qui può configurare dove si trova la libreria legacy
 - **oslpNone**: questa è l'impostazione predefinita, la libreria legacy dovrebbe trovarsi nella stessa cartella in cui si trova il binario o in un percorso noto.
 - **oslpDefaultFolder**: imposta automaticamente il percorso della libreria legacy dove le librerie dovrebbero trovarsi per tutte le personalità dell'IDE.
 - **oslpCustomFolder**: se questa è l'opzione selezionata, definisca il percorso completo nella proprietà LibPathCustom.
 - **OpenSSL_Options.Legacy.LibPathCustom**: quando LibPath = oslpCustomFolder definisca qui il percorso completo dove si trova la libreria legacy.

sgclndy

- Impostare la proprietà **SSLOptions.Legacy** su True.
- Prima di avviare il server o il client, imposti il percorso in cui si trova la libreria legacy.dll. Utilizzi la funzione **IdOpenSSLSetOSSLPPath** e passi il percorso come argomento.

OpenSSL | Verifica certificato

Quando si utilizza OpenSSL e si imposta l'opzione Verifica certificato, potrebbe apparire il seguente errore:

Error connecting with SSL.error:80000002:system library::No such file or directory.
Se si gestisce l'evento OnVerifyPeer e il parametro Error ha un valore di 20, l'errore significa:

X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY

Il motivo principale di questo errore è che uno o più certificati presentati dal server remoto non sono presenti nel certificate store dell'applicazione. Per risolvere il problema, è possibile utilizzare la proprietà RootCertFile e impostare il percorso del file CA. Se non se ne dispone, è possibile scaricarne uno da Mozilla, ad esempio:

<https://curl.haxx.se/docs/caextract.html>

Dopo aver impostato RootCertFile, l'errore precedente dovrebbe essere risolto.

OpenSSL | Carica funzioni aggiuntive

Per impostazione predefinita, Indy definisce le funzioni OpenSSL più comuni necessarie per cifrare le comunicazioni, ma a volte sono necessarie ulteriori funzioni per cifratura, firma, ecc. È possibile utilizzare il metodo `IdOpenSSLSetLoadFuncsCallback` per assegnare un callback per il caricamento dinamico di funzioni OpenSSL aggiuntive.

IdOpenSSLSetLoadFuncsCallback

```
delegate void TIdLoadSSLFuncsCallback(TIdLibHandle hIdSSL, TIdLibHandle hIdCrypto, TStringList FailedLoadList);
```

Si tratta di un tipo di procedura che funge da callback; accetta tre parametri:

- **hIdSSL**: `TIdLibHandle` - Handle alla libreria SSL caricata.
- **hIdCrypto**: `TIdLibHandle` - Handle alla libreria Crypto caricata.
- **FailedLoadList**: `TStringList` - Un elenco delle funzioni che non sono state caricate.

Lo scopo di questo callback è consentire all'utente di eseguire un'elaborazione personalizzata durante il caricamento delle funzioni OpenSSL, ad esempio registrando i caricamenti di funzioni falliti o gestendo gli errori.

IdOpenSSLSetUnLoadFuncsCallback

```
delegate void TIdUnLoadSSLFuncsCallback();
```

Funge da callback per lo scaricamento delle funzioni SSL. È utile per eseguire operazioni di pulizia quando le librerie OpenSSL vengono scaricate.

Come caricare una funzione personalizzata

Di seguito è riportato un semplice esempio di come caricare la funzione `EVP_PKEY_CTX_set_rsa_padding` utilizzando le callback.

```
// Note: Direct function pointer loading is not applicable in .NET/C#.
// Use P/Invoke or the managed OpenSSL wrapper instead.
// The following is a conceptual equivalent:

IntPtr EVP_PKEY_CTX_set_rsa_padding = IntPtr.Zero;

void DoOpenSSLLoadFuncsCallback(TIdLibHandle hIdSSL, TIdLibHandle hIdCrypto, TStringList FailedLoadList)
{
    EVP_PKEY_CTX_set_rsa_padding = LoadLibFunction(hIdCrypto, "EVP_PKEY_CTX_set_rsa_padding");
}

void DoOpenSSLUnLoadFuncsCallback()
{
    EVP_PKEY_CTX_set_rsa_padding = IntPtr.Zero;
}

IdOpenSSLSetLoadFuncsCallback(DoOpenSSLLoadFuncsCallback);
IdOpenSSLSetUnLoadFuncsCallback(DoOpenSSLUnLoadFuncsCallback);
```

Eventi WebSocket

Le connessioni WebSocket dispongono dei seguenti eventi:

OnConnect

L'evento generato quando viene stabilita una nuova connessione.

OnDisconnect

L'evento generato quando una connessione viene chiusa.

OnError

L'evento generato quando una connessione presenta qualsiasi errore.

OnMessage

L'evento generato quando viene ricevuto un nuovo messaggio di testo.

OnBinary

L'evento generato quando viene ricevuto un nuovo messaggio binario.

Per impostazione predefinita, `sgcWebSockets` utilizza un meccanismo **asincrono** per generare questi eventi; quando uno di essi viene generato internamente, il messaggio viene accodato e inviato dal sistema operativo quando possibile. Questo comportamento può essere modificato tramite la proprietà **NotifyEvents**; per impostazione predefinita è selezionato **neAsynchronous**. Se si seleziona **neNoSync**, gli eventi verranno generati senza sincronizzarsi con il thread principale (se è necessario aggiornare un controllo VCL o accedere a risorse condivise, sarà necessario implementare un proprio metodo di sincronizzazione).

neNoSync è consigliato quando:

1. È necessario gestire un gran numero di messaggi in un periodo di tempo molto breve.
2. Il progetto è compilato per la riga di comando (se non si imposta `neNoSync`, non si riceverà alcun evento).
3. Il progetto è una libreria.

In caso contrario, è possibile utilizzare il valore predefinito della proprietà **neAsynchronous**.

WebSocket Parameters Connection

Supportato da

[TsgcWebSocketClient](#)
Javascript

A volte è utile passare parametri dal client al server quando viene stabilita una nuova connessione WebSocket. Se è necessario passare alcuni parametri al server, è possibile utilizzare la seguente proprietà:

Opzioni / Parametri

Per impostazione predefinita, è impostato su '/', se è necessario passare un parametro come id=1, è possibile impostare questa proprietà su '/?id=1'

Sul lato server, è possibile gestire i parametri del client utilizzando il seguente parametro:

```
public void WSServerConnect(TsgcWSConnection Connection)
{
    if (Connection.URL == "/?id=1")
    {
        HandleThisParameter;
    }
}
```

Usando Javascript, è possibile passare parametri tramite l'URL di connessione, ad esempio:

```
<script src="http://localhost/sgcWebSockets.js" type="text/javascript"></script>
<script type="text/javascript">var socket = new sgcWebSocket('ws://localhost/?id=1');</script>
```

Utilizzo all'interno di una DLL

Se è necessario lavorare con Dynamic Link Library (DLL) e sgcWebSockets (o applicazioni console), la proprietà **NotifyEvents** deve essere impostata su **neNoSync**.

Test del browser Web

TsgcWebSocketServer implementa una pagina Web integrata che consente di testare la connessione al Server WebSocket con il browser Web preferito.

Per accedere a questa pagina di test, digitare il seguente URL:

```
http://host:port/sgcWebSockets.html
```

Esempio: se il server WebSocket è configurato sull'IP 127.0.0.1 e utilizza la porta 80, è necessario digitare:

```
http://127.0.0.1:80/sgcWebSockets.html
```

In questa pagina è possibile testare i seguenti metodi WebSocket:

Open
Chiudi
Stato
Send

Per disabilitare le pagine di test HTML del WebBrowser, impostare semplicemente
`TsgcWebSocketServer.Options.HTMLFiles = false;`

Autenticazione

Supportato da

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)
[TsgcWebSocketClient](#)

Java script (*è supportata solo l'autenticazione URL)

La specifica WebSocket non definisce alcun metodo di autenticazione e le implementazioni dei browser web non consentono l'invio di intestazioni personalizzate sulle nuove connessioni WebSocket.

Per abilitare questa funzionalità è necessario accedere alla seguente proprietà:

Authentication/ Enabled

sgcWebSockets implementa 3 diversi tipi di autenticazione WebSocket:

Sessione: il client deve effettuare una richiesta HTTP GET passando un nome utente e una password; se autenticato, il server risponde con un ID di sessione. Con questo ID di sessione, il client apre una connessione WebSocket passandolo come parametro. È possibile utilizzare una normale richiesta HTTP per ottenere un ID di sessione passando nome utente e password come parametri

```
http://host:port/sgc/req/auth/session/:user/:password
```

esempio: (user=admin, password=1234) --> http://localhost/sgc/req/auth/session/admin/1234

Questo restituisce un token che viene utilizzato per connettersi al server tramite connessioni WebSocket:

```
ws://localhost/sgc/auth/session/:token
```

URL: il client apre una connessione WebSocket passando nome utente e password come parametri.

```
ws://host:port/sgc/auth/url/username/password
```

esempio: (user=admin, password=1234) --> http://localhost/sgc/auth/url/admin/1234

Basic: implementa l'autenticazione di accesso Basic, si applica solo ai WebSocket VCL (server e client) e alle richieste HTTP (i browser Web non implementano questo tipo di autenticazione). Quando un client tenta di connettersi, invia un'intestazione utilizzando la specifica AUTH BASIC.

È possibile definire un elenco di utenti autenticati utilizzando la proprietà **Authentication/ AuthUsers**. È necessario definire ogni elemento seguendo questo schema: utente=password. Esempio:

```
admin=admin
user=1234
....
```

Esiste un evento chiamato **OnAuthentication** dove è possibile gestire l'autenticazione nel caso in cui l'utente non sia nella lista AuthUsers, il client non invia una richiesta di autorizzazione... È possibile verificare i parametri User e Password e, se corretti, impostare la variabile Authenticated su True. Esempio:

```
private void OnAuthenticationEvent(TsgcWSConnection Connection, string User, string Password, ref bool Authenticated)
{
    if ((User == "user") && (Password == "1234"))
```

```
{  
  Authenticated = true;  
}
```


Connessioni sicure

Supportato da

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)
[TsgcWebSocketClient](#)
 Browser web

Il supporto SSL è basato sull'implementazione di Indy, quindi è necessario distribuire le librerie openssl per utilizzare questa funzionalità.

TsgcWebSocketClient supporta Microsoft SChannel, quindi non è necessario distribuire le librerie openssl per Windows a 32 e 64 bit se l'opzione SChannel è selezionata nel WebSocket Client.

Lato server

Per abilitare questa funzionalità, è necessario abilitare la seguente proprietà:

SSL/ Abilita

Ci sono altre proprietà che è necessario definire:

SSLOptions/ CertFile/ KeyFile/ RootCertFile: è necessario un certificato in formato .PEM per cifrare le comunicazioni WebSocket.

SSLOptions/ Password: questo è facoltativo ed è necessario solo se il certificato ha una password.

SSLOptions/ Port: porta utilizzata sulle connessioni SSL.

Lato client

Per abilitare questa funzionalità, è necessario abilitare la seguente proprietà:

TLS/ Abilita

OpenSSL

Per impostazione predefinita, i componenti client e server basati su Indy utilizzano le librerie openssl per connettersi a server WebSocket sicuri.

Indy supporta solo l'API OpenSSL 1.0.2, pertanto l'API 1.1 non è supportata. Se si compila sgcWebSockets con la nostra libreria Indy personalizzata è possibile utilizzare l'API 1.1 e selezionare la versione TLS 1.3. Basta selezionare nelle proprietà OpenSSL_Options quale API OpenSSL si desidera utilizzare:

- **osIAPI_1_0:** questa è l'API Indy predefinita; è possibile utilizzare il pacchetto Indy standard con le librerie openssl 1.0.2.
- **osIAPI_1_1:** selezionare solo se si compila sgcWebSockets con la nostra libreria Indy personalizzata (Enterprise Edition). Utilizzerà le librerie OpenSSL 1.1.1.
- **osIAPI_3_0:** selezionare solo se si sta compilando sgcWebSockets con la nostra libreria Indy personalizzata (Enterprise Edition). Utilizzerà le librerie openssl 3.0.0.
 - **ECDHE:** consente di abilitare ECDHE per TLS 1.2 (connessioni più sicure).

Microsoft SChannel

Da sgcWebSockets 4.2.6 è possibile utilizzare SChannel al posto di openssl (solo per Windows da Windows 7+). Ciò significa che non è necessario distribuire le librerie openssl. TLS 1.0 è supportato da Windows 7, ma se si necessita di implementazioni più moderne come TLS 1.2 su Windows 7 è necessario abilitare TLS 1.1 e TLS 1.2 nel Registro di sistema di Windows.

HeartBeat

Supportato da

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)
[TsgcWebSocketServer_HTTPAPI](#)
[TsgcWebSocketClient](#)

Nei componenti server, invia automaticamente un ping a tutte le connessioni WebSocket attive ogni x secondi.

Sui componenti client, invia automaticamente un ping al server ogni x secondi.

HeartBeat ha le seguenti proprietà:

- **Enabled:** se true, invia un ping
- **Interval:** è il valore in secondi al quale verrà inviato un ping. Esempio: se il valore è 10, un ping verrà inviato ogni 10 secondi
- **Timeout:** è il tempo di attesa per una risposta dal server. Esempio: se il valore è 30, attenderà 30 secondi per ricevere una risposta prima di chiudere la connessione.
- **HeartBeatType:** consente di personalizzare il funzionamento del HeartBeat
 - **hbtAlways:** invia un ping ogni x secondi definiti nell'Interval.
 - **hbtOnlyIfNoMsgRcvInterval:** invia un ping ogni x secondi solo se non sono stati ricevuti messaggi durante gli ultimi x secondi definiti nella proprietà Interval.

Personalizza HeartBeat

I componenti client e server consentono la personalizzazione di HeartBeat per inviare ping personalizzati e verificare che la connessione sia ancora attiva. L'evento **OnBeforeHeartBeat** è stato creato esattamente per questo scopo; consente di inviare un messaggio personalizzato e/o di non inviare il ping standard.

Esempio: invia un testo di messaggio come ping ogni 30 secondi.

```
void OnBeforeHeartBeat(TObject Sender; const TsgcWSCConnection Connection; ref bool Handled)
{
    Connection.WriteData("ping");
    Handled = true;
}
```

WatchDog

Supportato da

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)
[TsgcWebSocketServer_HTTPAPI](#)
[TsgcWebSocketClient](#)

Server

Nei componenti server, riavvia automaticamente il server dopo un arresto imprevisto. Per controllare se il server è attivo ogni 60 secondi, basta impostare le seguenti proprietà:

```
WatchDog.Enabled = true;  
WatchDog.Interval = 60;  
WatchDog.Attempts = 0;
```

WatchDog.Monitor Le consente di verificare se nuovi client possono connettersi al server. Ciò avviene tramite un client interno che tenta di aprire una connessione WebSocket al server; se fallisce, riavvia il server. Per monitorare se i client possono connettersi al server con un time-out di 10 secondi, imposti le seguenti proprietà:

```
WatchDog.Enabled = true;  
WatchDog.Interval = 60;  
WatchDog.Attempts = 0;  
WatchDog.Monitor.Enabled = true;  
WatchDog.Monitor.TimeOut = 10;
```

Client

Sui componenti client, riconnette automaticamente al server dopo una disconnessione imprevista. Per riconnettere dopo una disconnessione ogni 10 secondi, basta impostare le seguenti proprietà:

```
WatchDog.Enabled = true;  
WatchDog.Interval = 10;  
WatchDog.Attempts = 0;
```

Log

Supportato da

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)
[TsgcWebSocketClient](#)

Questa è una funzionalità utile che consente il debug delle connessioni WebSocket; per abilitarla, è necessario accedere alla seguente proprietà:

LogFile/ Abilitato

Una volta abilitata, ogni volta che viene stabilita una nuova connessione, questa verrà registrata in un file di testo. Sul componente server, se il file non esiste verrà creato, ma non sarà possibile accedervi finché il server non viene chiuso. Se si desidera aprire il file di log mentre il server è attivo, il file di log deve essere creato prima di avviare il server.

Esempio:

```
127.0.0.1:49854 Stat Connected.

127.0.0.1:49854 Recv 09/11/2013 11:17:03: GET / HTTP/1.1
Upgrade: websocket
Connection: Upgrade
Host: 127.0.0.1:5414
Origin: http://127.0.0.1:5414
Pragma: no-cache
Cache-Control: no-cache
Sec-WebSocket-Key: 1n598ldHs9SdRfxUK8u4Vw==
Sec-WebSocket-Version: 13
Sec-WebSocket-Extensions: x-webkit-deflate-frame

127.0.0.1:49854 Sent 09/11/2013 11:17:03: HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: gDuzFRzwHBc18P1CfinlvKv1BJc=

127.0.0.1:49854 Stat Disconnected.
0.0.0.0:0 Stat Disconnected.
```

Messaggi WebSocket

I frame WebSocket possono essere mascherati, il che significa che il messaggio registrato non può essere letto. Quando la proprietà **LogFile.UnMaskFrames** = True (per impostazione predefinita è true)

- I messaggi **inviati** dal **WebSocket Client** vengono salvati come **non mascherati**.
- I messaggi **ricevuti** dal **WebSocket Server** vengono salvati **mascherati** e **non mascherati** (il motivo è che quando il socket legge il buffer, non conosce ancora il protocollo del messaggio, quindi salva entrambi).

HTTP

Supportato da

[TsgcWebSocketHTTPServer](#)

TsgcWebSocketHTTPServer è un componente che consente di gestire connessioni WebSocket e HTTP sulla stessa porta. È molto utile quando è necessario configurare un server in cui è abilitata solo la porta HTTP (solitamente la porta 80). Questo componente supporta tutte le funzionalità di [TsgcWebSocketServer](#) e consente di servire pagine HTML.

È possibile **servire pagine HTML staticamente**, utilizzando la proprietà **DocumentRoot**; ad esempio: se si salva test.html nella directory "C:\inetpub\wwwroot" e si imposta **DocumentRoot** su "C:\inetpub\wwwroot", se un client tenta di accedere a test.html, verrà servito automaticamente, ad esempio:

`http://localhost/test.html`

Oppure è possibile **servire HTML o altre risorse dinamicamente** tramite codice; a tale scopo è disponibile un evento denominato **OnCommandGet** che viene attivato ogni volta che un client richiede una nuova pagina HTML, immagine, file javascript... In sostanza, è necessario verificare quale documento sta richiedendo il client (tramite `ARequestInfo.Document`) e inviare una risposta al client (tramite `AResponseInfo.ContentText` per il contenuto della risposta, `AResponse.ContentType` per il tipo di risposta e `AResponseInfo.ResponseNo` con il numero del codice di risposta, di norma 200). Esempio:

```
private void OnCommandGetEvent(TsgcWSConnection Connection, TsgcWSHTTPRequestInfo RequestInfo,
    ref TsgcWSHTTPResponseInfo ResponseInfo)
{
    if (RequestInfo.Document == "/myfile.js")
    {
        ResponseInfo.ContentText = "<script type='text/javascript'>alert('Hello!');</script>";
        ResponseInfo.ContentType = "text/javascript";
        ResponseInfo.ResponseNo = 200;
    }
}
```

Broadcast e canali

Supportato da

[TsgcWebSocketServer](#)

[TsgcWebSocketHTTPServer](#)

[TsgcWebSocketServer_HTTPAPI](#)

Il metodo **Broadcast** per impostazione predefinita invia un messaggio a **tutti i client connessi**, ma Lei può usare l'argomento **channels** per filtrare e **inviare il messaggio in broadcast solo ai client iscritti** a un canale.

Esempio: il Suo server ha 2 tipi di client connessi, dispositivi desktop e mobili, quindi Lei può creare 2 canali "desktop" e "mobile".

Se Lei può identificare nell'evento OnConnect del server se un client è mobile, può fare qualcosa di simile a quanto segue:

```
void OnServerConnect(TsgcWSConnection Connection)
{
    if (desktop == true)
    {
        (TsgcWSConnectionServer)(Connection).Subscribe("desktop");
    }
}
```

Per prima cosa esegua il cast di Connection a TsgcWSConnectionServer per accedere ai metodi di sottoscrizione e, se corrisponde al Suo filtro, verrà iscritto al canale desktop. La sottoscrizione a un canale può essere effettuata in qualsiasi evento. Ad esempio, Lei può chiedere al client di comunicarle se è mobile o meno e inviare un messaggio dal client al server con le informazioni sul client. Quindi Lei può inviare in broadcast solo alle connessioni desktop:

```
Server.Broadcast("Your text message", "desktop");
```

Se Lei ha 100 connessioni e 30 sono mobili, il messaggio verrà inviato solo alle altre 70.

Bindings

Supportato da

[TsgcWebSocketServer](#)

[TsgcWebSocketHTTPServer](#)

Di norma, i server hanno più di un IP. Se si abilita un server WebSocket e si imposta la porta di ascolto su 80, all'avvio del server esso tenta di mettersi in ascolto sulla porta 80 di TUTTI gli IP. Quindi, se si dispone di 3 IP, si legherà la porta 80 su ciascuno di essi.

I binding consentono di definire esattamente quale IP e porta vengono utilizzati dal Server. Esempio: se si desidera ascoltare sulla porta 80 per l'IP 127.0.0.1 (indirizzo interno) e 80.254.21.11 (indirizzo pubblico), è possibile farlo prima che il server venga attivato:

```
WSServer.Bindings = "127.0.0.1:80,80.254.21.11:80";
```


Inviare file di grandi dimensioni tramite POST

Supportato da

[TsgcWebSocketHTTPServer](#)

[TsgcWebSocketServer_HTTPAPI](#)

Quando un client HTTP invia uno stream **multipart/form-data**, lo stream viene salvato dal server in memoria. Quando i file sono di grandi dimensioni, il server può generare un'eccezione **out of memory**. Per evitare queste eccezioni, il server dispone di una proprietà chiamata **HTTPUploadFiles** con cui è possibile configurare come gli stream POST vengono gestiti: in memoria o come file stream. Se gli stream vengono gestiti come file stream, gli stream ricevuti vengono memorizzati direttamente sul disco rigido, evitando così i problemi di memoria.

Per configurare il Suo server in modo che salvi gli stream multipart/form-data come file stream, segua i passaggi successivi:

1. Imposti la proprietà **HTTPUploadFiles.StreamType = pstFileStream**. Con questa configurazione, il server memorizzerà questi stream sul disco rigido.
2. È possibile configurare quale sia la **dimensione minima in byte** a partire dalla quale i file verranno memorizzati come file stream. Per impostazione predefinita il valore è zero, il che significa che tutti gli stream verranno memorizzati come file stream.
3. La cartella in cui gli stream vengono memorizzati si indica tramite **SaveDirectory**; se non impostata, verranno memorizzati nella stessa cartella in cui si trova l'applicazione.
4. Quando un client invia un multipart/form-data, il contenuto è codificato all'interno di boundary; se la proprietà **RemoveBoundaries** è abilitata, il contenuto dei boundary verrà estratto automaticamente dopo che lo stream completo è stato ricevuto.

Codice di esempio

Per prima cosa crei una nuova istanza del server e imposti che gli Stream vengano salvati come File Stream.

```
TsgcWebSocketHTTPServer oServer = new TsgcWebSocketHTTPServer();
oServer.Port = 5555;
oServer.HTTPUploadFiles.StreamType = TwsPostStreamType.pstFileStream;
oServer.Active = true;
```

Quindi crei un nuovo file html con la seguente configurazione

```
<html>
<head><title>sgcWebSockets - Upload Big File</title></head>
<body>
<form action="http://127.0.0.1:5555/file" method="post" enctype="multipart/form-data"
accept-charset="UTF-8">
<input type="file" name="file_1" />
<input type="submit" />
</form>
</body>
</html>
```

Infine apra il file html con un browser web e invii un file al server. Il server creerà un nuovo file stream con l'estensione ".sgc_ps" e, quando lo stream sarà stato ricevuto completamente, estrarrà il file dai boundary.

Eventi

Esistono 2 eventi che possono essere utilizzati per personalizzare il flusso di caricamento dei file (richiede che la proprietà **HTTPUploadFiles.RemoveBoundaries** sia abilitata)

OnHTTPUploadBeforeSaveFile

Questo evento viene generato PRIMA che il file venga salvato e consente di personalizzare il nome del file ricevuto.

```
private void OnHTTPUploadBeforeSaveFileEvent(TObject Sender, ref string aFileName, ref string aFilePath)
{
    if (aFileName == "test.jpg")
    {
        aFileName = "custom_test.jpg";
    }
}
```

OnHTTPUploadAfterSaveFile

Questo evento viene generato DOPO che il file è stato salvato e Le consente di conoscere il nome del file salvato.

```
private void OnHTTPUploadBeforeSaveFileEvent(TObject Sender, string aFileName, string aFilePath)
{
    DoLog("File Received: " + aFileName);
}
```

OnHTTPUploadReadInput

Questo evento viene generato quando il decoder legge un valore di input ricevuto diverso dall'input file (esempio: se il form contiene alcune variabili come nome, data...).

```
private void OnHTTPUploadReadInputEvent(TObject Sender, string aName, string aValue)
{
    DoLog("Input value Received: " + aName + ":" + aValue);
}
```

Compressione

Supportato da

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)
[TsgcWebSocketClient](#)
Browser Web come Chrome

Questa è una funzionalità che funziona molto bene quando ha bisogno di inviare una grande quantità di dati, di solito utilizzando un messaggio binario, perché comprime il messaggio WebSocket utilizzando il protocollo "PerMessage_Deflate", che è supportato da alcuni browser come Chrome.

Per abilitare questa funzionalità, è necessario attivare la seguente proprietà:

Extensions/ PerMessage_Deflate / Enabled

Quando un client tenta di connettersi a un server WebSocket e questa proprietà è abilitata, invia un'intestazione con questa proprietà abilitata. Se il server ha attivato questa funzionalità, invia una risposta al client con questo protocollo attivato e tutti i messaggi saranno compressi. Se il server non dispone di questa funzionalità, tutti i messaggi vengono inviati senza compressione.

Nei browser web non è necessario fare nulla, se questa estensione è supportata verrà utilizzata automaticamente, altrimenti i messaggi verranno inviati senza compressione.

Se i messaggi WebSocket sono piccoli, è preferibile non abilitare questa proprietà perché consuma cicli CPU per comprimere/decomprimere i messaggi. Tuttavia, se si trasferisce una grande quantità di dati, si noterà un aumento della velocità di scambio dei messaggi.

Flash

Supportato da

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)

I WebSocket sono supportati nativamente da un'ampia gamma di browser web (si prega di consultare <http://caniuse.com/websockets>), ma esistono alcune versioni precedenti che non implementano i WebSocket (come Internet Explorer 6, 7, 8 o 9). È possibile abilitare il **Fallback Flash** per tutti questi browser che non implementano i WebSocket.

Quasi tutti gli altri browser o i browser più vecchi supportano Flash installando Adobe Flash Player. Per supportare la connessione Flash, è necessario **aprire la porta 843** sul proprio server poiché Flash utilizza questa porta per motivi di sicurezza al fine di verificare l'accesso cross-domain. Se la porta 843 non è raggiungibile, attende 3 secondi e tenta di connettersi alla porta predefinita del server.

Flash viene applicato solo se il browser non supporta WebSocket nativamente. Pertanto, se si abilita il fallback Flash lato server e il browser supporta WebSocket nativamente, utilizzerà comunque WebSocket come trasporto.

Per abilitare il Flash Fallback, è necessario accedere alla proprietà **FallBack / Flash** sul server e **abilitarla**. Sono disponibili altre 2 proprietà:

1. Domain: se necessita di limitare le connessioni flash a uno o più domini (per impostazione predefinita tutti i domini sono consentiti). Esempio: questo consentirà l'accesso al dominio `swf.example.com`

`swf.example.com`

2. Porte: se è necessario limitare le connessioni flash a una o più porte (per impostazione predefinita sono consentite tutte le porte). Esempio: questo consentirà l'accesso alle porte 123, 456, 457 e 458

`123,456-458`

Le connessioni Flash supportano solo messaggi di testo, i messaggi binari non sono supportati.

Gruppi

Supportato da

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)
[TsgcWebSocketServer_HTTPAPI](#)

sgcWebSockets fornisce un metodo potente per **trasmettere messaggi a sottoinsiemi specifici di client connessi**. Un gruppo può avere un numero qualsiasi di client e un client può far parte di un numero qualsiasi di gruppi. Non è necessario creare esplicitamente i gruppi. Di fatto, un gruppo viene creato automaticamente la prima volta che si specifica il suo nome in una chiamata a `Groups.Add`.

Quando si aggiunge un utente a un gruppo utilizzando il metodo **Groups.Add**, l'utente riceve i messaggi indirizzati a quel gruppo per la durata della connessione corrente.

Aggiunta e rimozione di utenti

Per aggiungere o rimuovere utenti da un gruppo, chiamare i metodi `Add` o `Remove` passando il nome del gruppo e la classe `TsgcWSConnection`. Non è necessario rimuovere manualmente un utente da un gruppo quando la connessione termina.

L'esempio seguente mostra il metodo `Groups.Add`.

```
void OnConnect(TsgcWSConnection Connection)
{
    TsgcWebSocketServer1.Groups.Add("Room1", Connection);
}
```

Invio di Messaggi a un Gruppo

È possibile inviare un messaggio a tutti i membri di un gruppo come mostrato nell'esempio seguente.

```
TsgcWebSocketServer1.Groups.Group["Room1"].Broadcast("Hello Members of Room1");
```

Oppure è possibile inviare un messaggio a tutti i gruppi che iniziano con "Room" (quindi se esistono Room1, Room2, Room3... questi utenti riceveranno un messaggio).

```
TsgcWebSocketServer1.Groups.Broadcast("Room*", "Hello Members of Room");
```

Eventi

Esistono 2 eventi che possono essere utilizzati per gestire i Gruppi e i Client ogni volta che un nuovo client viene aggiunto a un gruppo o quando uno viene rimosso:

[OnClientAdded](#)
[OnClientRemoved](#)

Esempio: inviare un messaggio al gruppo quando un membro lascia il gruppo.

```
TsgcWebSocketServer1.Groups.OnClientRemoved() = OnClientRemovedEvent();  
  
void OnClientRemovedEvent(TObject Sender, const TsgcWSServerGroupItem aGroup,  
    const TsgcWSConnection aConnection)  
{  
    aGroup.Broadcast("Client " + aConnection.Guid + " has disconnected");  
}
```

IOCP

Supportato da

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)

IOCP per Windows è un'API che consente di gestire migliaia di connessioni utilizzando un pool limitato di thread invece di usare un thread per connessione come fa Indy per impostazione predefinita. Per abilitare IOCP per i Server Indy, vada alla proprietà **IOHandlerOptions** e selezioni **iohIOCP** come tipo di IO-Handler.

```
Server.IOHandlerOptions.IOHandlerType = iohIOCP;  
Server.IOHandlerOptions.IOCP.IOCPThreads = 0;  
Server.IOHandlerOptions.IOCP.WorkOpThreads = 0;
```

IOCPThreads sono i thread utilizzati per le richieste asincrone IOCP (operazioni overlapped). Per impostazione predefinita il valore è zero, il che significa che il numero di thread viene calcolato in base al numero di processori (eccetto per Delphi 7 e 2007, dove il numero di thread è impostato a 32 perché la funzione `cpucount` non è supportata).

WorkOpThreads deve essere abilitato solo se si desidera che le connessioni vengano sempre elaborate nello stesso thread. Quando si utilizza IOCP, le richieste vengono elaborate da un pool di thread e ogni richiesta (per la stessa connessione) può essere elaborata in thread diversi. Se si desidera gestire ogni connessione nello stesso thread, impostare in **WorkOpThreads** il numero di thread utilizzati per gestire tali richieste. Ciò influisce sulle prestazioni del server ed è consigliabile impostare un valore maggiore di zero solo se questa funzionalità è necessaria.

L'abilitazione di IOCP per i server windows è consigliata quando è necessario gestire migliaia di connessioni. Se il Suo server gestisce solo un massimo di 100 connessioni concorrenti, può rimanere con il modello a thread Indy predefinito.

Evento OnDisconnect non attivato

IOCP funziona in modo diverso dal gestore IOHandler predefinito di Indy. Con il gestore IOHandler predefinito di Indy, ogni connessione viene eseguita in un thread e questi thread sono sempre attivi, verificando se la connessione è attiva, quindi in caso di disconnessione viene notificata in breve tempo.

IOCP funziona in modo diverso: dispone di un pool di thread che gestisce tutte le connessioni, invece di 1 thread = 1 connessione come fa Indy per impostazione predefinita. Per IOCP, l'unico modo per rilevare se una connessione è ancora attiva è tentare di scrivere sul socket. In caso di errore, significa che la connessione è chiusa. Esistono 2 opzioni per rilevare le disconnessioni:

1. Se si utilizza **TsgcWebSocketClient**, è possibile abilitarlo nella proprietà Options, **CleanDisconnect := True** (disabilitato per impostazione predefinita). Se abilitato, prima della disconnessione il client invia un messaggio al server per informarlo della disconnessione, in modo che il server riceva questo messaggio e venga generato l'evento **OnDisconnect**.
2. È possibile abilitare l'**heartbeat** sul lato **server**, ad esempio ogni 60 secondi, in modo che tenti di inviare un ping a tutti i client connessi e, se un client si è disconnesso, verrà chiamato **OnDisconnect**.

ALPN

Supportato da

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)
[TsgcWebSocketClient](#)

Application-Layer Protocol Negotiation (ALPN) è un'estensione Transport Layer Security (TLS) per la negoziazione del protocollo a livello applicazione. ALPN consente al livello applicazione di negoziare quale protocollo deve essere eseguito su una connessione sicura in modo da evitare round trip aggiuntivi e che è indipendente dai protocolli del livello applicazione. È necessario per le connessioni HTTP/2 sicure, che migliorano la compressione delle pagine web e ne riducono la latenza rispetto a HTTP/1.x.

Client

È possibile configurare in `TLSOptions.ALPNProtocols` quali protocolli sono supportati dal client. Quando il client si connette al server, questi protocolli vengono inviati nell'handshake TLS iniziale 'Client Hello', elencando i protocolli supportati dal client, e il server seleziona il protocollo da utilizzare, se disponibile.

È possibile sapere quale protocollo è stato selezionato dal server accedendo alla proprietà `ALPNProtocol` di `TsgcWSConnectionClient`.

Server

When there is a new TLS connection, `OnSSLALPNSelect` event is called, here you can access to a list of protocols which are supported by client and server can select which of them is supported.

If there is no support for any protocol, `aProtocol` can be left empty.

```
// Client
void OnClientConnect(TsgcWSConnection Connection)
{
    string vProtocol = ((TsgcWSConnectionClient)Connection).ALPNProtocol;
}

// Server
void OnSSLALPNSelect(string Protocols, ref string Protocol)
{
    if (Array.IndexOf(Protocols.Split(','), "h2") >= 0)
    {
        Protocol = "h2";
    }
}
```


Inoltra richieste HTTP

Supportato da

[TsgcWebSocketHTTPServer](#)
[TsgcWebSocketServer_HTTPAPI](#)
[TsgcWSHTTPWebBrokerBridgeServer](#)
[TsgcWSHTTP2WebBrokerBridgeServer](#)
[TsgcWSServer_HTTPAPI_WebBrokerBridge](#)

È possibile configurare il server per inoltrare alcune richieste HTTP a un altro server. Ciò è molto utile quando si dispone di più server e solo uno è in ascolto su un indirizzo pubblico.

Esempio: può configurare il Suo server in modo da inoltrare a un altro server tutte le richieste verso `/internal`, mentre tutte le altre richieste vengono gestite dal server `sgcWebSockets`.

Utilizzare l'evento **OnBeforeForwardHTTP** per verificare se l'URL richiesto deve essere inoltrato e, in tal caso, impostare l'URL di inoltra.

Esempio: se desidera inoltrare tutte le richieste per il documento `/internal` al server `localhost:8080`, proceda come segue:

```
void OnBeforeForwardHTTP(TsgcWSConnection Connection, TsgcWSHTTPRequestInfo ARequestInfo,
    ref TsgcWSServerForwardHTTP aForward)
{
    if (ARequestInfo.Document == "/internal")
    {
        aForward.Enabled = true;
        aForward.URL = "http://localhost:8080";
    }
}
```

Altre opzioni

Quando si desidera inoltrare una richiesta HTTP, sono disponibili le seguenti opzioni aggiuntive:

1. Per impostazione predefinita, la richiesta viene inoltrata utilizzando il documento originale. **Esempio:** se si inoltra la richiesta `http://localhost:8080/internal` al server interno `http://localhost:5555`, l'URL inoltrato sarà `http://localhost:5555/internal`. È tuttavia possibile modificare il documento utilizzando la proprietà **Document** dell'oggetto **Forward** (per impostazione predefinita verrà utilizzato lo stesso della richiesta originale).

```
aForward.Document = "/NewInternal"
```

2. Se si reindirizza una connessione HTTP sicura (HTTPS), è possibile personalizzare le opzioni SSL/TLS nella proprietà **TLSTOptions** dell'oggetto **Forward**. **Esempio:** impostare la versione TLS

```
aForward.TLSTOptions.Version = tls1_2
```

3. Le seguenti proprietà possono essere utilizzate per personalizzare la richiesta HTTP:

- **QueryParams:** i parametri dopo il documento, ad esempio: `'id=1&user=2'`.
- **Host:** specifica l'host e il numero di porta del server a cui viene inviata la richiesta. Esempio: `www.esegece.com:443`
- **Origin:** l'origine (schema, nome host e porta) che ha causato la richiesta. Esempio: `https://www.esegece.com/document`.
- **LogFilename:** il nome del file in cui verranno memorizzate la richiesta e la risposta.
- **NoCache:** se la richiesta non deve utilizzare la cache del browser web; per impostazione predefinita è abilitato.
- **CustomHeaders:** un elenco di intestazioni personalizzate da aggiungere alla richiesta. Esempio: `CustomHeaders.Add('X-ReverseProxy-Host: http://127.0.0.1:8888/test');`

Code

Supportato da

[TsgcWSPServer_sgc](#)

[TsgcWSPClient_sgc](#)

Javascript

[SGC Default Protocol](#) implementa le Queue per aggiungere persistenza ai messaggi pubblicati (è disponibile solo per i **Published messages**)

Livello 0: I messaggi non vengono accodati sul Server

Livello 1: solo l'ultimo messaggio viene accodato sul Server e viene inviato ogni volta che un client si sottoscrive a un nuovo canale o si connette al server.

Livello 2: Tutti i messaggi sono in coda sul server e vengono inviati ogni volta che un client si iscrive a un nuovo canale o si connette al server.

Livello 0

Il messaggio non viene accodato dal Server

La tabella seguente mostra il flusso del protocollo Queue di livello 0.

Client	Messaggio e direzione	Server
Queue = 0	PUBLISH ----->	Azione: Pubblica un messaggio agli iscritti

Livello 1

Un messaggio con livello Queue 1 viene memorizzato sul server e se ci sono altri messaggi memorizzati per questo canale, vengono eliminati.

La tabella seguente mostra il flusso del protocollo Queue di livello 1.

Client	Messaggio e direzione	Server
Queue = 1	PUBLISH ----->	Azioni: • Elimina tutti i messaggi di questo canale • Memorizza l'ultimo messaggio per canale
Azione: Elabora messaggio	NOTIFY <-----	Azione: Ogni volta che un nuovo client si sottoscrive a questo canale, viene inviato l'ultimo messaggio.

Ciò è utile quando i publisher inviano messaggi su base "segnalazione per eccezione", dove potrebbe passare del tempo tra un messaggio e l'altro. Questo consente ai nuovi subscriber di ricevere immediatamente i dati con il valore mantenuto, o Ultimo Valore Noto.

Livello 2

Tutti i messaggi con Queue di livello 2 vengono memorizzati sul server.

La tabella seguente mostra il flusso del protocollo Queue di livello 2.

Client	Messaggio e direzione	Server
Queue = 2	PUBLISH ----->	Azione: Memorizza il messaggio
Azione: Elabora messaggio	NOTIFY <-----	Azione: Ogni volta che un nuovo client si iscrive a questo canale, TUTTI i messaggi vengono inviati.

Transazioni

Supportato da

[TsgcWSPServer_sgc](#)

[TsgcWSPClient_sgc](#)

Javascript

Il protocollo SGC di sgcWebSockets supporta la messaggistica transazionale; quando un client esegue il commit di una transazione, tutti i messaggi inviati dal client vengono elaborati lato server. Esistono 3 metodi chiamati dal client:

StartTransaction

Crea una nuova transazione sul lato server e tutti i messaggi inviati dal client al server dopo questo metodo vengono accodati sul lato server fino a quando il client non chiama Commit o Rollback.

Client	Messaggio e direzione	Server
Channel = X	STARTTRANSACTION ----->	Action: Crea una nuova coda per memorizzare tutti i messaggi del canale specificato
Channel = X	PUBLISH ----->	Azione: Il messaggio è memorizzato lato server.
Action: Il client riceve la conferma del messaggio inviato	ACKNOWLEDGEMENT < -----	Azione: Il server restituisce un messaggio di conferma al client perché il messaggio è stato memorizzato.
....		

Commit

Quando un client chiama Commit, tutti i messaggi accodati dal server vengono elaborati.

Client	Messaggio e direzione	Server
Channel = X	COMMIT ----->	Azione: Elabora tutti i messaggi accodati dalla Transazione

RollBack

Quando un client chiama RollBack, tutti i messaggi accodati dal server vengono eliminati e non vengono elaborati lato server.

Client	Messaggio e direzione	Server

Channel = X	ROLLBACK ----->	Azione: Elimina tutti i messaggi accodati dalla Transazione

Connessioni TCP

Supportato da

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)
[TsgcWebSocketClient](#)

Per impostazione predefinita, sgcWebSocket utilizza WebSocket come protocollo, ma è possibile utilizzare il protocollo TCP semplice nei componenti client e server.

Client Componente

Disabilita il protocollo WebSocket.

```
Client.Specifications.RFC6455 = false;
```

Componente Server

Gestire l'evento OnUnknownProtocol e impostare Transport come trpTCP e accettare la connessione.

```
void OnUnknownProtocol(TsgcWSConnection Connection, ref bool Accept)
{
    Accept = true;
}
```

Quando un client si connette al server, questa connessione verrà definita come TCP e utilizzerà il protocollo TCP semplice anziché WebSockets. Le connessioni TCP semplici non distinguono tra messaggi di testo e binari, quindi tutti i messaggi ricevuti vengono gestiti dall'evento OnBinary.

Fine del messaggio

Se i messaggi sono di grandi dimensioni, possono a volte essere ricevuti in modo frammentato. Esiste un metodo per rilevare la fine di un messaggio specificando quali byte cercare. Esempio: nel protocollo STOMP, tutti i messaggi terminano con i byte 0 e 10.

```
void OnWSClientConnect(TsgcWSConnection Connection)
{
    Connection.SetTCPEndOfFrameScanBuffer(TtcpEOFFrameScanBuffer.eofScanAllBytes);
    Connection.AddTCPEndOfFrame(0);
    Connection.AddTCPEndOfFrame(10);
}
```

SubProtocol

Supportato da

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)
[TsgcWebSocketClient](#)
[TsgcWebSocketServer_HTTPAPI](#)

WebSocket fornisce una semplice negoziazione del sottoprotocollo. In pratica aggiunge un'intestazione con i nomi dei protocolli supportati dalla richiesta. Questi protocolli vengono ricevuti e, se il destinatario ne supporta uno, invia una risposta con il sottoprotocollo supportato.

sgcWebSockets supporta diversi sottoprotocolli: [MQTT](#), [WAMP](#)... e altri ancora. È possibile implementare i propri sottoprotocolli utilizzando un metodo molto semplice: basta chiamare `RegisterProtocol` e passare il nome del sottoprotocollo come argomento.

Esempio: deve connettersi a un server che implementa il sottoprotocollo "Test 1.0"

```
Client = new TsgcWebSocketClient();
Client.Host = "server host";
Client.Port = server.port;
Client.RegisterProtocol("Test 1.0");
Client.Active = true;
```

Per **utilizzare più di 1 protocollo in una singola connessione**, è possibile usare i componenti **Broker Protocol** (Server e Client) per gestirlo. Basta inserire un Broker tra il Client/Server e i protocolli. **Esempio:** Utilizzare i protocolli SGC e Files con una singola connessione.

```
// ... server
oServer = new TsgcWebSocketServer();
oServerBroker = new TsgcWSPServer_Broker();
oServerBroker.Server = oServer;
oServerSGC = new TsgcWSPServer_sgc();
oServerSGC.Broker = oServerBroker;
oServerFiles = new TsgcWSPServer_files();
oServerFiles.Broker = oServerBroker;
// ... client
oClient = new TsgcWebSocketClient();
oClientBroker = new TsgcWSPClient_Broker();
oClientBroker.Client = oClient;
oClientSGC = new TsgcWSPClient_sgc();
oClientSGC.Broker = oClientBroker;
oClientFiles = new TsgcWSPClient_files();
oClientFiles.Broker = oClientBroker;
```

Quando un protocollo broker è collegato tra il server/client e il protocollo, gli eventi **OnConnect** e **OnDisconnect** vengono generati nel componente Broker (invece che nel componente Server o Client).

Throttle

Supportato da

[TsgcWebsocketServer](#)
[TsgcWebsocketHTTPServer](#)
[TsgcWebsocketClient](#)

La limitazione della larghezza di banda è supportata dai componenti server e client; se abilitata, può limitare il numero di bit al secondo inviati/ricevuti dal socket. Indy utilizza un metodo bloccante, quindi se un client sta limitando la propria lettura, i dati non letti rimarranno all'interno del socket del client e il server verrà bloccato dalla scrittura di nuovi dati verso il client. Più lentamente il client legge i dati, più lentamente il server può scrivere nuovi dati.

Server-sent Events (Notifiche Push)

Supportato da

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)
Javascript

Gli SSE non fanno parte dei WebSocket, definiscono un'API per l'apertura di una connessione HTTP per la ricezione di notifiche push da un server.

Gli SSE vengono inviati tramite HTTP tradizionale. Ciò significa che non richiedono un protocollo speciale o un'implementazione server dedicata per funzionare. Inoltre, i Server-Sent Events dispongono di diverse funzionalità che i WebSocket non possiedono per design, come la riconnessione automatica, gli ID degli eventi e la capacità di inviare eventi arbitrari.

Eventi

- **Open:** quando viene aperta una nuova connessione SSE.
- **Messaggio:** quando il client riceve un nuovo messaggio.
- **Errore:** quando si verifica un errore di connessione come una disconnessione.

JavaScript API

Per iscriversi a un flusso di eventi, crei un oggetto EventSource e gli passi l'URL del Suo stream:

```
var sse = new EventSource('sse.html');

sse.addEventListener('message', function(e) {
  console.log(e.data);
}, false);

sse.addEventListener('open', function(e) {
  // Connection was opened.
}, false);

sse.addEventListener('error', function(e) {
  if (e.readyState == EventSource.CLOSED) {
    // Connection was closed.
  }
}, false);
```

Quando gli aggiornamenti vengono inviati dal server, il gestore onmessage si attiva e i nuovi dati sono disponibili nella sua proprietà e.data. Se la connessione viene chiusa, il browser si riconnetterà automaticamente alla sorgente dopo ~3 secondi (questo è l'intervallo di retry predefinito, modificabile lato server).

Campi

I seguenti nomi di campo sono definiti dalla specifica:

event

Il tipo dell'evento. Se specificato, nel browser viene inviato un evento al listener per il nome evento specificato; il sito web utilizzerebbe addEventListener() per ascoltare gli eventi nominati. Il gestore onmessage viene chiamato se non viene specificato alcun nome evento per un messaggio.

data

Il campo dati del messaggio. Quando EventSource riceve più righe consecutive che iniziano con data:, le concatena inserendo un carattere di nuova riga tra ciascuna. Le nuove righe finali vengono rimosse.

id

L'ID evento a cui impostare il valore dell'ultimo ID evento dell'oggetto EventSource.

riprovare

Il tempo di riconnessione da utilizzare quando si tenta di inviare l'evento. Deve essere un intero che specifica il tempo di riconnessione in millisecondi. Se viene specificato un valore non intero, il campo viene ignorato.

Tutti gli altri nomi di campo vengono ignorati.

Per le stringhe multiriga, utilizzare #10 come carattere di avanzamento riga.

Esempi di utilizzo:

Se si desidera inviare un messaggio a un client, basta usare il metodo WriteData.

```
// If you need to send a message to a client, just use WriteData method.
Connection.WriteData("Notification from server");

// To send a message to all Clients, use Broadcast method.
Connection.Broadcast("Notification from server");

//To send a message to all Clients using url 'sse.html', use Broadcast method and Channel parameter:
Connection.Broadcast("Notification from server", "/sse.html");

// You can send a unique id with an stream event by including a line starting with "id:":
Connection.WriteData("id: 1 \r data: Notification from server");

// If you need to specify an event name:
Connection.WriteData("event: notifications \r data: Notification from server");
```

javascript code to listen "notifications" channel:

```
sse.addEventListener('notifications', function(e) {
  console.log('notifications:' + e.data);
}, false);
```

LoadBalancing

Supportato da

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)
[TsgcWebSocketLoadBalancerServer](#)

Il Load Balancing consente di distribuire il lavoro tra diversi server di back-end: ogni volta che un nuovo client richiede una connessione, si collega a un server di load balancer (che è connesso ai server di back-end) e riceve una stringa di connessione contenente informazioni su host, porta... che il client utilizza per connettersi a un server. Se, ad esempio, dispone di 4 server, con questo metodo tutti i server avranno, più o meno, lo stesso numero di connessioni e il carico di lavoro sarà simile.

Se un client desidera inviare un messaggio a tutti i client di tutti i server, è sufficiente utilizzare il metodo Broadcast: il messaggio verrà trasmesso a tutti i server connessi al Load Balancer Server.

Per abilitare questa funzionalità:

1. Rilasciare un componente [TsgcWebSocketLoadBalancerServer](#), impostare una porta di ascolto e impostare Active su True.
2. I componenti Server e Client hanno una proprietà denominata LoadBalancer, in cui è necessario impostare host e porta del Server Load Balancer e impostare Enabled su True.

Il componente consente il bilanciamento del carico dei protocolli **WebSocket** e **HTTP**.

File

Supportato da

[TsgcWSPServer_sgc](#)

[TsgcWSPClient_sgc](#)

Questo protocollo consente di inviare file dal client al server e dal server al client in modo semplice. È possibile inviare file di qualsiasi dimensione con un basso utilizzo di memoria. È possibile impostare:

1. Dimensione del pacchetto in byte.
2. Utilizzare canali personalizzati per inviare file solo ai client sottoscritti.
3. L'avanzamento dell'invio e della ricezione dei file.
4. Autorizzazione dei file ricevuti.
5. Riconoscimento dei pacchetti inviati.

Proxy

Supportato da

[TsgcWebSocketClient](#)

I componenti client WebSocket supportano le connessioni WebSocket tramite proxy HTTP. Per abilitare la connessione tramite proxy, è necessario attivare le seguenti proprietà:

Proxy / Abilitato

Una volta impostato su True, è possibile configurare:

Host: Indirizzo del server proxy

Port: Porta del server proxy

UserName/Password: Autenticazione per la connessione al proxy, solo se richiesta.

ProxyType: sono supportati i seguenti proxy:

- HTTP
- Socks4
- Socks4A
- Socks5

Può configurare i proxy SOCKS accedendo alla proprietà SOCKS e impostando Enabled a True.

Messaggi Frammentati

Supportato da

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)
[TsgcWebSocketClient](#)
[TsgcWebSocketServer_HTTPAPI](#)

Per impostazione predefinita, quando uno stream viene inviato tramite la libreria `sgcWebSockets`, invia tutti i dati in un singolo pacchetto oppure memorizza nel buffer tutti i pacchetti e quando viene ricevuto l'ultimo pacchetto, viene chiamato l'evento `OnBinary message`.

Questo comportamento può essere personalizzato tramite la proprietà **Options.FramgmentedMessages**, che accetta i seguenti valori:

1. `frgOnlyBuffer`: questo è il valore predefinito. Significa che i messaggi in pacchetti verranno memorizzati nel buffer e solo quando l'intero stream viene ricevuto verrà chiamato l'evento messaggio `OnBinary`.
2. `frgOnlyFramgmented`: questo significa che l'evento `OnFramgmented` verrà chiamato solo per ogni pacchetto ricevuto.
3. `frgAll`: indica che l'evento `OnFramgmented` verrà chiamato per ogni pacchetto ricevuto e quando viene ricevuto il flusso completo.

L'evento **OnFramgmented** è utile quando è necessario inviare flussi di grandi dimensioni e il destinatario deve mostrare l'avanzamento del trasferimento.

Esempio: il client deve inviare uno stream di 1.000.000 byte al server e il server vuole mostrare l'avanzamento ogni 1000 byte ricevuti.

Il client invierà uno stream usando il metodo `writedata` con una dimensione di un pacchetto di 1000

```
Client.WriteData(stream, 1000);
```

Il server imposterà `Options.FramgmentedMessages := frgAll` e gestirà l'evento `OnFramgmented` per ricevere il progresso degli stream

```
void OnFramgmented(TsgcWSConnection Connection, TMemoryStream Data, TOpCode OpCode, boolean Continuation)
{
    ShowProgress(Data.Size);
    if (Continuation == false)
    {
        SaveStream(Data);
    }
}
```

TsgcWebSocketClient

Componente *WebSocket client* — si connette a qualsiasi server *WebSocket RFC 6455* e invia/riceve messaggi di testo e binari.

Introduzione

TsgcWebSocketClient implementa il componente *WebSocket client* e può connettersi a un server *WebSocket*. Segua i passaggi seguenti per configurare questo componente:

1. Crei un'istanza di **TsgcWebSocketClient** nel codice (`new TsgcWebSocketClient()`).
2. Imposti **Host** e **Port** (il valore predefinito è 80) per connettersi a un server *WebSocket* disponibile. Può impostare la proprietà **URL** e **Host**, **Port**, **Parameters**... verranno aggiornati a partire da **URL**. **Esempio:** `wss://127.0.0.1:8080/ws/` produrrà:

```
oClient = new TsgcWebSocketClient();
oClient.Host = "127.0.0.1";
oClient.Port = 80;
oClient.TLS = true;
oClient.Options.Parameters = "/ws/";
```

3. Può selezionare se richiedere **TLS** (connessione sicura) o meno; per impostazione predefinita, non è attivato.

4. È possibile connettersi tramite un server *HTTP Proxy*; è necessario definire le proprietà del proxy:

Host: nome host del server proxy.

Port: numero di porta del server proxy.

Username: nome utente per l'autenticazione, lasciare vuoto per l'accesso anonimo.

Password: password per l'autenticazione, lasciare vuoto per l'accesso anonimo.

5. Se il server supporta la **compressione**, è possibile abilitare la compressione per comprimere i messaggi inviati.

6. Imposti le **Specifications** consentite; per impostazione predefinita, tutte le specifiche sono abilitate.

RFC6455: è la specifica *WebSocket* standard e raccomandata.

Hixie76: è sempre false

7. Imposti `Active = true` (o chiami `Connect()`) per avviare una nuova connessione *WebSocket*.

Riferimento

- Proprietà — 23 proprietà disponibili sul componente.
- Metodi — 22 metodi pubblici.
- Eventi — 16 eventi che può gestire.

Guide

Usi più comuni

- Connessione
 - [Come connettersi a un server WebSocket](#)
 - [Aprire una connessione client](#)
 - [Chiudere una connessione client](#)
 - [Mantenere la connessione attiva](#)
 - [Disconnessioni interrotte](#)
 - [Connettere un server TCP](#)
 - [Reindirizzamenti WebSocket](#)
- Server sicuri
 - [Connettersi a un server sicuro](#)
 - [Certificati OpenSSL](#)

- **Certificati SChannel**
- **Inviare messaggi**
 - **Inviare un messaggio di testo**
 - **Inviare un messaggio binario**
- **Ricevere messaggi**
 - **Ricevere messaggi di testo**
 - **Ricevere messaggi binari**
- **Autenticazione**
 - **Autenticazione del client**
- **Altro**
 - **Eccezioni del client**
 - **HandShake WebSocket del client**
 - **Registrazione del protocollo del client**
 - **Proxy del client**

TsgcWebSocketClient | Connetti al server WebSocket

Proprietà URL

Il modo più semplice per connettersi a un server WebSocket è utilizzare la proprietà **URL** e impostare **Active = true**.

Esempio: connessione a www.esegece.com tramite connessione sicura.

```
oClient = new TsgcWebSocketClient();
oClient.URL = "wss://www.esegece.com:2053";
oClient.Active = true;
```

Host, Porta e Parametri

È possibile connettersi a un server WebSocket usando le proprietà Host e port.

Esempio: connessione a www.esegece.com tramite connessioni sicure

```
oClient = new TsgcWebSocketClient();
oClient.Host = "www.esegece.com";
oClient.Port = 2053;
oClient.TLS = true;
oClient.Active = true;
```

TsgcWebSocketClient | Apertura connessione client

Una volta configurato il client per la connessione a un server, ci sono 3 diverse opzioni per aprire una nuova connessione.

Proprietà Active

Il modo più semplice per aprire una nuova connessione è impostare la proprietà Active su true. Questo tenterà di connettersi al server utilizzando la configurazione del componente.

Se imposta la proprietà Active su false, la connessione verrà chiusa se attiva.

Questo metodo viene eseguito nello stesso thread del chiamante. Pertanto, se viene chiamato dal Thread Principale, il metodo verrà eseguito nel Thread Principale dell'applicazione.

Apri connessione

```
oClient = new TsgcWebSocketClient();  
....  
oClient.Active = true;
```

Quando si imposta Active = true, **non è ancora possibile inviare dati al server** perché il client potrebbe essere ancora in fase di connessione. È necessario attendere prima che venga generato l'evento OnConnect, dopodiché è possibile iniziare a inviare messaggi al server.

Chiudi connessione

```
oClient.Active = false;
```

Quando si imposta Active = false, **non si può essere certi che la connessione sia già chiusa** subito dopo questo codice; è quindi necessario attendere fino a quando non viene generato l'evento OnDisconnect.

Metodi Start/Stop

Quando si chiama Start() o Stop() per connettere/disconnettere dal server, la chiamata viene eseguita in un thread secondario, quindi non blocca il thread da cui è stata chiamata. Utilizzare questo metodo se si desidera connettersi a un server e lasciare che il codice successivo continui l'esecuzione.

Apri connessione

```
oClient = new TsgcWebSocketClient();  
....  
oClient.Start();
```

Quando si chiama Start(), **non è ancora possibile inviare dati al server** perché il client potrebbe essere ancora in fase di connessione. È necessario attendere prima che l'evento OnConnect venga attivato, e poi si può iniziare a inviare messaggi al server.

Chiudi connessione

```
oClient.Stop();
```

Quando chiama `Stop()`, **non può essere sicuro che la connessione sia già chiusa** subito dopo questo codice, quindi deve attendere finché non viene generato l'evento `OnDisconnect`.

Metodi Connect/Disconnect

Quando si chiama `Connect()` o `Disconnect()` per aprire/chiusare una connessione al server, la chiamata viene eseguita nello stesso thread in cui viene invocata, ma attende fino al termine del processo. È necessario impostare un `Timeout` per definire il tempo massimo da attendere fino al termine del processo (per impostazione predefinita 10 secondi).

Esempio: connessione al server e attesa fino a 5 secondi

```
oClient = new TsgcWebSocketClient();
....
if (oClient.Connect(5000) == true)
{
    oClient.WriteData("Hello from client");
}
else
{
    Error();
}
```

Se il metodo `Connect()` restituisce un risultato positivo, è già possibile inviare un messaggio al server perché la connessione è attiva.

Esempio: disconnettersi dal server e attendere fino a 10 secondi

```
if (oClient.Disconnect(10000) == true)
{
    ShowMessage("Disconnected");
}
else
{
    ShowMessage("Not Disconnected");
}
```

Se il metodo `Disconnect()` restituisce un risultato positivo, significa che la connessione è già chiusa.

L'evento **OnBeforeConnect** può essere utilizzato per personalizzare le proprietà di connessione al server prima che il client tenti di connettersi ad esso.

TsgcWebSocketClient | Client Close Connection

La connessione può essere chiusa utilizzando la proprietà `Active` oppure i metodi `Stop` o `Disconnect`; ne legga di più da [Client Open Connection](#).

CleanDisconnect

Quando una connessione viene chiusa, è possibile notificare all'altro peer che la connessione sta per essere chiusa inviando un messaggio di chiusura. Per abilitare questa funzionalità, impostare la proprietà `Options.CleanDisconnect` su `true`.

Se questa proprietà è abilitata, prima della chiusura della connessione verrà inviato al server un messaggio `Close` per notificare che il client sta chiudendo la connessione.

Disconnetti

[TsgcWSConnection](#) dispone di un metodo chiamato `Disconnect()`, che consente di disconnettere la connessione a livello di socket. Chiamando questo metodo, il socket verrà disconnesso direttamente senza attendere alcuna risposta dal server. Con questo metodo è possibile inviare un `Codice di Chiusura`.

Chiudi

[TsgcWSConnection](#) ha un metodo chiamato `Close()`, che Le consente di inviare un messaggio al server richiedendo di chiudere la connessione. Se il server riceve questo messaggio, deve chiudere la connessione e il client riceverà una notifica che la connessione è chiusa. Può inviare un `Close Code` con questo metodo.

TsgcWebSocketClient | Mantieni la Connessione Client Aperta

Una volta che il client si è connesso a un server, la connessione può talvolta essere chiusa a causa di segnale debole, errori di connessione, ecc. Esistono 2 proprietà che aiutano a mantenere la connessione attiva.

HeartBeat

HeartBeat consente di inviare un Ping ogni X secondi per mantenere attiva la connessione. Alcuni server chiudono le connessioni TCP se non vengono scambiati dati tra i peer. HeartBeat risolve questo problema inviando un ping a un intervallo specifico. Di solito questo è sufficiente per mantenere attiva una connessione, ma è possibile impostare un intervallo di Timeout per chiudere la connessione quando una risposta dal server non viene ricevuta dopo X secondi.

Esempio: inviare un ping ogni 30 secondi

```
oClient = new TsgcWebSocketClient();
oClient.HeartBeat.Interval = 30;
oClient.HeartBeat.Timeout = 0;
oClient.HeartBeat.Enabled = true;
oClient.Active = true;
```

È disponibile un evento denominato **OnBeforeHeartBeat** che consente di personalizzare il comportamento di HeartBeat. Per impostazione predefinita, se HeartBeat è abilitato, il client invierà un ping WebSocket ogni X secondi come impostato dalla proprietà HeartBeat.Interval.

OnBeforeHeartBeat ha un parametro chiamato Handled, per impostazione predefinita è false, il che significa che il flusso è controllato dal componente TsgcWebSocketClient. Se si imposta il valore su True, il ping non verrà inviato ed è possibile inviare un messaggio personalizzato utilizzando la classe Connection.

WatchDog

Se WatchDog è abilitato, quando il client rileva una disconnessione, WatchDog tenta di riconnettersi ogni X secondi finché la connessione non è nuovamente attiva.

Esempio: riconnessione ogni 10 secondi dopo una disconnessione con tentativi illimitati.

```
oClient = new TsgcWebSocketClient();
oClient.WatchDog.Interval = 10;
oClient.WatchDog.Attempts = 0;
oClient.WatchDog.Enabled = true;
oClient.Active = true;
```

È possibile utilizzare l'evento **OnBeforeWatchDog** per cambiare il server a cui il client tenterà di connettersi. **Esempio:** dopo 3 tentativi, se il client non riesce a connettersi a un server, tenterà di connettersi a un server secondario.

La proprietà **Handled**, se impostata su True, significa che il client non tenterà di riconnettersi.

TsgcWebSocketClient | Disconnessioni interrotte

Una volta stabilita la connessione, se nessun peer invia dati, non vengono inviati pacchetti in rete. TCP è un protocollo passivo, quindi presuppone che la connessione sia ancora attiva.

Motivi di disconnessione

- **L'applicazione si chiude:** quando un processo è terminato, di solito invia un pacchetto FIN che conferma all'altro peer che la connessione è stata chiusa. Ma se un processo va in crash, non c'è alcuna garanzia che questo pacchetto venga inviato all'altro peer.
- **Il dispositivo si disconnette:** se il dispositivo si disconnette, molto probabilmente non verrà inviata alcuna notifica al riguardo.
- **Cavo di rete scollegato:** se il cavo di rete viene scollegato, è come se un router si chiudesse; non vengono trasferiti dati quindi la connessione non viene chiusa.
- **Perdita del segnale dal router:** se l'applicazione perde il segnale dal router, la connessione rimarrà attiva.

Rilevare disconnessioni Half-Open

È possibile provare a rilevare le disconnessioni utilizzando i seguenti metodi

Seconda connessione

È possibile tentare di aprire una seconda connessione e provare a connettersi, ma ciò presenta alcuni svantaggi: si consumano più risorse, si creano nuovi thread, ecc. Inoltre, se l'altro peer ha riavviato, la seconda connessione funzionerà ma la prima no.

Ping all'altro peer

Se si tenta di inviare un ping o qualsiasi altro messaggio con una connessione half-open, si noterà che non viene ricevuto alcun errore.

Abilita KeepAlive a livello di socket TCP

Un pacchetto TCP keep-alive è semplicemente un ACK con il numero di sequenza impostato a uno in meno rispetto al numero di sequenza corrente per la connessione. Un host che riceve uno di questi ACK risponde con un ACK per il numero di sequenza corrente. I keep-alive possono essere utilizzati per verificare che il computer all'estremità remota di una connessione sia ancora disponibile. I TCP keep-alive possono essere inviati ogni `TCPKeepAlive.Time` (predefinito 7.200.000 millisecondi o due ore) se non sono stati trasmessi altri dati o keep-alive di livello superiore sulla connessione TCP. Se non vi è risposta a un keep-alive, viene ripetuto ogni `TCPKeepAlive.Interval` secondi. `KeepAliveInterval` è predefinito a 1000 millisecondi.

È possibile abilitare KeepAlive per connessione e consentire al protocollo TCP di verificare se la connessione è attiva o meno. Questo è il metodo preferito se si desidera rilevare disconnessioni improvvise (ad esempio: quando si scollega un cavo di rete).

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
oClient.TCPKeepAlive.Enabled = true;
oClient.TCPKeepAlive.Time = 5000;
oClient.TCPKeepAlive.Interval = 1000;
```

TsgcWebSocketClient | Connessione al server TCP

TsgcWebSocketClient può connettersi a server WebSocket, ma può anche connettersi a server TCP semplici.

Proprietà URL

Il modo più semplice per connettersi a un server TCP è utilizzare la proprietà **URL** e impostare **Active = true**.

Esempio: connettersi a 127.0.0.1 porta 5555

```
oClient = new TsgcWebSocketClient();
oClient.URL = "tcp://127.0.0.1:5555";
oClient.Active = true;
```

Host, Porta e Parametri

È possibile connettersi a un server TCP utilizzando le proprietà Host e port.

Esempio: connettersi a 127.0.0.1 porta 5555

```
oClient = new TsgcWebSocketClient();
oClient.Specifications.RFC6455 = false;
oClient.Host = "127.0.0.1";
oClient.Port = 5555;
oClient.Active = true;
```


TsgcWebSocketClient | Connessioni

TIME_WAIT

Quando un client avvia una disconnessione dal server, avviene uno scambio tra client e server per comunicare lo stato della disconnessione. Al termine del processo, il socket del client entra nello stato TIME_WAIT per una durata variabile. Si tratta di un comportamento normale; nei sistemi operativi Windows, questa durata è di circa 4 minuti per impostazione predefinita.

È possibile ridurre o eliminare questo comportamento, ma procedere con cautela, utilizzando le seguenti alternative.

REGEDIT

È possibile ridurre il valore TIME_WAIT utilizzando il Regedit di Windows

1. Aprire Regedit e accedere alle sottochiavi di registro HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TCP/IP\Parameters.
2. Creare un nuovo valore REG_DWORD denominato **TcpTimedWaitDelay**
3. Impostare il valore in secondi. Esempio: se si imposta un valore di 5, significa che TIME_WAIT attenderà per un massimo di 5 secondi.
4. Salvare e riavviare il sistema.

LINGER

Un'altra opzione per evitare lo stato TIME_WAIT consiste nell'utilizzare l'opzione socket SO_LINGER. Se abilitata, invece di chiudere la connessione correttamente, il client reimposta la connessione in modo da evitare lo stato TIME_WAIT.

È possibile abilitare questa opzione tramite la proprietà **LingerState**, che per impostazione predefinita ha un valore di -1. Se si imposta un valore pari a zero, la connessione verrà reimpostata durante la disconnessione dal socket senza timeout.

Questa opzione è probabilmente la meno consigliata e dovrebbe essere utilizzata solo come ultima risorsa.

TsgcWebSocketClient | Reindirizzamenti WebSocket

Quando il client si connette a un server WebSocket, il server può restituire un codice di risposta HTTP 30x. Se il codice di risposta è 301, significa che la posizione è stata spostata definitivamente e il nuovo URL è fornito nell'intestazione HTTP Location.

Il client WebSocket gestisce i reindirizzamenti automaticamente, quindi se rileva che la risposta del server contiene un reindirizzamento, disconnetterà la connessione corrente e tenterà di connettersi al nuovo URL Location.

Esempio

1. Il client tenta prima di connettersi all'url `ws://127.0.0.1:5000`
2. Il server restituisce un codice di risposta 301 e contiene un header Location con il valore `ws://80.50.1.2:3000`
3. Il client legge la risposta dal server, rileva che si tratta di un reindirizzamento e legge la posizione (Location)
 1. Prima disconnette la connessione corrente.
 2. Aggiorna la proprietà URL con il valore dell'Header Location (`ws://80.50.1.2:3000`)
 3. Si connette al nuovo server.

TsgcWebSocketClient | Connect Secure Server

TsgcWebSocketClient può connettersi a server WebSocket tramite connessioni sicure e non sicure.

È possibile configurare una connessione sicura, utilizzando la proprietà URL oppure le proprietà Host / Port; consultare [Connetti al Server WebSocket](#).

TLSOptions

Nella proprietà **TLSOptions** sono presenti le proprietà per **personalizzare una connessione sicura**. La proprietà più importante è **version**, che specifica la **versione del protocollo TLS**. Di solito impostare la **proprietà TLS su true** e **TLSOptions.Version su tlsUndefined** è sufficiente per la grande maggioranza dei server WebSocket.

TLSOptions.Version consente di impostare la versione TLS utilizzata per connettersi al server oppure di lasciare che il client negozi la versione TLS tra quelle disponibili (questa è l'impostazione predefinita quando il valore è **tlsUndefined**).

Se si riceve un **errore durante la connessione a un server** relativo al protocollo TLS, **molto probabilmente** il server **richiede una versione TLS più recente** di quella impostata.

Se **TLSOptions.IOHandler** è impostato su **iohOpenSSL**, è necessario **distribuire le librerie OpenSSL** (che sono le librerie che gestiscono tutto ciò che riguarda TLS); consultare il seguente articolo su [OpenSSL](#).

Se **TLSOptions.IOHandler** è impostato su **iohSChannel**, allora **non è necessario distribuire** alcuna libreria (è supportato solo Windows).

TsgcWebSocketClient | Certificati OpenSSL

Quando il server richiede che il client si connetta utilizzando un certificato SSL, utilizzare la proprietà `TLSEOptions` di `TsgcWebSocketClient` per impostare i file del certificato. Il certificato deve essere in formato PEM; se il certificato ha un formato diverso, deve prima essere convertito in PEM.

La connessione tramite le librerie OpenSSL richiede che `TLSEOptions.IOHandler = iohOpenSSL`.

Configuri le seguenti proprietà:

- **CertFile:** è il percorso del certificato in formato PEM.
- **KeyFile:** è il percorso della chiave privata del certificato.
- **RootCertFile:** è il percorso della radice del certificato.
- **Password:** se il certificato è protetto da una password, inserire qui il segreto.

TsgcWebSocketClient | Certificati SChannel

Quando il server richiede che il client si connetta utilizzando un certificato SSL, utilizzare la proprietà `TLSOptions` di `TsgcWebSocketClient` per impostare i file del certificato.

La connessione tramite SChannel richiede che `TLSOptions.IOHandler = iohSChannel`.

SChannel supporta 2 tipi di autenticazione tramite certificato:

1. Utilizzo di un **certificato PFX**
2. Impostazione dell'**Hash Certificate** di un certificato già installato nel sistema Windows.

Certificato PFX

Il certificato PFX è un file che contiene il certificato e la chiave privata; a volte si dispone di un certificato in formato PEM, quindi prima di utilizzarlo è necessario convertirlo in PFX.

Utilizzare il seguente comando openssl per convertire un certificato PEM in PFX

```
openssl pkcs12 -inkey certificate-pem.key -in certificate-pem.crt -export -out certificate.pfx
```

Una volta che il certificato è in formato PFX, è sufficiente distribuirlo e impostare la proprietà `TLSOptions.CertFile` sul percorso corrispondente.

```
TLSOptions.IOHandler = iohSChannel
TLSOptions.CertFile = <certificate path>
TLSOptions.Password = <certificate optional password>
```

Hash Certificate

Se il certificato è già installato nell'archivio certificati di Windows, è necessario conoscere solo l'identificazione personale del certificato e impostarla nella proprietà `TLSOptions.SChannel_Options`.

Trovare l'hash di un certificato è altrettanto semplice in **powershell** quanto eseguire un comando **dir** sul contenitore dei certificati.

```
dir cert:\localmachine\my
```

L'hash è il valore **Thumbprint** esadecimale.

```
Directory: Microsoft.PowerShell.Security\Certificate::localmachine\my
Thumbprint Subject
-----
C12A8FC8AE668F866B48F23E753C93D357E9BE10 CN=*.mydomain.com
```

Una volta ottenuto il valore Thumbprint, è necessario impostare l'hash e la posizione del certificato nella proprietà `TLSOptions.SChannel_Options`.

```
TLSOptions.IOHandler = iohSChannel
TLSOptions.SChannel_Options.CertHash = <certificate thumbprint>
TLSOptions.SChannel_Options.CertStoreName = <certificate store name>
TLSOptions.SChannel_Options.CertStorePath = <certificate store path>
TLSOptions.Password = <certificate optional password>
```

TsgcWebSocketClient | Invio di messaggi di testo dal client

Una volta che il client si è connesso al server, può inviare messaggi di testo. Per inviare un messaggio di testo, chiami semplicemente il metodo `WriteData()`.

Invia un messaggio di testo

Chiamare il metodo **`WriteData()`** per inviare un messaggio di testo. Questo metodo viene eseguito sullo **stesso thread** da cui viene chiamato.

```
TsgcWebSocketClient1.WriteData("My First sgcWebSockets Message!.");
```

Se **`QueueOptions.Text`** ha un **valore diverso da `qmNone`**, invece di essere elaborato nello stesso thread in cui viene chiamato, verrà elaborato in un thread secondario. Per impostazione predefinita questa opzione è disabilitata.

Invia un Messaggio di Testo e Attendi la Risposta

Chiamare il metodo `WriteAndWaitData()` per inviare un messaggio di testo e attendere una risposta dal server. La funzione restituisce il messaggio di testo ricevuto.

```
TsgcWebSocketClient1.WriteAndWaitData("My First sgcWebSockets Message!.");
```

TsgcWebSocketClient | Invio di un messaggio binario dal client

Una volta che il client si è connesso al server, può inviare messaggi binari. Per inviare un messaggio binario, è sufficiente chiamare il metodo `WriteData()`.

Invia un messaggio binario

Chiamare il metodo **`WriteData()`** per inviare un messaggio binario. Questo metodo viene eseguito sullo **stesso thread** da cui viene chiamato.

```
byte[] bytes;  
...  
TsgcWebSocketClient1.WriteData(bytes);
```

Se **`QueueOptions.Binary`** ha un **valore diverso da `qmNone`**, invece di essere elaborato sullo stesso thread da cui viene chiamato, verrà elaborato su un thread secondario. Per impostazione predefinita questa opzione è disabilitata.

Inviare un messaggio binario e attendere la risposta

Chiamare il metodo `WriteAndWaitData()` per inviare un messaggio binario e attendere una risposta dal server. La funzione restituisce il messaggio binario ricevuto.

```
TsgcWebSocketClient1.WriteAndWaitDataData(bytes);
```

TsgcWebSocketClient | Client Send a Text and Binary Message

Il protocollo WebSocket consente solo due tipi di messaggi: testo o binario. Tuttavia, non è possibile inviare dati binari insieme a testo nello stesso messaggio.

Un modo per risolvere questo problema è aggiungere un'intestazione al messaggio binario prima che venga inviato e decodificare questo messaggio binario quando viene ricevuto.

Ci sono 2 funzioni in `sgcWebSocket_Helpers` che possono essere utilizzate per impostare una breve descrizione di un pacchetto binario. Ciò aggiunge fondamentalmente un'intestazione al flusso che viene utilizzata per identificare il pacchetto binario.

Prima di inviare un messaggio binario, chiami il metodo per codificare lo stream.

```
sgcWSBytesWrite("00001", oBytes);  
TsgcWebSocketClient1.WriteData(oBytes);
```

Quando viene ricevuto un messaggio binario, chiami il metodo per decodificare lo stream.

```
sgcWSBytesRead(oBytes, vID);
```

L'unica limitazione è che il testo utilizzato per identificare il messaggio binario ha una lunghezza massima di 10 caratteri (questo valore può essere modificato se si ha accesso al codice sorgente).

TsgcWebSocketClient | Ricezione di messaggi di testo

Quando il client riceve un Text Message, viene generato l'evento **OnMessage**. Legga il parametro Text per recuperare la stringa del messaggio ricevuto.

```
void OnMessage(TsgcWSConnection Connection, string Text)
{
    MessageBox.Show("Message Received from Server: " + Text);
}
```

Per impostazione predefinita, il client utilizza il metodo **neAsynchronous** per inviare l'evento OnMessage, ciò significa che **questo evento viene eseguito nel contesto del thread principale**, quindi è thread-safe aggiornare qualsiasi controllo di un form, ad esempio.

Se il client riceve molti messaggi o è necessario controllare la sincronizzazione con altri thread, impostare la proprietà NotifyEvents su **neNoSync**; ciò significa che l'evento OnMessage verrà **eseguito nel contesto del thread di connessione**, quindi se è necessario aggiornare un controllo di un form o accedere a oggetti condivisi, è necessario implementare i propri metodi di sincronizzazione.

TsgcWebSocketClient | Receive Binary Messages

Quando il client riceve un Messaggio Binario, viene generato l'evento **OnBinary**. Leggere il parametro Data per recuperare il messaggio binario ricevuto.

```
private void OnBinary(TsgcWSConnection Connection, byte[] Bytes)
{
    MemoryStream stream = new MemoryStream(Bytes);
    pictureBox1.Image = new Bitmap(stream);
}
```

Per impostazione predefinita, il client utilizza il metodo **neAsynchronous** per inviare l'evento OnMessage, il che significa che **questo evento viene eseguito nel contesto del Thread Principale**, quindi è thread-safe aggiornare qualsiasi controllo di un form, ad esempio.

If your client receives lots of messages or you need to control the synchronization with other threads, set NotifyEvents property to **neNoSync**, this means that OnMessage event will be **executed on the context of connection thread**, so if you require to update any control of a form or access shared objects, you must implement your own synchronization methods.

TsgcWebSocketClient | Client Authentication

TsgcWebSocket client supporta 4 tipi di autenticazione:

- **Basic:** invia un'intestazione HTTP durante l'handshake WebSocket con utente e password codificati come autenticazione Basic.
- **Token:** invia un Token come Header HTTP durante l'HandShake WebSocket. Imposti il token richiesto in Authentication.Token.AuthToken come richiesto dal server.
- **Session:** il client richiede prima una sessione HTTP al server e, se il server restituisce una sessione, questa viene passata nell'intestazione HTTP GET dell'handshake WebSocket. (* metodo di autenticazione proprietario della libreria sgcWebSockets).
- **URL:** il client richiede l'autorizzazione utilizzando l'intestazione HTTP GET dell'handshake WebSocket. (* metodo di autenticazione proprio della libreria sgcWebSockets).

Autenticazione Basic

Questo è un semplice metodo di autorizzazione in cui nome utente e password vengono codificati e passati come intestazione HTTP. È sufficiente impostare User e Password e abilitare solo il tipo di autorizzazione Basic per utilizzare questo metodo.

```
oClient = new TsgcWebSocketClient();
oClient.
```

Authentication

```
.Enabled = true;
oClient.
```

Authentication

```
.Basic.Enabled = true;
oClient.
```

Authentication

```
.User = "your user";
oClient.
```

Authentication

```
.Password = "your password";
oClient.
```

Authentication

```
.Token.Enabled = false;
oClient.
```

Authentication

```
.URL.Enabled = false;
oClient.
```

Authentication

```
.Session.Enabled = false;
oClient.Active = true;
```

Token di autenticazione

Le consente di autorizzare utilizzando JWT. Questo richiede di ottenere un token utilizzando uno strumento esterno (ad esempio: una connessione HTTP, OAuth2, ecc.).

Se si collega un componente OAuth2, è possibile ottenere questo token automaticamente. Per ulteriori informazioni, consultare [OAuth2](#).

È necessario impostare il proprio AuthToken e abilitare l'Autenticazione Token.

```
oClient = new TsgcWebSocketClient();
oClient.
```

Authentication

```
.Enabled = true;
oClient.
```

Authentication

```
.Token.Enabled = true;
oClient.
```

Authentication

```
.Token.AuthToken = "your token";
oClient.
```

Authentication

```
.Basic.Enabled = false;
oClient.
```

Authentication

```
.URL.Enabled = false;
oClient.
```

Authentication

```
.Session.Enabled = false;
oClient.Active = true;
```

Sessione di autenticazione

Per prima cosa il client si connette al server utilizzando una connessione HTTP richiedendo una nuova Sessione. In caso di successo, il server restituisce un SessionId e il client invia questo SessionId nell'header HTTP GET dell'HandShake WebSocket.

Richiede di impostare UserName e Password e di abilitare l'autenticazione di sessione.

```
oClient = new TsgcWebSocketClient();
oClient.
```

Authentication

```
.Enabled = true;
oClient.
```

Authentication

```
.Session.Enabled = true;
oClient.
```

Authentication

```
.User = "your user";
oClient.
```

Authentication

```
.Password = "your password";
oClient.
```

Authentication

```
.Basic.Enabled = false;
oClient.
```

Authentication

```
.URL.Enabled = false;
oClient.
```

Authentication

```
.Token.Enabled = false;
oClient.Active = true;
```

URL di autenticazione

Questo metodo di autenticazione passa il nome utente e la password nell'intestazione HTTP GET dell'handshake WebSocket.

```
oClient = new TsgcWebSocketClient();  
oClient.
```

Authentication

```
.Enabled = true;  
oClient.
```

Authentication

```
.URL.Enabled = true;  
oClient.
```

Authentication

```
.User = "your user";  
oClient.
```

Authentication

```
.Password = "your password";  
oClient.
```

Authentication

```
.Basic.Enabled = false;  
oClient.
```

Authentication

```
.Session.Enabled = false;  
oClient.
```

Authentication

```
.Token.Enabled = false;  
oClient.Active = true;
```

TsgcWebSocketClient | Eccezioni del Client

A volte si verificano errori nelle comunicazioni: il server può disconnettere una connessione perché non è autorizzata o un messaggio non ha il formato corretto. Ci sono 2 eventi in cui gli errori vengono acquisiti.

OnError

Questo evento viene generato ogni volta che si verifica un errore nel protocollo WebSocket, come tipo di messaggio non valido, stringa utf8 non valida...

```
private void OnError(TsgcWSConnection Connection, string aError)
{
    Console.WriteLine("#error: " + aError);
}
```

OnException

Questo evento viene generato ogni volta che si verifica un'eccezione, ad esempio durante la scrittura su un socket non attivo o l'accesso a un oggetto inesistente.

```
private void OnException(TsgcWSConnection Connection, Exception E)
{
    Console.WriteLine("#exception: " + E.Message);
}
```

Per impostazione predefinita, quando una **connessione viene chiusa dal server**, verrà **generata un'eccezione**. Se non si desidera che queste eccezioni vengano generate, disabilitare **Options.RaiseDisconnectExceptions**.

TsgcWebSocketClient | WebSocket Hand-Shake

Il protocollo WebSocket utilizza un handshake HTTP per eseguire l'upgrade dal protocollo HTTP al protocollo WebSocket. Questo handshake è gestito internamente dal componente client TsgcWebSocket, ma è possibile aggiungere intestazioni HTTP personalizzate se il server richiede informazioni aggiuntive nell'intestazione HTTP.

Esempio: se ha bisogno di aggiungere questo Header HTTP "Client: sgcWebSockets"

```
void OnHandshake(TsgcWSConnection Connection, ref string Headers)
{
    Headers = Headers + Environment.NewLine + "Client: sgcWebSockets";
}
```

TsgcWebSocketClient | Protocollo di registrazione client

Per impostazione predefinita, TsgcWebSocketClient non utilizza alcun SubProtocol. I sottoprotocolli WebSocket sono costruiti sopra il protocollo WebSocket e definiscono un protocollo di messaggi personalizzato. Esempi di sottoprotocolli WebSocket includono MQTT, STOMP, ecc.

Il nome del SubProtocol WebSocket viene inviato come intestazione HTTP durante l'handshake WebSocket. Questa intestazione viene elaborata dal server e, se il server supporta tale sottoprotocollo, accetterà la connessione. In caso contrario, la connessione verrà chiusa automaticamente.

Esempio: connessione a un server websocket con nome SubProtocol 'myprotocol'

```
Client = new TsgcWebSocketClient();
Client.Host = "server host";
Client.Port = server.port;
Client.RegisterProtocol("myprotocol");
Client.Active = true;
```


TsgcWebSocketClient | Client Proxies

Il client TsgcWebSocket supporta le connessioni attraverso proxy. Per configurare una connessione proxy, è sufficiente compilare le proprietà **Proxy** del client TsgcWebSocket.

```
Client = new TsgcWebSocketClient();
Client.Proxy.Enabled = true;
Client.Proxy.Username = "user";
Client.Proxy.Password = "secret";
Client.Proxy.Host = "80.55.44.12";
Client.Proxy.Port = 8080;
Client.Active = true;
```

TsgcTCPClient

TsgcTCPClient è un componente client TCP che consente la connessione a server TCP semplici.

Introduzione

TsgcTCPClient è un componente client TCP che consente di connettersi a server TCP semplici. Fornisce supporto per connessioni TLS/SSL, server proxy, monitoraggio dell'heartbeat, riconnessione automatica (WatchDog) e TCP keep-alive.

1. Rilasci un componente **TsgcTCPClient** sul form.
2. Imposti **Host** e **Port** (predefinito è 80) per connettersi a un TCP Server disponibile.
3. Impostare **TLS** su True se si richiede una connessione sicura (il valore predefinito è False).
4. Configuri facoltativamente il **Proxy** se si connette tramite un server proxy.
5. Gestire gli eventi: **OnConnect**, **OnDisconnect**, **OnMessage**, **OnBinary**, **OnError**, **OnException**.
6. Impostare **Active** su True per aprire la connessione.

Riferimento

- Proprietà — 23 proprietà disponibili sul componente.
- Metodi — 7 metodi pubblici.
- Eventi — 12 eventi gestibili.

TsgcTCPClient

TsgcTCPClient è un componente client TCP che consente la connessione a server TCP semplici.

Introduzione

TsgcTCPClient è un componente client TCP che consente di connettersi a server TCP semplici. Fornisce supporto per connessioni TLS/SSL, server proxy, monitoraggio dell'heartbeat, riconnessione automatica (WatchDog) e TCP keep-alive.

1. Rilasci un componente **TsgcTCPClient** sul form.
2. Imposti **Host** e **Port** (predefinito è 80) per connettersi a un TCP Server disponibile.
3. Impostare **TLS** su True se si richiede una connessione sicura (il valore predefinito è False).
4. Configuri facoltativamente il **Proxy** se si connette tramite un server proxy.
5. Gestire gli eventi: **OnConnect**, **OnDisconnect**, **OnMessage**, **OnBinary**, **OnError**, **OnException**.
6. Impostare **Active** su True per aprire la connessione.

Riferimento

- Proprietà — 23 proprietà disponibili sul componente.
- Metodi — 7 metodi pubblici.
- Eventi — 12 eventi gestibili.

TsgcWebSocketServer

TsgcWebSocketServer implementa il componente WebSocket Server e può gestire più connessioni client multi-thread.

Introduzione

TsgcWebSocketServer implementa il componente WebSocket server e può gestire più connessioni client in thread separati. Seguire i passaggi riportati di seguito per configurare questo componente:

1. Rilasciare un componente TsgcWebSocketServer sul form
2. Impostare la porta (il valore predefinito è 80). Se si è protetti da un firewall, probabilmente sarà necessario configurarlo.
3. Impostare le specifiche consentite; per impostazione predefinita, tutte le specifiche sono consentite.

RFC6455: è la specifica WebSocket standard e raccomandata.

Hixie76: è una bozza ed è consigliato stabilire connessioni Hixie76 solo se si desidera fornire supporto per browser obsoleti come Safari 4.2

```
void OnTCPConnectEvent(SgcWSConnection aConnection, ref bool Accept)
{
    aConnection.Transport = trpTcp;
    Accept = true;
}
```

5. Creare una procedura e impostare la proprietà Active = True.

Riferimento

- Proprietà — 27 proprietà disponibili nel componente.
- Metodi — 7 metodi pubblici.
- Eventi — 22 eventi che è possibile gestire.

Guide

- **Avvio**
 - [Avvio del server](#)
 - [Binding del server](#)
 - [Avvio - Arresto del server](#)
 - [Mantenimento del server attivo](#)
- **Connessioni**
 - [Server Keep Connections Alive](#)
 - [Server Plain TCP](#)
 - [Server Close Connection](#)
- **Autenticazione**

- Autenticazione del Server
- **Invia messaggi**
 - Server - Invia messaggio di testo
 - Server - Invia messaggio binario
- **Ricevi Messaggi**
 - Il Server Riceve un Messaggio di Testo
 - Il Server Riceve un Messaggio Binario
- **SSL**
 - Server SSL
 - Server SSL SChannel

TsgcWebSocketServer | Avvia Server

La prima cosa da impostare quando si vuole avviare un server è la porta di ascolto. Per impostazione predefinita, questa è impostata sulla porta 80, ma è possibile cambiarla con qualsiasi porta. Una volta impostata la porta, esistono 2 metodi per avviare un server.

Proprietà Active

Se si imposta la proprietà Active a true, il server inizierà ad ascoltare tutte le connessioni in entrata sulla porta configurata.

```
oServer = new TsgcWebSocketServer();  
oServer.Port = 80;  
oServer.Active = true;
```

Se imposta la proprietà Active su false, il server si arresterà e chiuderà tutte le connessioni attive.

```
oServer.Active = false;
```

Metodi Start / Stop

Mentre l'impostazione della proprietà Active avvia/arresta il server nello stesso thread, i metodi Start e Stop vengono eseguiti in un thread secondario.

```
oServer = new TsgcWebSocketServer();  
oServer.Port = 80;  
oServer.Start();
```

Se si chiama il metodo Stop(), il server si fermerà e chiuderà tutte le connessioni attive.

```
oServer.Stop();
```

È possibile utilizzare il metodo **ReStart**, per fermare e avviare il server in un thread secondario.

TsgcWebSocketServer | Binding del server

Per impostazione predefinita, se si compila solo la **proprietà Port**, il server **collega la porta di ascolto su TUTTI gli IP**, quindi se ad esempio si dispone di 3 IP: 127.0.0.1, 80.54.11.22 e 12.55.41.17, il server collegherà questa porta su tutti e 3 gli IP.

Di solito è consigliabile associarsi solo agli IP necessari. È qui che è possibile utilizzare la proprietà Bindings. Invece di utilizzare la proprietà Port, è sufficiente usare la proprietà Bindings e specificare l'IP e la porta richiesti.

Esempio: associa la porta 5555 all'IP 127.0.0.1 e all'IP 80.58.25.40

```
oServer = new TsgcWebSocketServer();  
oServer.Bindings = "127.0.0.1:5555,80.58.25.40:5555";  
oServer.Active = true;
```

TsgcWebSocketServer | Avvio e Arresto del Server

Una volta impostate tutte le configurazioni necessarie del server, ci sono 2 utili eventi per sapere quando il server è avviato e quando si è fermato.

OnStartup

Questo evento viene generato quando il server è stato avviato e può elaborare nuove connessioni.

```
void OnStartup()  
{  
    Console.WriteLine("#server started");  
}
```

OnShutdown

Questo evento viene attivato dopo che il server si è fermato e non vengono più accettate connessioni.

```
void OnShutdown()  
{  
    Console.WriteLine("#server stopped");  
}
```


TsgcWebSocketServer | Mantenimento attivo del server

Una volta avviato il server, a volte può arrestarsi per qualsiasi motivo. Se si desidera riavviare il server dopo un arresto imprevisto, è possibile utilizzare la proprietà WatchDog.

WatchDog

Se WatchDog è abilitato, quando il server rileva un arresto, WatchDog prova a riavviarsi ogni X secondi finché il server non torna nuovamente attivo.

Esempio: riavvia ogni 10 secondi dopo un'interruzione imprevista con tentativi illimitati.

```
oServer = new TsgcWebSocketServer();  
oServer.WatchDog.Interval = 10;  
oServer.WatchDog.Attempts = 0;  
oServer.WatchDog.Enabled = true;  
oServer.Active = true;
```

TsgcWebSocketServer | SSL server

Il server può essere configurato per utilizzare **Certificati SSL**. Per ottenere un server di produzione con un certificato server, è necessario **acquistare** un certificato da un **provider rinomato**: Namecheap, GoDaddy, Thawte, ecc. A **scopo di test** è possibile utilizzare un **certificato auto-firmato** (consultare l'esempio Demos/Chat che utilizza un certificato auto-firmato).

Il certificato deve essere in **formato PEM**, PEM (da Privacy Enhanced Mail) è definito nelle RFC da 1421 a 1424, si tratta di un formato contenitore che può includere solo il certificato pubblico (come nelle installazioni Apache e nei file di certificato CA /etc/ssl/certs), oppure può includere un'intera catena di certificati comprendente chiave pubblica, chiave privata e certificati root. Per creare un singolo certificato pem, basta aprire il file della sua chiave privata, copiarne il contenuto e incollarlo nel file del certificato.

Esempio:

certificate.crt

```
-----BEGIN CERTIFICATE-----
.....
-----END CERTIFICATE-----
```

certificate.key

```
-----BEGIN PRIVATE KEY-----
.....
-----END PRIVATE KEY-----
```

certificate.pem

```
-----BEGIN PRIVATE KEY-----
.....
-----END PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
.....
-----END CERTIFICATE-----
```

Per abilitare SSL, è sufficiente **abilitare la proprietà SSL** e configurare i percorsi di **CertFile**, **KeyFile** e **RootFile**. Se il certificato contiene l'intera catena di certificati (chiave pubblica, chiave privata, ecc.), impostare tutti i percorsi allo stesso certificato.

Un'altra proprietà da impostare è **SSLOptions.Port**, che è la porta utilizzata per le connessioni sicure.

Configurazione SSL Semplice

Esempio: configurare SSL sull'IP 127.0.0.1 e sulla porta 443

```
oServer = new TsgcWebSocketServer();
oServer.SSL = true;
oServer.SSLOptions.CertFile = "c:\\certificates\\mycert.pem";
oServer.SSLOptions.KeyFile = "c:\\certificates\\mycert.pem";
oServer.SSLOptions.RootCertFile = "c:\\certificates\\mycert.pem";
oServer.SSLOptions.Port = 443;
oServer.Port = 443;
oServer.Active = true;
```

SSL e Non-SSL

È possibile configurare il server per ascoltare su più di un IP e porta; consulti l' [articolo Binding](#) che spiega come funziona. Il server può essere configurato per consentire connessioni SSL e connessioni non SSL contemporaneamente (ovviamente, ascoltando su porte diverse). Le basta effettuare il binding su due porte diverse e configurare una porta per le connessioni SSL e un'altra porta per le connessioni non SSL.

Esempio: configuri il server in IP 127.0.0.1, porta 80 (non crittografata) e 443 (SSL)

```
oServer = new TsgcWebSocketServer();  
oServer.Bindings = "127.0.0.1:80,127.0.0.1:443"  
oServer.SSL = true;  
oServer.Port = 80;  
oServer.SSLOptions.CertFile = "c:\\certificates\\mycert.pem";  
oServer.SSLOptions.KeyFile = "c:\\certificates\\mycert.pem";  
oServer.SSLOptions.RootCertFile = "c:\\certificates\\mycert.pem";  
oServer.SSLOptions.Port = 443;  
oServer.Active = true;
```

Server SSL | SChannel per server Indy

I componenti server basati su Indy (**TsgcWebSocketServer**, **TsgcWebsocketHTTPServer**) possono utilizzare **Windows SChannel** (Secure Channel) come provider TLS invece di OpenSSL. SChannel è l'implementazione TLS nativa di Windows, quindi non richiede alcuna DLL esterna.

Come funziona

Quando un componente server ha SSL abilitato e l'IOHandler è impostato su **iohSChannel**, il server crea un'istanza di **TsgcIdServerIOHandlerSSLSChannel** che gestisce tutte le operazioni TLS tramite l'API Windows SChannel. Per ogni connessione client in entrata, il server esegue l'handshake TLS tramite il provider SChannel, negoziando la versione del protocollo, la suite di cifratura e associando il certificato configurato.

SChannel legge i certificati dal **Windows Certificate Store** o da un file **PFX** (.pfx / .p12). Non sono necessari file PEM e non è necessario distribuire librerie OpenSSL.

Configurazione

Per abilitare SChannel su un server Indy, configuri le seguenti proprietà:

1. Impostare la proprietà **SSL** su **True**.
2. Impostare **SSLOptions.IOHandler** su **iohSChannel**.
3. Imposti **SSLOptions.Version** alla versione TLS desiderata (tls1_2, tls1_3, ...).
4. Impostare **SSLOptions.Port** sulla porta utilizzata per le connessioni sicure.
5. Configurare il certificato utilizzando uno dei due metodi descritti di seguito.

Proprietà di SChannel_Options

La sotto-proprietà **SSLOptions.SChannel_Options** espone le impostazioni specifiche di SChannel:

Proprietà	Descrizione
CertHash	L'impronta digitale (hash esadecimale) di un certificato installato nell'archivio certificati di Windows.
CertStoreName	Quale archivio di certificati cercare: scsnMY (Personale), scsnRoot , scsnTrust , scsnCA .
CertStorePath	Posizione dell'archivio: scspStoreLocalMachine o scspStoreCurrentUser .
CipherList	Elenco facoltativo separato da due punti di algoritmi di cifratura (ad es. CALG_AES_256:CALG_AES_128). Vuoto significa utilizzare le impostazioni predefinite del sistema.
UseLegacyCredentials	Se True, utilizza la struttura SCHANNEL_CRED legacy invece di SCH_CREDENTIALS. Abilitare questa opzione per le versioni precedenti di Windows che non supportano la nuova API.

Inoltre, le proprietà generali di **SSLOptions CertFile** e **Password** vengono utilizzate durante il caricamento di un certificato da un file PFX.

Certificato da Windows Store

Se il certificato è già installato nell'archivio certificati di Windows, fornire l'**impronta digitale** del certificato e indicare dove si trova.

Per trovare il thumbprint, aprire **PowerShell** ed eseguire:

```
dir cert:\localmachine\my
```

La colonna **Thumbprint** mostra l'hash esadecimale necessario.

```
Directory: Microsoft.PowerShell.Security\Certificate::localmachine\my
Thumbprint                               Subject
-----
C12A8FC8AE668F866B48F23E753C93D357E9BE10  CN=*.mydomain.com
```

```
TsgcWebSocketServer oServer = new TsgcWebSocketServer();
oServer.SSL = true;
oServer.SSLOptions.IOHandler = TsgcSSLIOHandler.iohSChannel;
oServer.SSLOptions.Version = TsgcTLSVersion.tls1_2;
oServer.SSLOptions.Port = 443;
oServer.Port = 443;
oServer.SSLOptions.SChannel_Options.CertHash = "C12A8FC8AE668F866B48F23E753C93D357E9BE10";
oServer.SSLOptions.SChannel_Options.CertStoreName = TsgcSChannelCertStoreName.scsnMY;
oServer.SSLOptions.SChannel_Options.CertStorePath = TsgcSChannelCertStorePath.scspStoreLocalMachine;
oServer.Active = true;
```

Certificato da file PFX

Se si dispone di un file di certificato PFX (.pfx o .p12), impostare le proprietà **CertFile** e **Password** su SSLOptions. SChannel importerà il certificato all'avvio.

```
TsgcWebSocketServer oServer = new TsgcWebSocketServer();
oServer.SSL = true;
oServer.SSLOptions.IOHandler = TsgcSSLIOHandler.iohSChannel;
oServer.SSLOptions.Version = TsgcTLSVersion.tls1_2;
oServer.SSLOptions.Port = 443;
oServer.Port = 443;
oServer.SSLOptions.CertFile = "c:\certificates\server.pfx";
oServer.SSLOptions.Password = "mypassword";
oServer.Active = true;
```

Se si dispone di un certificato PEM e di una chiave privata, convertirli prima nel formato PFX utilizzando OpenSSL:

```
openssl pkcs12 -inkey server.key -in server.crt -export -out server.pfx
```

Versione TLS

Usi la proprietà **SSLOptions.Version** per controllare quale versione TLS il server accetta:

Valore	Descrizione
tls1_0	TLS 1.0 (non raccomandato)
tls1_1	TLS 1.1
tls1_2	TLS 1.2 (consigliato)
tls1_3	TLS 1.3
tlsUndefined	Accetta TLS 1.0, 1.1 e 1.2

Elenco delle cifrature

Per impostazione predefinita SChannel utilizza la configurazione dei cifrari di sistema. È possibile limitare i cifrari consentiti impostando **SChannel_Options.CipherList** su un elenco di nomi di algoritmi separati da due punti, ad esempio:

```
CALG_AES_256:CALG_AES_128
```

Lasciare questa proprietà vuota per utilizzare le impostazioni predefinite di Windows.

Credenziali legacy

Windows Server 2019 e versioni precedenti potrebbero non supportare la nuova API **SCH_CREDENTIALS**. Se il server non si avvia su una versione precedente di Windows, impostare **SChannel_Options.UseLegacyCredentials** su **True** per utilizzare la struttura legacy **SCHANNEL_CRED**.

Il componente rileva automaticamente la versione di Windows nella maggior parte dei casi, ma se necessario può forzare la modalità legacy.

Vantaggi rispetto a OpenSSL

- **Nessuna DLL esterna:** SChannel è integrato in Windows, quindi non è necessario distribuire libeay32.dll / ssleay32.dll o libcrypto / libssl.
- **Integrazione con l'archivio certificati di Windows:** utilizzo dei certificati già installati e gestiti dal sistema operativo.
- **Aggiornamenti automatici:** i miglioramenti TLS e le patch di sicurezza vengono applicati tramite Windows Update.

Note

- SChannel è disponibile **solo su Windows**. Per i server multiplatforma, utilizzi OpenSSL (iohOpenSSL).
- Il server deve avere la **chiave privata** associata al certificato. Quando si utilizza il metodo Windows Store, il certificato deve essere stato importato con la sua chiave privata.
- Per i server di produzione che utilizzano il metodo dell'archivio certificati, si consiglia l'archivio **Local Machine** (scspStoreLocalMachine) in modo che il certificato sia disponibile indipendentemente dall'account utente con cui viene eseguito il servizio.
- Sono supportati solo i file di certificato **PFX** (.pfx / .p12). Se si dispone di file PEM, convertirli prima nel formato PFX.

TsgcWebSocketServer | Verifica certificato server

Per impostazione predefinita, il server non verifica i certificati peer. Per configurare il server in modo che verifichi i certificati client, eseguire i seguenti passaggi:

1. Impostare la proprietà `SSLOptions.VerifyCertificate = true`

Gestire l'evento `OnSSLVerifyPeer` e implementare il codice seguente per ricevere una notifica ogni volta che un client si connette con un certificato.

```
private void OnSSLVerifyPeerEvent(object Sender, TlsX509 Certificate, ref bool Accept)
{
    // ... validate the certificate
    if (Certificate_OK)
        Accept = true;
    else
        Accept = false;
}
```

Si noti che l'evento `OnSSLVerifyPeer` viene **chiamato solo se il client fornisce un certificato**, se un client non fornisce un certificato l'evento non viene attivato.

È possibile configurare il **server in modo da consentire solo connessioni SSL che utilizzano un certificato**. Per farlo, impostare la seguente proprietà:

- `SSLOptions.VerifyCertificate_Options.FailIfNoCertificate = true`

Se il client non fornisce un certificato, la connessione verrà chiusa durante l'handshake SSL.

TsgcWebSocketServer | Mantenere le connessioni attive sul server

Una volta che un client si è connesso al server, a volte la connessione può essere interrotta a causa di un segnale debole, errori di connessione, ecc. Utilizzare HeartBeat per mantenere attiva la connessione.

HeartBeat

La proprietà **HeartBeat** consente di **inviare un Ping** ogni **X secondi** per **mantenere attiva la connessione**. Alcuni client chiudono le connessioni TCP se non vengono scambiati dati tra i peer. HeartBeat risolve questo problema inviando un ping a intervalli specifici. Di solito questo è sufficiente per mantenere attiva una connessione, ma è possibile impostare un intervallo Timeout se si desidera chiudere la connessione quando non si riceve una risposta dal client entro X secondi.

Esempio: inviare un ping a tutti i client connessi ogni 30 secondi

```
oServer = new TsgcWebSocketServer();
oServer.HeartBeat.Interval = 30;
oServer.HeartBeat.Timeout = 0;
oServer.HeartBeat.Enabled = true;
oServer.Active = true;
```


TsgcWebSocketServer | Server Plain TCP

Il server WebSocket accetta WebSocket, HTTP, SSE e altri protocolli, ma può anche gestire connessioni TCP semplici. Per ulteriori informazioni, consultare [Connessioni TCP](#).

Ci sono 2 eventi che possono essere utilizzati per gestire le connessioni TCP.

OnTCPConnect

Questo evento viene chiamato dopo che un client si connette al server e prima di qualsiasi handshake tra client e server. L'evento OnConnect viene attivato solo dopo che il client invia un messaggio (per consentire al server di rilevare quale protocollo viene utilizzato).

Questo evento consente di sapere che un nuovo client sta cercando di connettersi al server, e il server può accettare o rifiutare la connessione. Per impostazione predefinita, il server accetta sempre la connessione.

OnUnknownProtocol

Questo evento viene chiamato quando il server riceve il primo messaggio da un client ma non riesce a rilevare se si tratta di uno dei protocolli noti. In questo evento, il server può accettare o rifiutare il protocollo.

OnConnect

Questo evento viene attivato dopo una connessione avvenuta con successo e completa. Se la connessione è TCP semplice, viene attivata dopo che il protocollo è stato accettato nell'evento OnUnknownProtocol.

TsgcWebSocketServer | Chiusura della connessione del server

Una singola connessione può essere chiusa utilizzando i metodi Close o Disconnect.

Disconnetti

[TsgcWSConnection](#) ha un metodo chiamato `Disconnect()`, che Le consente di disconnettere la connessione a livello di socket. Se chiama questo metodo, il socket verrà disconnesso direttamente senza attendere alcuna risposta dal client. Può inviare un Close Code con questo metodo.

Chiudi

[TsgcWSConnection](#) ha un metodo chiamato `Close()`, che consente di inviare un messaggio al client richiedendo la chiusura della connessione. Se il client riceve questo messaggio, deve chiudere la connessione e il server riceverà una notifica che la connessione è chiusa. È possibile inviare un Close Code con questo metodo.

DisconnectAll

Disconnette tutte le connessioni attive. Questo metodo viene chiamato automaticamente prima che il server smetta di ascoltare, ma può essere chiamato in qualsiasi momento.

TsgcWebSocketServer | Connessioni Client

Per accedere alle connessioni client attive, è possibile utilizzare la proprietà `Connections` per scorrere l'elenco e accedere alla classe di connessione client. La proprietà `Connections` accede a un elenco con thread, quindi è necessario prima bloccare l'elenco e, al termine, sbloccarlo.

```
void DoClientIPAddresses()
{
    TList oList = TsgcWebSocketHTTPServer1.LockList();
    try
    {
        for (int i = 0; i < oList.Count; i++)
        {
            TsgcWSConnectionServer oConnection = (TsgcWSConnectionServer)((TIdContext)oList[i]).Data;
            MessageBox.Show(oConnection.IP + ":" + oConnection.Port.ToString());
        }
    }
    finally
    {
        TsgcWebSocketHTTPServer1.UnlockList();
    }
}
```

TsgcWebSocketServer | Autenticazione del Server

Il server TsgcWebSocket supporta 3 tipi di autenticazione:

- **Basic:** legge un'intestazione HTTP durante l'HandShake WebSocket con Utente e Password codificati come Autorizzazione Basic.
- **Session:** il client richiede prima una sessione HTTP al server e, se il server restituisce una sessione, questa viene passata nell'intestazione HTTP GET dell'handshake WebSocket. (* metodo di autorizzazione proprietario della libreria sgcWebSockets).
- **URL:** legge l'autorizzazione della richiesta utilizzando l'intestazione HTTP GET dell'handshake WebSocket. (* metodo di autorizzazione proprietario per la libreria sgcWebSockets).

È possibile impostare un elenco di utenti autenticati utilizzando la proprietà **AuthUsers**; specificare gli utenti nel seguente formato: user=password

OnAuthentication

Ogni volta che il server riceve una richiesta di autenticazione da un client, questo evento viene chiamato per indicare se l'utente è autenticato o meno.

Usare il parametro **Authenticated** per accettare o rifiutare la connessione.

```
void OnAuthentication(TsgcWSConnection Connection, string aUser, string aPassword,
    ref bool Authenticated)
{
    if ((aUser == "user") && (aPassword == "secret"))
    {
        Authenticated = true;
    }
    else
    {
        Authenticated = false;
    }
}
```

OnUnknownAuthentication

Se il tipo di autenticazione non è supportato per impostazione predefinita, come JWT, è comunque possibile utilizzare questo evento per accettare o rifiutare la connessione. Basta leggere i parametri e decidere se accettare la connessione.

```
void OnUnknownAuthenticationEvent(TsgcWSConnection Connection, string AuthType, string AuthData,
    ref string User, ref string Password, ref bool Authenticated)
{
    if (AuthType == "Bearer")
    {
        if (AuthData == "jwt_token")
        {
            Authenticated = true;
        }
        else
        {
            Authenticated = false;
        }
    }
    else
    {
        Authenticated = false;
    }
}
```

```
} }
```

TsgcWebSocketServer | Invio Messaggi di Testo dal Server

Una volta che il client si è connesso al server, il server può inviare messaggi di testo. Per inviare un messaggio di testo, chiamare il metodo `WriteData()` per inviare un messaggio a un singolo client, oppure utilizzare `Broadcast` per inviare un messaggio a tutti i client.

Invia un messaggio di testo

Chiamare il metodo **`WriteData()`** per inviare un messaggio di testo.

```
TsgcWebSocketServer1.WriteData("guid", "My First sgcWebSockets Message!.");
```

Se **`QueueOptions.Text`** ha un valore diverso da **`qmNone`**, invece di essere elaborato nello stesso thread in cui viene chiamato, verrà elaborato in un thread secondario. Per impostazione predefinita questa opzione è disabilitata.

QueueOptions non funziona se la proprietà `IOHandlerOptions.IOHandlerType = iohIOCP` (a causa dell'architettura IOCP, questa funzionalità non è supportata).

È possibile chiamare anche il metodo `WriteData()` da **`TsgcWSConnection`**, **esempio:** invia un messaggio al client quando si connette al server.

```
void OnConnect(TsgcWSConnection Connection)
{
    Connection.WriteData("Hello From Server");
}
```

Invia un messaggio a TUTTI i client connessi

Chiamare il metodo **`Broadcast()`** per inviare un messaggio di testo a tutti i client connessi.

```
TsgcWebSocketServer1.Broadcast("Hello From Server");
```

TsgcWebSocketServer | Server Send Binary Message

Una volta che il client si è connesso al server, il server può inviare messaggi binari. Per inviare un messaggio binario, chiamare il metodo `WriteData()` per inviare un messaggio a un singolo client, o utilizzare `Broadcast` per inviare un messaggio a tutti i client.

Invia un messaggio binario

Richiamare il metodo **`WriteData()`** per inviare un messaggio binario.

```
TsgcWebSocketServer1.WriteData("guid", new MemoryStream());
```

Se **`QueueOptions.Binary`** ha un **valore diverso da `qmNone`**, invece di essere elaborato sullo stesso thread che lo chiama, verrà elaborato su un thread secondario. Per impostazione predefinita questa opzione è disabilitata.

QueueOptions non funziona se la proprietà `IOHandlerOptions.IOHandlerType = iohIOCP` (a causa dell'architettura IOCP, questa funzionalità non è supportata).

È possibile chiamare anche il metodo `WriteData()` da **`TsgcWSConnection`**, **esempio:** invia un messaggio al client quando si connette al server.

```
void OnConnect(TsgcWSConnection Connection)
{
    Connection.WriteData(new MemoryStream());
}
```

Invia un messaggio a TUTTI i client connessi

Chiamare il metodo **`Broadcast()`** per inviare un messaggio binario a tutti i client connessi.

```
TsgcWebSocketServer1.Broadcast(new MemoryStream());
```

TsgcWebSocketServer | Ricezione Messaggio di Testo dal Server

Quando il server riceve un messaggio di testo, viene generato l'evento **OnMessage** . Legga il parametro Text per recuperare la stringa del messaggio ricevuto.

```
void OnMessage(TsgcWSConnection Connection, string Text)
{
    MessageBox.Show("Message Received from Client: " + Text);
}
```

Per impostazione predefinita, il server utilizza il metodo **neAsynchronous** per generare l'evento OnMessage, il che significa che **questo evento viene eseguito nel contesto del Thread Principale**, quindi è thread-safe aggiornare qualsiasi controllo di un form, ad esempio.

Se il server riceve molti messaggi o si ha la necessità di controllare la sincronizzazione con altri thread, impostare la proprietà NotifyEvents su **neNoSync**; ciò significa che l'evento OnMessage verrà **eseguito nel contesto del thread di connessione**; pertanto, se si desidera aggiornare qualsiasi controllo di un form o accedere a oggetti condivisi, è necessario implementare i propri metodi di sincronizzazione.

TsgcWebSocketServer | Ricezione messaggio binario dal server

Quando il server riceve un messaggio binario, viene generato l'evento **OnBinary**. Leggere il parametro Data per recuperare il messaggio binario ricevuto.

```
private void OnBinary(TsgcWSConnection Connection, byte[] Bytes)
{
    MemoryStream stream = new MemoryStream(Bytes);
    pictureBox1.Image = new Bitmap(stream);
}
```

Per impostazione predefinita, il server utilizza il metodo **neAsynchronous** per generare l'evento OnMessage, il che significa che **questo evento viene eseguito nel contesto del Thread Principale**, quindi è thread-safe aggiornare qualsiasi controllo di un form, ad esempio.

Se il server riceve molti messaggi o si ha la necessità di controllare la sincronizzazione con altri thread, impostare la proprietà NotifyEvents su **neNoSync**; ciò significa che l'evento OnMessage verrà **eseguito nel contesto del thread di connessione**; pertanto, se si desidera aggiornare qualsiasi controllo di un form o accedere a oggetti condivisi, è necessario implementare i propri metodi di sincronizzazione.

TsgcWebSocketServer | Lettura delle Intestazioni del Client dal Server

Quando un **client si connette** al server WebSocket, invia un elenco di **intestazioni** con informazioni sulla connessione del client. Per leggere queste intestazioni client, è possibile utilizzare l'evento **OnHandshake** del componente server, che viene chiamato quando il server riceve le intestazioni dal client e prima di inviare una risposta. Le intestazioni del client sono memorizzate nella proprietà **HeadersRequest** di **TsgcWSConnectionServer**.

```
void OnServerHandshake(TsgcWSConnection Connection, TStringList Headers)
{
    MessageBox.Show(Headers.Text);
}
```

TsgcWebSocketHTTPServer

AResponseInfo.

Introduzione

TsgcWebSocketHTTPServer implementa il componente Server WebSocket e può gestire connessioni client multi-thread come [TsgcWebSocketServer](#), e consente di servire pagine HTML tramite un server HTTP integrato, condividendo la stessa porta per le connessioni WebSocket e le richieste HTTP.

Segua i passi seguenti per configurare questo componente:

1. Inserire un componente TsgcWebSocketHTTPServer nel form
2. Impostare la porta (il valore predefinito è 80). Se si è protetti da un firewall, probabilmente sarà necessario configurarlo.
3. Impostare le specifiche consentite; per impostazione predefinita, tutte le specifiche sono consentite.

RFC6455: è la specifica WebSocket standard e raccomandata.

Hixie76: è una bozza ed è consigliato stabilire connessioni Hixie76 solo se si desidera fornire supporto per browser obsoleti come Safari 4.2

```
AResponseInfo.ContentText := '<HTML><HEADER>TEST</HEAD><BODY>Hello!</BODY></HTML>';
```

OnHTTPUploadBeforeCreatePostStream: questo evento viene chiamato dopo che le intestazioni sono state lette e prima che il post stream venga creato.

* In alcuni casi, si potrebbe riscontrare un elevato consumo di CPU dovuto a connessioni non richieste; in tal caso, è sufficiente restituire un errore 500 per le richieste HTTP o chiudere la connessione per le richieste con protocollo sconosciuto.

5. Creare una procedura e impostare la proprietà Active = true.

Riferimento

- Proprietà — 33 proprietà disponibili sul componente.
- Metodi — 9 metodi pubblici.
- Eventi — 36 eventi gestibili.

Guide

- HTTP
 - [Richieste Server HTTP](#)
 - [Distribuzione File HTTP](#)
 - [Server HTTP/2](#)
 - [Server Push HTTP/2](#)
 - [Servizio Alternativo HTTP/2](#)
 - [Thread Server HTTP/2](#)
 - [HTTP Post File di Grandi Dimensioni](#)

- **SSL**
 - [Server SSL SChannel](#)

TsgcWebSocketHTTPServer | Richieste HTTP Server

Utilizzare `OnCommandGet` per gestire le richieste client HTTP. Utilizzare i seguenti parametri:

- **RequestInfo:** contiene le informazioni della richiesta HTTP.
- **ResponseInfo:** è la risposta HTTP alla richiesta HTTP.
 - **ContentText:** è la risposta in formato testo.
 - **ContentType:** è il tipo di Content-Type.
 - **ResponseNo:** numero della risposta HTTP, esempio: 200.

```
void OnCommandGet(TsgcWSConnection Connection, TsgcWSHTTPRequestInfo RequestInfo,
  ref TsgcWSHTTPResponseInfo ResponseInfo)
{
  if (RequestInfo.Document == "/")
  {
    ResponseInfo.ContentText = "<html><head><title>Test Page</title></head><body></body></html>";
    ResponseInfo.ContentType = "text/html";
    ResponseInfo.ResponseNo = 200;
  }
}
```

OnBeforeCommand

Utilizzare questo evento per personalizzare la risposta HTTP. Ad esempio, se si desidera che alcuni endpoint utilizzino uno schema di autorizzazione mentre altri sono accessibili senza autorizzazione, utilizzare il parametro `options` per abilitarlo o disabilitarlo. Di seguito è riportato un esempio in cui l'autorizzazione Basic è abilitata, ma quando un utente richiede l'endpoint `/public`, l'autorizzazione non è richiesta.

```
public void OnBeforeCommand(TsgcWSConnection aConnection, TIdHTTPRequestInfo ARequestInfo, TIdHTTPResponseInfo AResponseInfo)
{
  if (ARequestInfo.Document == "/public")
    aOptions = TsgcHTTPCommandOptions.hcoAuthorizedBasic;
}
```

TsgcWebSocketHTTPServer | HTTP Dispatch Files

Quando un client richiede un file, l'evento **OnCommandGet** viene attivato, ma è possibile utilizzare la proprietà **DocumentRoot** per distribuire i file automaticamente.

Esempio: se si imposta **DocumentRoot** su **c:/www/files**. Ogni volta che viene richiesto un nuovo file, il server lo cercherà in questa cartella. Se il file esiste, verrà inviato automaticamente.

TsgcWebSocketHTTPServer | HTTP/2 Server

Il server HTTP di `sgcWebSockets` consente di gestire richieste HTTP/1.1 e HTTP/2.0; è possibile abilitare il protocollo HTTP/2 utilizzando `HTTPOptions` del server.

Imposti **`HTTP2Options.Enabled = true`** per consentire al server di accettare richieste del protocollo HTTP/2. Le richieste possono essere elaborate dall'utente esattamente come con il protocollo HTTP/1.1, [legga di più](#).

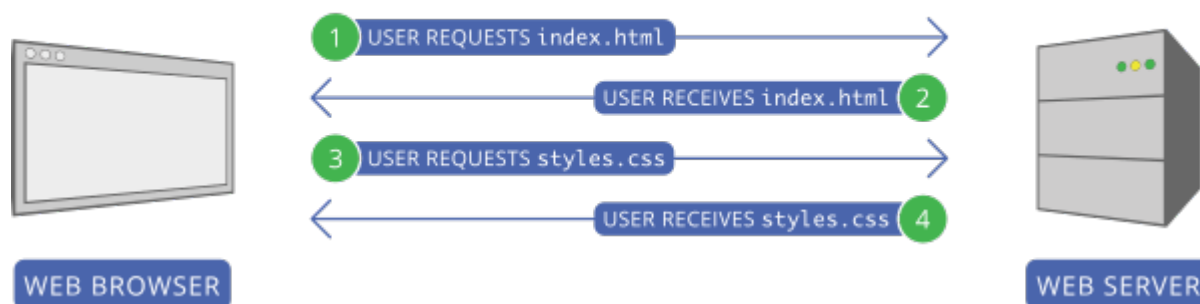
Quando il protocollo HTTP/2 è abilitato, il server supporterà comunque le richieste HTTP/1.1.

Per impostazione predefinita, gli eventi `OnConnect` e `OnDisconnect` non verranno chiamati quando viene stabilita una nuova connessione HTTP/2, ma questo può essere modificato accedendo alle proprietà `HTTP2Options.Events`, dove è possibile personalizzare se si desidera essere notificati ogni volta che viene stabilita o interrotta una nuova connessione HTTP/2.

TsgcWebSocketHTTPServer | HTTP/2 Server Push

HTTP funziona solitamente con un pattern Richiesta/Risposta, in cui il client invia una RICHIESTA per una risorsa al SERVER, e il SERVER invia una RISPOSTA con la risorsa richiesta o un errore. Di solito il client, come un browser, effettua una serie di richieste per risorse fornite dal server.

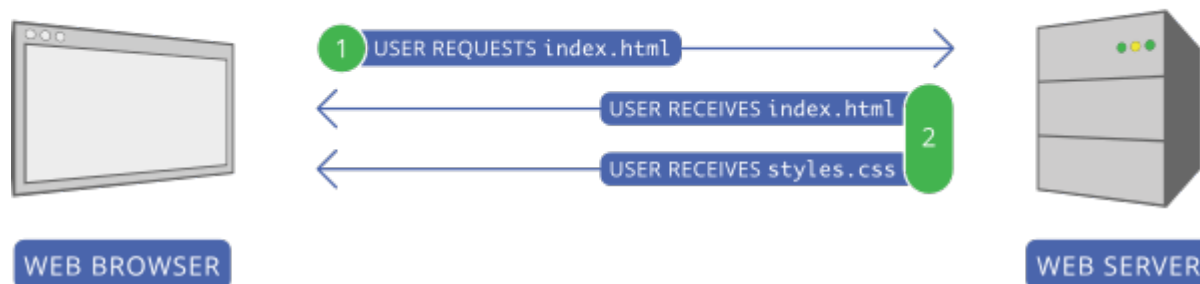
TYPICAL WEB SERVER COMMUNICATION



Il problema principale di questo approccio è che il client deve prima inviare una richiesta per ottenere la risorsa (ad esempio, index.html), attendere che il server invii la risposta, leggerne il contenuto e poi effettuare tutte le altre richieste (ad esempio, styles.css).

Il server push HTTP/2 cerca di risolvere questo problema. Quando il client richiede un file, se il server determina che questo file richiede file aggiuntivi, quei file verranno inviati automaticamente al client tramite PUSH.

WEB SERVER COMMUNICATION WITH HTTP/2 SERVER PUSH



Nello screenshot sopra, il client richiede prima index.html. Il server legge questa richiesta e invia 2 file come risposta: index.html e styles.css, evitando così una seconda richiesta per ottenere styles.css.

Configurare Server Push

In base agli screenshot precedenti, è possibile configurare il server in modo che ogni volta che arriva una nuova richiesta per il file /index.html, il server invii index.html e styles.css.

Utilizzi il metodo **PushPromiseAddPreLoadLinks** per associare ogni richiesta a un elenco di push promise.

```

TsgcWebSocketHTTPServer server = new TsgcWebSocketHTTPServer(this);
server.PushPromiseAddPreLoadLinks("/index.html", "/styles.css");
void OnCommandGet(TsgcWSConnection Connection, TsgcWSHTTPRequestInfo RequestInfo, ref TsgcWSHTTPResponseInfo Resp
{
    if (RequestInfo.Document == "/index.html")
    {

```

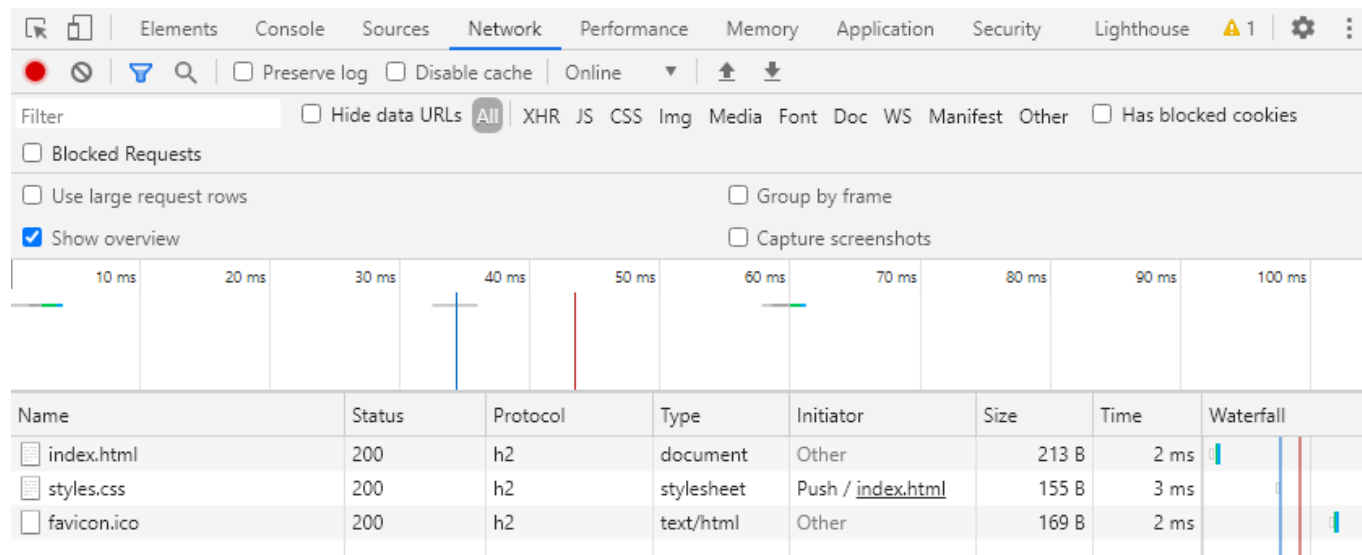


```

    ResponseInfo.ContentText = "";
    ResponseInfo.ContentType = "text/html";
    ResponseInfo.ResponseNo = 200;
}
else if (RequestInfo.Document == "/styles.css")
{
    ResponseInfo.ContentText = "";
    ResponseInfo.ContentType = "text/css";
    ResponseInfo.ResponseNo = 200;
}
}

```

Utilizzando lo strumento per sviluppatori di Chrome, è possibile verificare come il file styles.css viene inviato in push al client.



TsgcWebSocketHTTPServer | HTTP/2 Alternate Service

L'intestazione HTTP **Alt-Svc** viene utilizzata per **informare** i client che la **stessa risorsa è raggiungibile da un altro servizio o protocollo**; ciò è utile se si desidera informare i client HTTP che il proprio server supporta HTTP/2, ad esempio.

Esempio: se il server è in esecuzione su un IP locale 127.0.0.1 e ascolta su 2 porte: 80 (non cifrata) e 443 (cifrata). È possibile informare i client che HTTP/2 è supportato sulla porta 443 utilizzando il seguente header HTTP

```
Alt-Svc: h2=":443"
```

Quando HTTP/2 è abilitato, questo header viene aggiunto automaticamente se la connessione non è in esecuzione sul protocollo HTTP/2.

È possibile abilitare o disabilitare questa funzione usando la proprietà **HTTP2Options.AltSvc**.

TsgcWebSocketHTTPServer | HTTP/2 Server Threads

Di seguito sono riportate le differenze tra HTTP 1.1 e HTTP 2.0:

HTTP 1.1

Nel comportamento HTTP tradizionale, quando si effettuano più richieste sulla stessa connessione, il client deve attendere la risposta di ogni richiesta prima di inviare la successiva. Questo approccio sequenziale aumenta significativamente il tempo di caricamento delle risorse di un sito web. Per risolvere questo problema, HTTP/1.1 ha introdotto una funzionalità chiamata **pipelining**, che consente a un client di inviare più richieste senza attendere le risposte del server. Il server, a sua volta, risponde al client nello stesso ordine in cui ha ricevuto le richieste.

Sebbene il **pipelining** sembrasse una soluzione, ha incontrato alcune difficoltà:

- **Ignoranza del Server o Corruzione della Risposta:** Alcuni server ignoravano le richieste in pipeline o corrompevano le risposte, portando a una comunicazione inaffidabile.
- **Head-of-Line Blocking:** La prima richiesta nella pipeline potrebbe bloccare le richieste successive, causando un ritardo nell'elaborazione delle altre richieste. Questo fenomeno, noto come **head-of-line blocking**, ha comportato tempi di caricamento delle pagine più lenti.

Nel tentativo di ottimizzare il caricamento delle pagine dai server che supportano HTTP/1.1, i browser web hanno implementato una soluzione alternativa. Aprono da sei a otto connessioni parallele al server, consentendo la trasmissione simultanea di più richieste. Questo parallelismo mira a mitigare i problemi associati al **pipelining** e a migliorare i tempi complessivi di caricamento delle pagine.

La scelta di sei-otto connessioni parallele da parte dei browser web si basa su considerazioni di ottimizzazione. Le ragioni specifiche alla base di questa scelta potrebbero implicare un compromesso tra utilizzo delle risorse, efficienza della rete e prevenzione di potenziali colli di bottiglia.

HTTP 2.0

In risposta ai vincoli incontrati nel **pipelining**, HTTP/2 ha introdotto una funzionalità chiamata **multiplexing**. Il **multiplexing** consente una **comunicazione più efficiente** tra client e server, abilitando la **trasmissione concorrente di più richieste** e risposte **su una singola connessione**.

HTTP/2 utilizza un meccanismo di **framing binario**, il che significa che i messaggi HTTP vengono suddivisi in unità più piccole e indipendenti chiamate **frame**. Questi **frame** possono essere intercalati e inviati sulla connessione indipendentemente l'uno dall'altro. Sul lato ricevente, i **frame** vengono riassemblati per ricostruire il messaggio HTTP originale.

Questo meccanismo di **framing binario** è fondamentale per realizzare il **multiplexing** in HTTP/2. Consente al browser di inviare più richieste sulla stessa connessione senza incorrere in problemi di blocco. Di conseguenza, browser come Chrome utilizzano lo stesso ID di connessione per le richieste HTTP/2, permettendo una comunicazione efficiente e ininterrotta tra client e server.

In sostanza, la funzionalità di **multiplexing** di HTTP/2, abilitata dal meccanismo di **framing binario**, migliora l'efficienza e la velocità dello scambio di dati tra client e server facilitando la trasmissione simultanea di più richieste e risposte su una singola connessione.

TsgcWebSocketHTTPServer

Per migliorare le prestazioni del protocollo HTTP/2, le richieste vengono distribuite per impostazione predefinita in un pool di thread (32 per impostazione predefinita) ogni volta che il server riceve una nuova richiesta HTTP/2. Ciò evita attese quando una singola connessione invia molte richieste concorrenti che richiederebbero un'elaborazione sequenziale (nel contesto del thread di connessione) in assenza di questo pool di thread.

Il comportamento del pool di thread può essere configurato con le seguenti proprietà.

- **HTTP2Options.PoolOfThreads.Enabled:** (per impostazione predefinita false) abilitare questa opzione per distribuire le richieste HTTP/2 nel pool di thread anziché nel thread della connessione.
- **HTTP2Options.Threads:** (per impostazione predefinita 32) il numero di thread utilizzati per gestire le richieste HTTP/2. Imposti un numero in base al numero di processori sul Suo server.

Per **ottimizzare le richieste**, selezionando quali richieste devono essere elaborate nel pool di thread (perché richiedono molto tempo) mentre altre possono essere elaborate nel thread di connessione, può utilizzare l'evento **OnHttp2BeforeAsyncRequest**. Questo evento viene generato prima di accodare la richiesta nel pool di thread. Utilizzi il parametro **Async** per impostare se la richiesta deve essere eseguita su thread o meno.

```
void OnHTTP2BeforeAsyncRequest(object Sender, TsgcWSConnection Connection, TIdHTTPRequestInfo ARequestInfo, ref bool Async)
{
    if (ARequestInfo.Document == "/fast-request")
        Async = false;
}
```

TsgcWebSocketHTTPServer | Errore 404 senza corpo della risposta

Per impostazione predefinita, la libreria Indy aggiunge un corpo al contenuto delle risposte HTTP se non è assegnato `ContentText` o `ContentStream`. Se si desidera restituire un corpo di risposta vuoto (per un errore 404 o simile), è possibile utilizzare il seguente approccio.

Crei un nuovo `TStringStream` senza contenuto e lo assegni alla proprietà `ContentStream` della `Response HTTP`. In questo modo la risposta HTTP verrà inviata senza i tag HTML predefiniti.

Esempio

TsgcWebSocketHTTPServer | Sessioni

HTTP è un protocollo senza stato (almeno fino a HTTP 1.1), quindi il client richiede un file, il server invia una risposta e la connessione viene chiusa (è possibile abilitare il keep-alive in modo che la connessione non venga chiusa immediatamente, ma ciò va oltre lo scopo di questo articolo). Le sessioni consentono di memorizzare informazioni sul client, che possono essere utilizzate durante un login del client, ad esempio. È possibile utilizzare qualsiasi ID sessione univoco, cercare nell'elenco delle sessioni se ne esiste già una e, in caso contrario, crearne una nuova. Una sessione può essere distrutta dopo un periodo di inattività o manualmente dopo il logout del client.

Configurazione

Esistono alcune proprietà in TsgcWebSocketHTTPServer che abilitano/disabilitano le sessioni nel componente server. Le più importanti sono:

Property	Descrizione
SessionState	Questa è la prima proprietà che deve essere abilitata per poter utilizzare le Sessioni. Se questa proprietà è abilitata, le sessioni non funzioneranno
SessionTimeout	Qui è necessario impostare un valore maggiore di zero (in millisecondi) per il tempo massimo di attività di una sessione.
AutoStartSession	Le sessioni possono essere create automaticamente (AutoStartSession = true) o manualmente (AutoStartSession = false). Se le sessioni vengono create automaticamente, il server utilizza motelP come identificatore univoco per verificare se è presente una sessione attiva memorizzata.

```
TsgcWebSocketHTTPServer1.SessionState = true;
TsgcWebSocketHTTPServer1.SessionTimeout = 600000;
AutoStartSession = false;
```

Crea sessione

Per creare una nuova sessione, è necessario creare un nuovo **ID sessione** che sia **univoco**. È possibile utilizzare qualsiasi valore. **Esempio:** se il client si sta autenticando, è possibile utilizzare utente + password + IP remoto come ID sessione.

Quindi, cercare nell'elenco delle sessioni se esiste già. Se non esiste, crearne una nuova.

Quando viene creata una nuova sessione viene chiamato l'evento **OnSessionStart** e quando la sessione viene chiusa viene generato l'evento **OnSessionEnd**.

```
void OnCommandGet(object sender, TsgcHttpRequestEventArgs e)
{
    if (e.RequestInfo.Document == "/")
    {
        e.ResponseInfo.ServeFile("yourpathhere\\index.html");
    }
    else
    {
        // check if user is valid
        if (!(e.RequestInfo.AuthUsername == "user") && (e.RequestInfo.AuthPassword == "pass"))
        {
            e.ResponseInfo.AuthRealm = "Authenticate";
        }
        else
        {
            // create session
            string sessionId = e.RequestInfo.AuthUsername + e.RequestInfo.AuthPassword + e.RequestInfo.RemoteIP;
            Session session = new Session(sessionId, e.RequestInfo.AuthUsername, e.RequestInfo.AuthPassword);
            SessionList.Add(session);
        }
    }
}
```

```

// create a new session id with authentication data
string vID = e.RequestInfo.AuthUsername + "_" + e.RequestInfo.AuthPassword + "_" + e.RequestInfo.RemoteIP;
// search session
var oSession = TsgcWebSocketHTTPServer1.SessionList.GetSession(vID, e.RequestInfo.RemoteIP);
// create new session if not exists
if (oSession == null)
    oSession = TsgcWebSocketHTTPServer1.SessionList.CreateSession(e.RequestInfo.RemoteIP, vID);
e.ResponseInfo.ContentText = "<html><head></head><body>Authenticated</body></html>";
e.ResponseInfo.ResponseNo = 200;
    }
}
}

```

TsgcWebSocketHTTPServer | Stream Video

Per trasmettere in streaming un file video tramite il server, è possibile utilizzare la funzione `IndyStreamFileVideo` per trasmettere il file tramite la codifica di trasferimento a blocchi.

La funzione accetta i seguenti parametri:

- **AContext**: viene preso dall'evento `OnCommandGet`.
- **AResponseInfo**: viene recuperato dall'evento `OnCommandGet`.
- **aFileName**: è il percorso completo del file video da trasmettere in streaming.
- **ContentType**: per impostazione predefinita è "video/mpeg".
- **aBufferSize**: il valore predefinito è 1024 byte.

Esempio: trasmettere in streaming un video quando un utente accede al documento `/video.mp4` e questo video si trova nella cartella `c:\videos\video.mp4`

```
void OnServerCommandGet(TidContext AContext, TidHttpRequestInfo ARequestInfo, TidHttpResponseInfo AResponseInfo)
{
    if (ARequestInfo.Document == "/video.mp4")
    {
        IndyStreamFileVideo(AContext, AResponseInfo, "C:\\videos\\video.mp4");
    }
    else
    {
        AResponseInfo.ResponseNo = 404;
    }
}
```


Server SSL | SChannel per server Indy

I componenti server basati su Indy (**TsgcWebSocketServer**, **TsgcWebsocketHTTPServer**) possono utilizzare **Windows SChannel** (Secure Channel) come provider TLS invece di OpenSSL. SChannel è l'implementazione TLS nativa di Windows, quindi non richiede alcuna DLL esterna.

Come funziona

Quando un componente server ha SSL abilitato e l'IOHandler è impostato su **iohSChannel**, il server crea un'istanza di **TsgcIdServerIOHandlerSSLSChannel** che gestisce tutte le operazioni TLS tramite l'API Windows SChannel. Per ogni connessione client in entrata, il server esegue l'handshake TLS tramite il provider SChannel, negoziando la versione del protocollo, la suite di cifratura e associando il certificato configurato.

SChannel legge i certificati dal **Windows Certificate Store** o da un file **PFX** (.pfx / .p12). Non sono necessari file PEM e non è necessario distribuire librerie OpenSSL.

Configurazione

Per abilitare SChannel su un server Indy, configuri le seguenti proprietà:

1. Impostare la proprietà **SSL** su **True**.
2. Impostare **SSLOptions.IOHandler** su **iohSChannel**.
3. Imposti **SSLOptions.Version** alla versione TLS desiderata (tls1_2, tls1_3, ...).
4. Impostare **SSLOptions.Port** sulla porta utilizzata per le connessioni sicure.
5. Configurare il certificato utilizzando uno dei due metodi descritti di seguito.

Proprietà di SChannel_Options

La sotto-proprietà **SSLOptions.SChannel_Options** espone le impostazioni specifiche di SChannel:

Proprietà	Descrizione
CertHash	L'impronta digitale (hash esadecimale) di un certificato installato nell'archivio certificati di Windows.
CertStoreName	Quale archivio di certificati cercare: scsnMY (Personale), scsnRoot , scsnTrust , scsnCA .
CertStorePath	Posizione dell'archivio: scspStoreLocalMachine o scspStoreCurrentUser .
CipherList	Elenco facoltativo separato da due punti di algoritmi di cifratura (ad es. CALG_AES_256:CALG_AES_128). Vuoto significa utilizzare le impostazioni predefinite del sistema.
UseLegacyCredentials	Se True, utilizza la struttura SCHANNEL_CRED legacy invece di SCH_CREDENTIALS. Abilitare questa opzione per le versioni precedenti di Windows che non supportano la nuova API.

Inoltre, le proprietà generali di **SSLOptions CertFile** e **Password** vengono utilizzate durante il caricamento di un certificato da un file PFX.

Certificato da Windows Store

Se il certificato è già installato nell'archivio certificati di Windows, fornire l'**impronta digitale** del certificato e indicare dove si trova.

Per trovare il thumbprint, aprire **PowerShell** ed eseguire:

```
dir cert:\localmachine\my
```

La colonna **Thumbprint** mostra l'hash esadecimale necessario.

```
Directory: Microsoft.PowerShell.Security\Certificate::localmachine\my
Thumbprint                               Subject
-----
C12A8FC8AE668F866B48F23E753C93D357E9BE10  CN=*.mydomain.com
```

```
TsgcWebSocketHTTPServer oServer = new TsgcWebSocketHTTPServer();
oServer.SSL = true;
oServer.SSLOptions.IOHandler = TsgcSSLIHandler.iohSChannel;
oServer.SSLOptions.Version = TsgcTLSVersion.tls1_2;
oServer.SSLOptions.Port = 443;
oServer.Port = 443;
oServer.SSLOptions.SChannel_Options.CertHash = "C12A8FC8AE668F866B48F23E753C93D357E9BE10";
oServer.SSLOptions.SChannel_Options.CertStoreName = TsgcSChannelCertStoreName.scnMY;
oServer.SSLOptions.SChannel_Options.CertStorePath = TsgcSChannelCertStorePath.scspStoreLocalMachine;
oServer.Active = true;
```

Certificato da file PFX

Se si dispone di un file di certificato PFX (.pfx o .p12), impostare le proprietà **CertFile** e **Password** su SSLOptions. SChannel importerà il certificato all'avvio.

```
TsgcWebSocketHTTPServer oServer = new TsgcWebSocketHTTPServer();
oServer.SSL = true;
oServer.SSLOptions.IOHandler = TsgcSSLIHandler.iohSChannel;
oServer.SSLOptions.Version = TsgcTLSVersion.tls1_2;
oServer.SSLOptions.Port = 443;
oServer.Port = 443;
oServer.SSLOptions.CertFile = "c:\certificates\server.pfx";
oServer.SSLOptions.Password = "mypassword";
oServer.Active = true;
```

Se si dispone di un certificato PEM e di una chiave privata, convertirli prima nel formato PFX utilizzando OpenSSL:

```
openssl pkcs12 -inkey server.key -in server.crt -export -out server.pfx
```

Versione TLS

Usi la proprietà **SSLOptions.Version** per controllare quale versione TLS il server accetta:

Valore	Descrizione
tls1_0	TLS 1.0 (non raccomandato)
tls1_1	TLS 1.1
tls1_2	TLS 1.2 (consigliato)
tls1_3	TLS 1.3
tlsUndefined	Accetta TLS 1.0, 1.1 e 1.2

Elenco delle cifrature

Per impostazione predefinita SChannel utilizza la configurazione dei cifrari di sistema. È possibile limitare i cifrari consentiti impostando **SChannel_Options.CipherList** su un elenco di nomi di algoritmi separati da due punti, ad esempio:

```
CALG_AES_256:CALG_AES_128
```

Lasciare questa proprietà vuota per utilizzare le impostazioni predefinite di Windows.

Credenziali legacy

Windows Server 2019 e versioni precedenti potrebbero non supportare la nuova API **SCH_CREDENTIALS**. Se il server non si avvia su una versione precedente di Windows, impostare **SChannel.Options.UseLegacyCredentials** su **True** per utilizzare la struttura legacy **SCHANNEL_CRED**.

Il componente rileva automaticamente la versione di Windows nella maggior parte dei casi, ma se necessario può forzare la modalità legacy.

Vantaggi rispetto a OpenSSL

- **Nessuna DLL esterna:** SChannel è integrato in Windows, quindi non è necessario distribuire libeay32.dll / ssleay32.dll o libcrypto / libssl.
- **Integrazione con l'archivio certificati di Windows:** utilizzo dei certificati già installati e gestiti dal sistema operativo.
- **Aggiornamenti automatici:** i miglioramenti TLS e le patch di sicurezza vengono applicati tramite Windows Update.

Note

- SChannel è disponibile **solo su Windows**. Per i server multiplatforma, utilizzi OpenSSL (iohOpenSSL).
- Il server deve avere la **chiave privata** associata al certificato. Quando si utilizza il metodo Windows Store, il certificato deve essere stato importato con la sua chiave privata.
- Per i server di produzione che utilizzano il metodo dell'archivio certificati, si consiglia l'archivio **Local Machine** (scspStoreLocalMachine) in modo che il certificato sia disponibile indipendentemente dall'account utente con cui viene eseguito il servizio.
- Sono supportati solo i file di certificato **PFX** (.pfx / .p12). Se si dispone di file PEM, convertirli prima nel formato PFX.

TsgcWebSocketServer_HTTPAPI

L'API HTTP Server consente alle applicazioni di comunicare tramite HTTP senza utilizzare Microsoft Internet Information Server (IIS).

Introduzione

L'API HTTP Server consente alle applicazioni di comunicare tramite HTTP senza utilizzare Microsoft Internet Information Server (IIS). Le applicazioni possono registrarsi per ricevere richieste HTTP per URL specifici, ricevere richieste WebSocket e inviare risposte WebSocket. L'API HTTP Server include il supporto SSL affinché le applicazioni possano scambiare dati su connessioni HTTP sicure senza IIS. È inoltre progettata per funzionare con le porte di completamento I/O.

Il server supporta i seguenti protocolli:

- WebSockets (richiede Windows 8 o versioni successive)
- HTTP 1.1
- HTTP/2 (Richiede Windows 2016+ o Windows 10+).

Per impostazione predefinita, questo componente richiede che l'applicazione venga eseguita in modalità Amministratore per la registrazione degli URL. Se l'URL è già stato registrato con uno strumento esterno come netsh, è possibile eseguire senza diritti di amministratore: disabilitare la proprietà BindingOptions.ConfigureSSLCertificate per consentire l'avvio dell'applicazione senza diritti di amministratore.

Impostare FastMM4/FastMM5 come prima unità del proprio progetto.

Segua i passi seguenti per configurare questo componente:

1. Inserire nel form un componente TsgcWebSocketServer_HTTPAPI
2. Definire l'indirizzo e la porta di ascolto:

```
Server.Host = "127.0.0.1";
Server.Port = 80;
```

3. Impostare le specifiche consentite; per impostazione predefinita, tutte le specifiche sono consentite.

RFC6455: è la specifica WebSocket standard e raccomandata.

Hixie76: è una bozza ed è consigliato stabilire connessioni Hixie76 solo se si desidera fornire supporto per browser obsoleti come Safari 4.2

5. Creare una procedura e impostare la proprietà Active = true

Prenotazione URL

Il server HTTP.SYS utilizza la prenotazione degli URL per assegnare quali endpoint URL verranno utilizzati dal server HTTP.SYS.

Prenotazione URL di base

Questa è la modalità più semplice per configurare il Server: è sufficiente impostare l'Host e la Porta che il server HTTP.SYS gestirà.

Esempio: se il server è in esecuzione sull'IP 127.0.0.1 e sulla Porta 80, impostare le seguenti proprietà

```
Server.Host = "127.0.0.1";
Server.Port = 80;
```

Se il server è in esecuzione su più di un IP e si desidera associarsi a più IP, utilizzare il metodo **NewBinding**. Per prima cosa azzerare le proprietà Host e Bindings, quindi utilizzare il metodo NewBinding per definire tutti i Binding del Server.

```
Server.Host = "";
Server.Bindings.Clear();
Server.Bindings.NewBinding("127.0.0.1", 80, '');
Server.Bindings.NewBinding("80.50.55.11", 80, '');
```

Se il server richiede connessioni SSL, eseguire le seguenti operazioni per definire l'Host e la Porta che verranno utilizzati per gestire le connessioni SSL.

```
Server.Host := '127.0.0.1';
Server.Port := 443;
Server.SSL := true;
Server.SSLOptions.Hash := "CERTIFICATE_HASH";
```

Se il server richiede connessioni SSL con più indirizzi IP, prima cancellare la proprietà Host e Bindings, quindi registrare i nuovi Bindings.

```
Server.Host = "";
Server.Bindings.Clear();
Server.Bindings.NewBinding("127.0.0.1", 443, '', true, "CERTIFICATE_HASH1");
Server.Bindings.NewBinding("80.50.55.11", 443, '', true, "CERTIFICATE_HASH2");
```

Riferimento

- Proprietà — 27 proprietà disponibili nel componente.
- Metodi — 9 metodi pubblici.
- Eventi — 23 eventi gestibili.

Guide

- **Configurazione**
 - [Prenotazione URL](#)
- **Connessione**
 - [OnDisconnect non attivato](#)
- **SSL**
 - [HTTPAPI Server SSL](#)
 - [Self-Signed Certificates](#)
- **HTTP**
 - [Custom Headers](#)
 - [Send Text Response](#)
 - [Send File Response](#)
 - [Post Big Files](#)
- **HTTP/2**
 - [Disabilita HTTP/2](#)

HTTPAPI | Prenotazione URL

La prenotazione URL HTTP.SYS è una funzionalità del sistema operativo Windows che consente a un utente di riservare un URL specifico per la propria applicazione o servizio. Quando un URL viene riservato tramite HTTP.SYS, il sistema operativo intercetta tutte le richieste HTTP in arrivo per quell'URL e le instrada verso l'applicazione o il servizio specificato.

Per riservare un URL usando HTTP.SYS, un'applicazione o un servizio deve prima registrare l'URL con il driver HTTP.SYS effettuando una chiamata all'API HTTP. L'applicazione o il servizio specifica l'URL, il metodo HTTP (ad es. GET, POST) e qualsiasi impostazione aggiuntiva come i requisiti di autenticazione.

Una volta registrato l'URL, HTTP.SYS intercetterà le richieste HTTP in arrivo per quell'URL e cercherà l'applicazione o il servizio registrato in base all'URL e al metodo. Se viene trovata un'applicazione o un servizio corrispondente, il driver HTTP.SYS passerà la richiesta a quell'applicazione o servizio per l'elaborazione.

Comandi NETSH

Registrare un URL

In questo esempio, l'URL `http://example.com:80/` viene registrato per l'utente `DOMAIN\user`. È possibile sostituirlo con l'URL e l'utente desiderati.

```
netsh http add urlacl url=http://example.com:80/ user=DOMAIN\user
```

Elimina un URL

In questo esempio viene eliminato l'URL `http://example.com:80/`. È possibile sostituirlo con l'URL che si desidera eliminare.

```
netsh http delete urlacl url=http://example.com:80/
```

Mostra tutti gli URL

Questo comando visualizza un elenco di tutte le prenotazioni URL registrate nel sistema.

```
netsh http show urlacl
```

TsgcWebSocketServer_HTTPAPI

Il server HTTP.SYS registra automaticamente gli URL al suo avvio. Questo viene effettuato usando i seguenti parametri e metodi.

- **Host e Port:** se Host non è vuoto e Port è diverso da zero, il server tenterà di registrare l'URL. Esempio: l'URL `https://127.0.0.1:5000` verrà registrato utilizzando le seguenti proprietà
 - Host = '127.0.0.1';
 - Port = 5000
 - SSL = True
- **NewBinding:** utilizzare questo metodo per registrare uno o più URL.
 - Registrare l'url `https://127.0.0.1:5000` --> `NewBinding('127.0.0.1', 5000, '/', True)`
 - Registrare l'url `http://+:5000/ws/` --> `NewBinding('+', 5000, '/ws/')`

La registrazione dell'URL richiede privilegi di amministratore nei seguenti casi:

- Il numero di porta è inferiore a 1024
- L'host è un carattere jolly "+", anziché un indirizzo IP.

Se si desidera registrare la porta 443 per tutti gli indirizzi IP del server e ascoltare solo sull'endpoint "/ws/" senza eseguire il server con privilegi di amministratore, eseguire i seguenti passaggi:

- Registra l'URL utilizzando netsh
 - netsh http add urlacl url=https://+:443/ws/ user=DOMAIN\user
- Configurare il server con il seguente binding
 - NewBinding('+', 443, '/ws/', True);
- Disabilitare la proprietà ConfigureSSLCertificate
 - TsgcWebSocketServer_HTTPAPI.BindingOptions.ConfigureSSLCertificate = false;
- Configurare il certificato SSL
 - [HTTPAPI Server SSL](#)

HTTPAPI | FineTune

TsgcWSServer_HTTPAPI espone una proprietà pubblicata denominata `FineTune` di tipo `TsgcServerHTTPAPI_FineTune`. Raggruppa i controlli di basso livello in modalità kernel che influenzano il modo in cui la Windows HTTP Server API (`http.sys`) mette in coda, distribuisce e completa le richieste.

QueueLength

Racchiude l'impostazione del kernel `HttpServerQueueLengthProperty`. Controlla il numero massimo di richieste in attesa che `http.sys` manterrà nella sua coda in modalità kernel prima che il server le abbia rimosse dalla coda. Quando la coda è piena, il kernel risponde alle nuove connessioni con **503 Service Unavailable** direttamente, senza mai raggiungere la modalità utente.

Tipo: ULONG.

Predefinito: 1000 (il valore predefinito del kernel Windows).

Intervallo: fino a 65535 richieste.

Cosa migliora: aumentare questo valore impedisce al kernel di rifiutare il traffico legittimo su carichi di lavoro a burst - flotte IoT che si riconnettono dopo un'interruzione di rete, client di broadcast di dati di mercato che si riconnettono dopo un reset del provider, o worker di job pianificati che si connettono tutti su un tick cron condiviso.

SkipIOCPOnSuccess

Quando impostato su `True`, il server chiama `SetFileCompletionNotificationModes` sull'handle della coda delle richieste con i flag `FILE_SKIP_COMPLETION_PORT_ON_SUCCESS` e `FILE_SKIP_SET_EVENT_ON_HANDLE`. Le operazioni di I/O in modalità overlapped sulla coda delle richieste che si completano in modo sincrono non inviano più un pacchetto di completamento extra alla porta di completamento I/O.

Tipo: Boolean.

Predefinito: False (nessun flag è impostato, preservando il comportamento corrente).

Cosa migliora: elimina un passaggio da kernel a user-mode sul percorso critico delle richieste quando la chiamata restituisce `NO_ERROR` in modo sincrono — il worker gestisce il completamento inline sul thread chiamante anziché attendere un pacchetto IOCP. È il pattern utilizzato dal campione di riferimento HTTP Server High Performance di Microsoft. Il flag è pensato per essere abbinato a `OperatingMode = ompHighPerf`.

L'API viene risolta in fase di runtime tramite `GetProcAddress` da `kernel32`. Su Windows XP e versioni precedenti (target runtime Delphi 7) il punto di ingresso non esiste - il wrapper restituisce False e il server viene eseguito senza il flag impostato.

OperatingMode

Seleziona una delle due architetture di accept/dispatch.

Type: `TsgcHTTPAPIOperatingMode = (ompClassic, ompHighPerf)`.

Predefinito: `ompClassic`.

- **ompClassic** - un singolo thread acceptor chiama `HttpReceiveHttpRequest` in modo sincrono e smista ogni richiesta al pool di worker IOCP tramite `PostQueuedCompletionStatus`. Questo è il comportamento storico.
- **ompHighPerf** - implementa il pattern di riferimento di Microsoft HTTP Server High Performance. Il thread accettatore viene rimosso. All'avvio, il server pre-pubblica `ThreadPoolSize x HighPerfAcceptsPerWorker` ricezioni asincrone sull'IOCP associato alla coda (per impostazione predefinita $32 \times 4 = 128$ ricezioni simultanee in sospenso). Ogni worker gestisce il proprio completamento inline e pubblica nuovamente un'altra ricezione asincrona, mantenendo una finestra scorrevole fino all'arresto.

Cosa migliora: `ompHighPerf` conviene quando il server gestisce pipeline a stream singolo profonde (upload/download di frame di grandi dimensioni) oppure molti client concorrenti (da centinaia a migliaia). La finestra di ricezione pre-registrata assorbe i picchi senza allocazioni per connessione, e il dispatch inline elimina il collo di bottiglia del

passaggio all'acceptor. Lasciare il valore predefinito `ompClassic` per API a basso traffico e ambienti di sviluppo — su carichi leggeri il costo di gestire 128 contesti pre-registrati supera il risparmio. La modalità può essere cambiata solo prima dell'attivazione del server.

HighPerfAcceptsPerWorker

Controlla quante ricezioni asincrone ogni worker IOCP pre-pubblica quando `operatingMode = ompHighPerf`. Il valore viene ignorato in modalità `ompClassic`. Il numero totale di ricezioni in sospeso concorrenti che il server mantiene è pari a `ThreadPoolSize x HighPerfAcceptsPerWorker`.

Type: Integer.

Predefinito: 4.

Cosa migliora: una finestra per-worker più ampia consente al server di assorbire picchi maggiori di richieste in entrata senza allocare nuovi contesti sul percorso critico. Aumentarla per distribuzioni ad alta concorrenza (flotte IoT, distribuzione di dati di mercato, broker fan-out); il compromesso è la memoria - ogni ricezione pre-inviata mantiene un buffer di richiesta (~16 KB) riservato fino al completamento. Il valore predefinito 4 è un punto di equilibrio conservativo validato rispetto al campione HP di MSDN.

Esempio

Il seguente snippet configura un server HTTP.sys per un backend IoT ad alta concorrenza: una grande coda kernel per assorbire le tempeste di riconnessione, dispatch HighPerf con una finestra di ricezione pre-inviata allargata e dispatch con completamento inline abilitato.

Per una tipica API interna o a basso traffico, lasci ogni proprietà `FineTune` al suo valore predefinito:

TsgcWebSocketServer_HTTPAPI | HTTPAPI Server SSL

Il server può essere configurato per utilizzare **certificati SSL**. Per ottenere un server di produzione con un certificato del server, deve **acquistare** un certificato da un **provider rinomato**: Namecheap, GoDaddy, Thawte, ecc. Per **scopi di test** può utilizzare un **certificato autofirmato** (controlli l'esempio Demos/Chat che utilizza un certificato autofirmato). Legga il seguente articolo [Come creare un certificato autofirmato](#).

Una volta ottenuto il certificato, è necessario configurare il server per specificare quale certificato utilizzare per la cifratura delle connessioni.

Hash del certificato

Per prima cosa è necessario conoscere l'hash del proprio certificato. Trovare l'hash di un certificato è semplice in **powershell** eseguendo un comando **dir** sul contenitore dei certificati.

```
dir cert:\localmachine\my
```

L'hash è il valore **Thumbprint** esadecimale.

```
Directory: Microsoft.PowerShell.Security\Certificate::localmachine\my
Thumbprint                               Subject
-----
C12A8FC8AE668F866B48F23E753C93D357E9BE10  CN=*.mydomain.com
```

Una volta ottenuto il valore Thumbprint, è sufficiente impostarlo nella proprietà **TsgcWebSocketServer_HTTPAPI.TLSOptions.Hash**.

Una volta impostato l'hash, è sufficiente impostare **TsgcWebSocketServer_HTTPAPI.SSL = true** e il server è pronto per avviarsi.

Se si desidera registrare il certificato manualmente utilizzando netsh, usare il seguente comando:

```
netsh http add sslcert ipport=<IP>:<PORT> certhash=<THUMBPRINT>appid="{<GUID>}"
```

<IP>: Specifica l'indirizzo IP locale per il binding. Non utilizzi un binding con wildcard. Utilizzi un indirizzo IP valido.

<PORT>: Specifica la porta per il binding.

<THUMBPRINT>: Il thumbprint del certificato X.509.

<GUID>: Un GUID generato dallo sviluppatore per rappresentare l'app a scopo informativo.

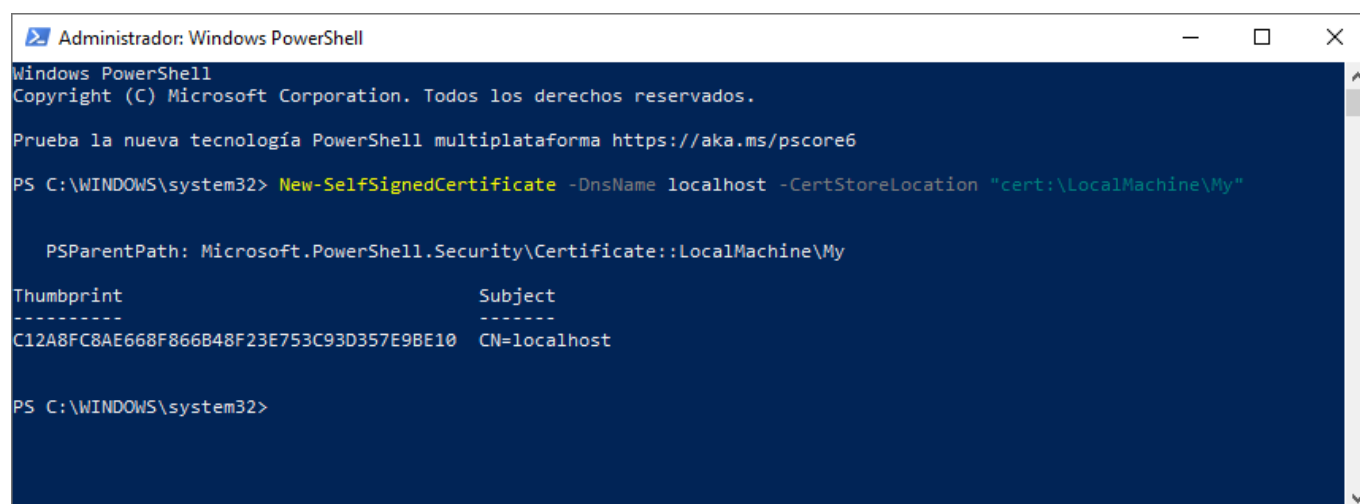
TsgcWebSocketServer_HTTPAPI | Certificati autofirmati

Se è necessario un certificato per i propri test, è possibile creare un certificato autofirmato sulla propria macchina di test. Seguire questi passaggi:

1. Eseguire **Powershell** come **Amministratore**.
2. Esegua il seguente comando per creare il certificato:

```
New-SelfSignedCertificate -DnsName localhost -CertStoreLocation "cert:\LocalMachine\My"
```

In caso di successo, si riceverà una conferma riguardo al nuovo certificato creato. Copiare il Thumbprint e incollarlo nella proprietà **TsgcWebSocketServer_HTTPAPI.TLSOptions.Hash**.



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\WINDOWS\system32> New-SelfSignedCertificate -DnsName localhost -CertStoreLocation "cert:\LocalMachine\My"

PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\My

Thumbprint                               Subject
-----
C12A8FC8AE668F866B48F23E753C93D357E9BE10  CN=localhost

PS C:\WINDOWS\system32>
```

3. **Opzionale**, può aggiungere il Suo certificato autofirmato come **autorità di certificazione attendibile**

```
Run MMC -32 as administrator
```

- 3.1. Selezionare **File / Aggiungi o rimuovi snap-in**
- 3.2. Selezionare Certificato e quindi fare clic su Aggiungi
- 3.3. Selezionare l'account del computer e premere Avanti.
- 3.4. Selezioni Local computer e preme Ok. Ora vedrà i suoi Certificati.
- 4.5. Selezionare il certificato da **Personale / Certificati** e incollarlo in **Autorità di certificazione radice attendibili / Certificati**.

TsgcWebSocketServer_HTTPAPI | Disabilita HTTP/2

Il protocollo HTTP/2 è abilitato per impostazione predefinita in **Server 2016+** e nei sistemi operativi **Windows 10+**. In alcuni browser meno recenti o client HTTP, potrebbe verificarsi un errore perché il protocollo non è completamente supportato. È possibile prevenire questi errori disabilitando il protocollo HTTP/2.

Come disabilitare HTTP/2

- Apri l'Editor del Registro di Windows
- Accedere alla seguente chiave di registro:

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\HTTP\Parameters

- Aggiungere i seguenti valori DWORD e impostare entrambi i valori a zero.
 - EnableHttp2Tls
 - EnableHttp2Cleartext
- Riavviare il computer.

TsgcWebSocketServer_HTTPAPI | Intestazioni personalizzate

È possibile personalizzare la risposta del server HTTP.SYS utilizzando la proprietà **CustomHeaders** dell'oggetto risposta.

Imposti il valore di CustomHeaders con il nome dell'intestazione e il valore dell'intestazione separati da caratteri di nuova riga.

Esempio: se si desidera aggiungere i seguenti header, di seguito è riportato un codice di esempio

Access-Control-Allow-Origin: *

Access-Control-Allow-Methods: GET, POST, OPTIONS, PUT, PATCH, DELETE

```
private void OnHttpRequest(TsgcWSConnection_HTTPAPI aConnection, const THttpRequest aRequestInfo,
    ref THttpResponse aResponseInfo)
{
    aResponseInfo.ResponseNo = 200;
    aResponseInfo.CustomHeaders = "Access-Control-Allow-Origin: *" + Environment.NewLine + "Access-Control-Allow-Methods: GET, POST, OPTIONS, PUT, PATCH, DELETE";
}
```

TsgcWebSocketServer_HTTPAPI | Invio di una risposta di testo

Utilizzi l'evento **OnHTTPRequest** per gestire le richieste HTTP.

La classe **THttpRequest** contiene i dati della richiesta HTTP.

- **Documento:** il documento a cui il peer sta cercando di accedere.
- **Metodo:** il metodo HTTP ('GET', 'POST', ecc.)
- **Headers:** le intestazioni della richiesta HTTP.
- **AcceptEncoding:** variabile di codifica accettata, esempio: "gzip, deflate, br".
- **ContentType:** esempio: "text/html"
- **Content:** contenuto della richiesta, se presente.
- **QueryParams:** i parametri della query.
- **Cookies:** i cookie, se presenti.
- **ContentLength:** dimensione del contenuto.
- **AuthExists, AuthUsername, AuthPassword:** dati della richiesta di autenticazione.
- **Stream:** se la richiesta HTTP ha un corpo, questo è lo stream del corpo.

La classe **THttpResponse** contiene i dati della risposta HTTP.

- **ContentText:** è la risposta come testo.
- **ContentType:** esempio: "text/html". Se si desidera codificare ContentText con UTF8, impostare `charset='utf-8'`. Esempio: `text/html; charset=utf-8`
- **CustomHeaders:** se ha bisogno di inviare intestazioni personalizzate, utilizzi questa variabile
- **AuthRealm:** se il server richiede l'autenticazione, imposti questa variabile.
- **ResponseNo:** il numero della risposta HTTP. Esempio: 200 significa che la risposta ha avuto successo.
- **ContentStream:** se la risposta contiene uno stream, impostarlo qui (non liberare lo stream, verrà liberato automaticamente).
- **FileName:** se la risposta è un nome di file, impostare qui il percorso completo del file.
- **Date, Expires, LastModified:** variabili datetime della risposta.
- **CacheControl:** consente di personalizzare il comportamento della cache.

Esempio: se il server riceve una richiesta GET per il documento `/test.html`, invia una risposta OK; altrimenti invia un 404 se si tratta di una richiesta GET per un altro documento, o un errore 500 se si tratta di un metodo diverso.

```
void OnHTTPRequest(TsgcWSConnection_HTTPAPI aConnection,
    THttpRequest aRequestInfo,
    ref THttpResponse aResponseInfo)
{
    if (aRequestInfo.Method == "GET")
    {
        if (aRequestInfo.Document == "/test.html")
        {
            aResponseInfo.ResponseNo = 200;
            aResponseInfo.ContentText = "OK";
            aResponseInfo.ContentType = "text/html; charset=utf-8";
        }
        else
        {
            aResponseInfo.ResponseNo = 404;
        }
    }
    else
    {
        aResponseInfo.ResponseNo = 500;
    }
}
```

TsgcWebSocketServer_HTTPAPI | Invio risposta file

Utilizzare la proprietà `FileName` dell'oggetto **THttpRequestResponse** se si desidera inviare un file come risposta a una richiesta HTTP.

```
void OnHttpRequest(TsgcWSConnection_HTTPAPI aConnection,
    THttpRequest aRequestInfo,
    ref THttpRequestResponse aResponseInfo)
{
    if (aRequestInfo.Method == "GET")
    {
        if (aRequestInfo.Document == "/test.zip")
        {
            aResponseInfo.ResponseNo = 200;
            aResponseInfo.FileName = "c:\\download\\test.zip";
            aResponseInfo.ContentType = "application/zip";
        }
        else
        {
            aResponseInfo.ResponseNo = 404;
        }
    }
    else
    {
        aResponseInfo.ResponseNo = 500;
    }
}
```

Download ripristinabili

Una risposta HTTP 206 Partial Content viene utilizzata quando un server soddisfa una richiesta per una porzione specifica (intervallo) di una risorsa, invece di inviare l'intero file. Questo è comunemente usato per download ripristinabili, streaming multimediale e trasferimenti di file di grandi dimensioni.

Come funziona:

Il client richiede una risorsa parziale: Il client (browser, downloader o media player) invia un'intestazione `Range` che specifica l'intervallo di byte desiderato. Esempio di richiesta:

```
GET /video.mp4 HTTP/1.1
Host: example.com
Range: bytes=1000-5000
```

Questo richiede i byte da 1000 a 5000 di video.mp4.

Il server risponde con HTTP 206: Se il server supporta le richieste di intervallo, risponde con 206 Partial Content e include un'intestazione `Content-Range`. Esempio di risposta:

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 1000-5000/1000000
Content-Length: 4001
Content-Type: video/mp4
```

L'intestazione `Content-Range` mostra:

L'intervallo servito (1000-5000)

La dimensione totale del file (1000000 byte).

L'intestazione `Content-Length` è la dimensione della porzione restituita (4001 byte).

Il Client può richiedere altri chunk:

Il client può inviare più richieste per parti diverse.

Questo consente download riprendibili e streaming efficiente.

```

[HttpGet("download")]
public async Task<IActionResult> DownloadFile()
{
    string filePath = "test.pdf";
    if (!System.IO.File.Exists(filePath))
    {
        return NotFound();
    }
    FileInfo fileInfo = new FileInfo(filePath);
    long fileSize = fileInfo.Length;
    string contentType = "application/pdf";
    HttpContext.Request.Headers.TryGetValue("Range", out var rangeHeader);

    if (!string.IsNullOrEmpty(rangeHeader))
    {
        long start, end;
        string[] range = rangeHeader.ToString().Replace("bytes=", "").Split('-');
        start = string.IsNullOrEmpty(range[0]) ? 0 : long.Parse(range[0]);
        end = string.IsNullOrEmpty(range[1]) ? fileSize - 1 : long.Parse(range[1]);
        if (end >= fileSize)
            end = fileSize - 1;
        long contentLength = end - start + 1;

        Response.StatusCode = (int)HttpStatusCode.PartialContent;
        Response.Headers["Content-Range"] = $"bytes {start}-{end}/{fileSize}";
        Response.Headers["Accept-Ranges"] = "bytes";
        byte[] buffer = new byte[contentLength];
        using (FileStream fs = new FileStream(filePath, FileMode.Open, FileAccess.Read, FileShare.Read))
        {
            fs.Seek(start, SeekOrigin.Begin);
            await fs.ReadAsync(buffer, 0, buffer.Length);
        }
        return File(buffer, contentType);
    }
    return File(System.IO.File.OpenRead(filePath), contentType);
}

```


TsgcWebSocketServer_HTTPAPI | OnDisconnect non attivato

Quando si lavora per la prima volta con il Server HTTPAPI, è molto comune notare che l'evento OnDisconnect non viene attivato immediatamente quando un client chiude la connessione. Il motivo è che il Server HTTPAPI funziona in modo leggermente diverso rispetto ad altri server come Indy. Nel **server Indy** è presente **un thread per ogni connessione** e questo thread controlla ogni x millisecondi se la **connessione è attiva**. Il **Server HTTPAPI** utilizza un **thread pool** che gestisce tutte le connessioni e **non controlla** per ogni connessione se è attiva o meno.

Per ricevere una notifica quando il client chiude la connessione, eseguire la seguente configurazione:

1. Se utilizza un [TsgcWebSocketClient](#), imposti **Options.CleanDisconnect := True**. Ciò significa che prima della chiusura della connessione, il client tenterà di inviare una notifica al server indicando che la connessione verrà chiusa. Se il server riceve questo messaggio, verrà chiamato l'evento OnDisconnect.
2. Per le altre disconnessioni, l'unica soluzione è scrivere qualcosa sul socket; se l'operazione fallisce, la connessione è disconnessa. **Abiliti HeartBeat** sul server HTTPAPI, e imposti ad esempio un intervallo di 60 secondi e un timeout di 0. Questa configurazione significa che ogni 60 secondi tutte le connessioni verranno pingate, e se qualcuna è disconnessa, verrà attivato l'evento **OnDisconnect**. Può impostare un valore di HeartBeat.Interval più basso, ma non lo imposti troppo basso (1 secondo ad esempio è troppo basso) perché le prestazioni del server ne sentirebbero.

TsgcWebSocketClient_WinHTTP

TsgcWebSocketClient_WinHTTP implementa un componente client VCL WebSocket in grado di connettersi a un server WebSocket.

Introduzione

TsgcWebSocketClient_WinHTTP implementa un componente VCL WebSocket Client che può connettersi a un server WebSocket. Si basa sull'API WinHTTP e richiede Windows 8 o versioni successive. Seguire i passaggi seguenti per configurare questo componente:

1. Trascinare un componente TsgcWebSocketClient_WinHTTP nel form
2. Imposti Host e Port (l'impostazione predefinita è 80) per connettersi a un server WebSocket disponibile. Può impostare la proprietà URL e Host, Port, Parameters... verranno aggiornati dall'URL. Esempio: wss://127.0.0.1:8080/ws/ produrrà:

```
oClient = new TsgcWebSocketClient_WinHTTP();
oClient.Host = "127.0.0.1";
oClient.Port = 80;
oClient.TLS = true;
oClient.Options.Parameters = "/ws/";
```

3. È possibile selezionare se si desidera utilizzare TLS (connessione sicura) o meno; disabilitato per impostazione predefinita.
4. È possibile utilizzare i seguenti eventi per personalizzare il flusso del client WebSocket:
5. Creare una procedura e impostare la proprietà Active su True.

Riferimento

- Proprietà — 16 proprietà disponibili sul componente.
- Metodi — 8 metodi pubblici.
- Eventi — 9 eventi gestibili.

TsgcWebSocketFirewall

TsgcWebSocketFirewall implementa un componente firewall completo che protegge i server WebSocket da un'ampia gamma di minacce alla sicurezza.

Introduzione

TsgcWebSocketFirewall implementa un componente firewall completo che protegge i server WebSocket da un'ampia gamma di minacce alla sicurezza. Fornisce sedici moduli di protezione tra cui filtro blacklist/whitelist degli IP, protezione contro la forza bruta con ban automatico degli IP, rilevamento di SQL injection, rilevamento XSS (Cross-Site Scripting), limitazione della velocità delle connessioni, protezione contro il flood di messaggi, filtro GeoIP basato sul paese, rilevamento dei bot basato su IP, punteggio dinamico delle minacce, rilevamento di path traversal, rilevamento di command injection, limitazione della dimensione del payload, protezioni specifiche per WebSocket (convalida dell'origine, limiti di dimensione dei frame, filtro dei subprotocolli), escalation progressiva del ban, un motore di regole personalizzate e statistiche in tempo reale.

Il firewall si integra automaticamente con i componenti server tramite la proprietà **Firewall** disponibile su TsgcWebSocketServer, TsgcWebSocketHTTPServer e TsgcWebSocketServer_HTTPAPI. Una volta assegnato e configurato, intercetta connessioni e messaggi senza richiedere alcun collegamento manuale di eventi.

Avvio rapido

1. Rilasciare un componente **TsgcWebSocketFirewall** sul form.
2. Configurare i moduli di protezione desiderati:

Blacklist: Impostare Enabled a True e aggiungere IP o intervalli CIDR alla lista IPs (es. "10.0.0.0/8", "192.168.1.100").

Whitelist: Impostare Enabled su True e aggiunga gli IP attendibili. Gli IP in whitelist ignorano tutti gli altri controlli.

BruteForce: Impostare Enabled su True. Configurare MaxAttempts (predefinito 5), TimeWindowSec (predefinito 60) e BanDurationSec (predefinito 300).

SQLInjection: impostare Enabled su True. Rileva i pattern comuni di SQL injection nei messaggi.

XSS: Impostare Enabled su True. Rileva i pattern di cross-site scripting nei messaggi.

RateLimit: Impostare Enabled su True. Limita le connessioni simultanee per IP (predefinito 10 ogni 60 secondi).

FloodProtection: Impostare Enabled su True. Limita i messaggi al secondo per IP (predefinito 100).

PayloadLimit: Impostare Enabled su True. Blocca i messaggi che superano MaxSizeBytes (predefinito 1 MB).

PathTraversal: Impostare Enabled su True. Rileva i pattern di directory traversal.

CommandInjection: Impostare Enabled su True. Rileva i pattern di iniezione di comandi shell.

ThreatScore: Impostare Enabled su True. Assegna punteggi dinamici agli IP in base alle violazioni.

BanEscalation: Impostare Enabled su True. Aumenta progressivamente la durata dei ban per i trasgressori recidivi.

GeoIP: Impostare Enabled su True. Blocca o consente le connessioni in base al codice paese.

BotDetection: Imposti Enabled su True. Classifica gli IP in connessione come crawler verificati, host di data-center, bot sospetti, in blacklist o umani.

WebSocketProtection: Impostare Enabled a True. Convalida origini, dimensioni dei frame e sottoprotocolli.

CustomRules: Impostare Enabled su True. Motore di regole configurabile per filtraggio complesso.

3. Assegna il firewall a un componente server:

```
sgcWebSocketHTTPServer1.Firewall := sgcWebSocketFirewall1;
```

4. Avviare il server. Il firewall protegge automaticamente tutte le connessioni e i messaggi.

Proprietà

Proprietà	Descrizione
Abilitato	Abilita o disabilita il firewall. Predefinito: True.
Blacklist	Configurazione della blacklist IP. Contiene le proprietà Enabled (Boolean) e IPs (TStringList). Supporta IP esatti e notazione CIDR (ad es. "192.168.0.0/16").
Whitelist	Configurazione della lista bianca degli IP. Gli IP nella lista bianca bypassano tutti i controlli di sicurezza. Stessa struttura della Blacklist.
BruteForce	Protezione contro la forza bruta. Proprietà: Enabled, MaxAttempts (predefinito 5), TimeWindowSec (predefinito 60), BandDurationSec (predefinito 300). Vieta automaticamente gli indirizzi IP che superano il limite di tentativi.
SQLInjection	Rilevamento di SQL injection. Proprietà: Enabled, Action (faDeny/faAllow/faLog), CustomPatterns (TStringList per pattern aggiuntivi).
XSS	Rilevamento cross-site scripting. Proprietà: Enabled, Action (faDeny/faAllow/faLog).
RateLimit	Rate limiting delle connessioni. Proprietà: Enabled, MaxConnectionsPerIP (predefinito 10), TimeWindowSec (predefinito 60).
Flood-Protection	Protezione contro il flood di messaggi. Proprietà: Enabled, MaxMessagesPerSec (predefinito 100), Action (faDeny/faAllow/faLog).
Payload-Limit	Limitazione della dimensione dei messaggi. Proprietà: Enabled, MaxSizeBytes (predefinito 1048576), Action (faDeny/faAllow/faLog).
PathTraversal	Rilevamento del path traversal. Proprietà: Enabled, Action (faDeny/faAllow/faLog).
CommandInjection	Rilevamento di command injection. Proprietà: Enabled, Action (faDeny/faAllow/faLog), CustomPatterns (TStringList per pattern aggiuntivi).

ThreatScore	Punteggio dinamico delle minacce. Proprietà: Enabled, AutoBanThreshold (predefinito 80), DecayPerHour (predefinito 5), più pesi per tipo di violazione per ottimizzare il calcolo del punteggio.
BanEscalation	Escalation progressiva del ban. Proprietà: Enabled, Levels (TStringList di durate in secondi), ResetAfterSec (predefinito 86400). Ogni ban successivo utilizza la durata del livello successivo. Un valore di livello pari a 0 equivale a un ban permanente.
GeoIP	Filtraggio geografico degli IP. Proprietà: Enabled, Mode (gmBlockList/gmAllowList), Countries (TStringList di codici paese ISO 3166-1 alpha-2), DatabaseFile (percorso di un database CSV GeoIP).
BotDetection	Rilevamento dei bot basato su IP. Proprietà: Enabled, KnownBotsFile, CacheDurationSec (predefinito 3600), DNSTimeoutMS (predefinito 2000), più i sotto-oggetti ReverseDNS, Datacenter e DNSBL. Solo classificazione: classifica gli IP come crawler verificati, host di datacenter, bot sospetti, in blocklist o umani senza bloccare.
WebSocketProtection	Protezioni specifiche per WebSocket. Proprietà: Enabled, AllowedOrigins (TStringList), MaxFrameSize (Integer), AllowedSubprotocols (TStringList).
CustomRules	Motore di regole personalizzato. Proprietà: Enabled, Rules (TCollection di TsgcFirewallRuleItem). Ogni regola definisce condizioni e azioni per logiche di filtraggio complesse.

Eventi

OnFiltered: Si attiva quando una connessione o un messaggio viene bloccato. Il parametro *Allow* (var Boolean) Le consente di sovrascrivere la decisione del firewall.

OnViolation: Attivato quando viene rilevata una violazione di sicurezza. Fornisce l'indirizzo IP, il tipo di violazione e i dettagli a scopo di logging.

OnResolveCountry: Generato quando GeoIP deve risolvere un IP in un codice paese. Assegnare un gestore per fornire una risoluzione paese personalizzata oppure caricare un file di database GeoIP per lookup automatici.

OnBotDetected: Si attiva quando il rilevamento dei bot classifica un IP. Fornisce l'indirizzo IP, il verdetto TsgcBotClassification e il nome del bot corrispondente per la registrazione o l'analisi.

OnResolveBot: Si attiva quando il rilevamento dei bot deve classificare un IP. Assegna un gestore per fornire un verdetto personalizzato al posto del resolver integrato.

OnThreatScoreChanged: Generato quando il punteggio di minaccia di un IP cambia. Fornisce l'indirizzo IP insieme ai valori del punteggio vecchio e nuovo.

Metodi pubblici

Metodo	Descrizione
IsConnectionAllowed(IP)	Verifica se una connessione dall'IP specificato è consentita. Viene chiamato automaticamente quando integrato con un server.

IsMessageAllowed(IP, Message)	Verifica se un messaggio proveniente dall'IP specificato supera tutti i filtri messaggi. Chiamato automaticamente quando integrato con un server.
RegisterConnection(IP)	Registra una nuova connessione per il monitoraggio della limitazione della velocità. Viene chiamato automaticamente.
UnregisterConnection(IP)	Rimuove una connessione dal tracciamento. Chiamato automaticamente alla disconnessione.
RegisterFailedAttempt(IP)	Registra un tentativo di autenticazione fallito per il rilevamento di attacchi a forza bruta. Chiamare questo metodo dall'handler dell'evento OnAuthentication quando l'autenticazione fallisce.
BanIP(IP, DurationSec)	Banna manualmente un indirizzo IP. DurationSec = 0 significa ban permanente.
UnbanIP(IP)	Rimuove un ban dall'indirizzo IP specificato.
IsBanned(IP)	Restituisce True se l'IP specificato è attualmente bannato.
ClearBans	Rimuove tutti i ban attivi.
ClearTracking	Reimposta tutti i dati di tracciamento interni (conteggi connessioni, log dei tentativi, conteggi messaggi).
SaveBansToFile(FileName)	Salva tutti i ban attivi in un file per la persistenza tra i riavvii.
LoadBansFromFile(FileName)	Ripristina i ban da un file precedentemente salvato.
LoadGeoIPDatabase(FileName)	Carica un database GeoIP in formato CSV. Formato atteso: start_ip,end_ip,country_code.
LookupCountry(IP)	Restituisce il codice paese ISO a 2 lettere per un indirizzo IP.
GetBotClassification(IP)	Restituisce il TsgcBotClassification per un indirizzo IP (privilegiando la cache; un cache miss può restituire bcUnknown e attivare una ricerca asincrona).
GetThreatScore(IP)	Restituisce il punteggio di minaccia corrente (0-100) per un indirizzo IP.
ResetThreatScore(IP)	Reimposta il punteggio di minaccia di un IP a 0.
GetTopThreats(Count)	Restituisce un elenco di IP con i punteggi di minaccia più elevati.
IsOriginAllowed(Origin)	Verifica se un header WebSocket Origin è consentito dalla configurazione WebSocketProtection.
IsFrameSizeAllowed(Size)	Verifica che la dimensione di un messaggio rientri nei limiti definiti da WebSocketProtection.MaxFrameSize.
IsSubprotocolAllowed(Subprotocol)	Verifica se un sottoprotocollo WebSocket è consentito dalla configurazione WebSocketProtection.

Statistiche

Oggetto TsgcFirewallStats pubblico di sola lettura con contatori in tempo reale per connessioni bloccate, messaggi filtrati, ban emessi, punteggi di minaccia e conteggi di hit per modulo.

Integrazione Automatica

Quando assegnato alla proprietà Firewall di un server, il componente automaticamente:

- **Alla nuova connessione:** controlla la blacklist, la whitelist, i ban, i limiti di frequenza, il paese GeoIP e valuta le regole personalizzate. Rifiuta la connessione prima dell'evento OnTCPConnect se bloccata.
- **Al ricevimento di un messaggio:** Verifica SQL injection, pattern XSS, dimensione del payload, path traversal, command injection, limiti di flood e valuta le regole personalizzate. Aggiorna i punteggi di minaccia. Disconnette il client se viene rilevata una violazione.
- **Alla disconnessione:** Annulla la registrazione della connessione dal tracciamento interno.

Supporto CIDR

La blacklist e la whitelist supportano la notazione CIDR per gli intervalli IP:

- **192.168.1.0/24** — corrisponde da 192.168.1.0 a 192.168.1.255
- **10.0.0.0/8** — corrisponde a tutti gli indirizzi 10.x.x.x
- **172.16.0.0/16** — corrisponde agli indirizzi 172.16.x.x

Esempio

```
// Configure firewall on server
server.Firewall.Enabled = true;
server.Firewall.Blacklist.Enabled = true;
server.Firewall.Blacklist.IPs.Add("10.0.0.0/8");
server.Firewall.Whitelist.Enabled = true;
server.Firewall.Whitelist.IPs.Add("192.168.1.1");
server.Firewall.BruteForce.Enabled = true;
server.Firewall.BruteForce.MaxAttempts = 3;
server.Firewall.BruteForce.BanDurationSec = 600;
server.Firewall.SQLInjection.Enabled = true;
server.Firewall.XSS.Enabled = true;
server.Firewall.RateLimit.Enabled = true;
server.Firewall.RateLimit.MaxConnectionsPerIP = 5;
server.Firewall.FloodProtection.Enabled = true;

// GeoIP: block connections from specific countries
server.Firewall.GeoIP.Enabled = true;
server.Firewall.GeoIP.Mode = GeoIPMode.BlackList;
server.Firewall.GeoIP.Countries.Add("CN");
server.Firewall.GeoIP.Countries.Add("RU");
server.Firewall.LoadGeoIPDatabase("geoip.csv");

// Threat scoring with auto-ban
server.Firewall.ThreatScore.Enabled = true;
server.Firewall.ThreatScore.AutoBanThreshold = 80;

// Progressive ban escalation (5min, 30min, 2hr, 24hr, permanent)
server.Firewall.BanEscalation.Enabled = true;
server.Firewall.BanEscalation.Levels.Add("300");
server.Firewall.BanEscalation.Levels.Add("1800");
server.Firewall.BanEscalation.Levels.Add("7200");
server.Firewall.BanEscalation.Levels.Add("86400");
server.Firewall.BanEscalation.Levels.Add("0");

// Additional content protection
server.Firewall.PayloadLimit.Enabled = true;
server.Firewall.PayloadLimit.MaxSizeBytes = 65536;
```

```

server.Firewall.PathTraversal.Enabled = true;
server.Firewall.CommandInjection.Enabled = true;

// Persistent bans
server.Firewall.SaveBansToFile("bans.dat");
server.Firewall.LoadBansFromFile("bans.dat");

// Optional: Track failed login attempts
server.OnAuthentication += (sender, e) =>
{
    e.Authenticated = (e.User == "admin") && (e.Password == "secret");
    if (!e.Authenticated)
        server.Firewall.RegisterFailedAttempt(e.Connection.IP);
};

// Optional: Handle firewall events for logging
server.Firewall.OnViolation += (sender, e) =>
{
    Log("Firewall violation from " + e.IP + ": " + e.Details);
};

```

Pattern di SQL Injection rilevati

Il rilevatore di SQL injection integrato verifica i seguenti pattern (senza distinzione tra maiuscole e minuscole):

- ' OR ', ' AND ' — iniezione booleana
- UNION SELECT — Iniezione basata su Union
- '; DROP, '; DELETE, '; INSERT, '; UPDATE — Iniezione di istruzioni
- -- (commenti SQL)
- EXEC(, EXECUTE(— Esecuzione di comandi
- xp_cmdshell — Stored procedure estese
- CAST(, CONVERT(— Abuso di conversione di tipo
- 1=1, 1='1 — Iniezione per tautologia

È possibile aggiungere ulteriori pattern personalizzati tramite la proprietà `SQLInjection.CustomPatterns`.

Pattern XSS rilevati

- <script — Script injection
- javascript: — XSS basato su protocollo
- onerror=, onload=, onclick=, onmouseover= — Iniezione di gestori di eventi
- eval(— Valutazione JavaScript
- document.cookie — Furto di cookie
- <iframe, <object, <embed — Iniezione di elementi
- <svg onload — XSS basato su SVG
- expression(— iniezione di espressioni CSS

Pattern di path traversal rilevati

Il rilevatore di path traversal integrato verifica questi pattern:

- ../ e ..\ — Traversata relativa di directory
- %2e%2e%2f, %2e%2e/, ..%2f — varianti codificate URL
- %00 — Iniezione di byte null
- /etc/passwd — accesso a file sensibili Unix
- c:\windows — accesso alla directory di sistema Windows

Pattern di command injection rilevati

Il rilevatore integrato di iniezione di comandi verifica i seguenti pattern:

- `;` `|` `&&` `||` — Operatori shell (concatenazione di comandi)
- ``command`` — Esecuzione con backtick
- `$(command)` — Esecuzione subshell
- `rm -rf` — Operazioni distruttive sui file
- `wget`, `curl` — Download di file remoto
- `/bin/sh`, `/bin/bash` — invocazione della shell Unix
- `cmd.exe` — interprete dei comandi di Windows
- `powershell` — Invocazione PowerShell

È possibile aggiungere pattern personalizzati aggiuntivi tramite la proprietà `CommandInjection.CustomPatterns`.

Sicurezza dei thread

Il componente firewall è completamente thread-safe. Tutti i metodi pubblici utilizzano sezioni critiche interne per proteggere l'accesso concorrente ai dati di tracciamento. È stato sottoposto a stress test con 20 thread concorrenti che eseguono 100.000 operazioni senza errori e senza perdite di memoria.

Il componente può essere condiviso in modo sicuro tra più istanze server e accessibile da qualsiasi thread (gestori di eventi server, thread timer, ecc.) senza sincronizzazione esterna.

Riferimento

- Proprietà — 16 proprietà disponibili sul componente.
- Metodi — 21 metodi pubblici.
- Eventi — 6 eventi gestibili.

Firewall: Blacklist e Whitelist

TsgcWebSocketFirewall fornisce il controllo degli accessi basato su IP tramite due meccanismi complementari: una blacklist che blocca IP specifici e una whitelist che concede accesso incondizionato agli IP attendibili.

Blacklist

La blacklist impedisce le connessioni dagli indirizzi IP o dagli intervalli IP specificati. Quando abilitata, qualsiasi connessione in entrata da un IP in blacklist viene rifiutata prima di raggiungere gli eventi di connessione del server.

Proprietà	Descrizione
Blacklist.Enabled	Abilita o disabilita il controllo della blacklist. Predefinito: False.
Blacklist.IPs	TStringList contenente indirizzi IP bloccati o range CIDR.

Aggiunta di IP in fase di progettazione

Fare clic sulla proprietà IPs nell'Object Inspector per aprire l'editor di String List. Aggiungere un IP o un intervallo CIDR per riga:

```
192.168.1.100
10.0.0.0/8
172.16.0.0/12
```

Aggiunta di IP in fase di esecuzione

```
server.Firewall.Blacklist.Enabled = true;
server.Firewall.Blacklist.IPs.Add("192.168.1.100");
server.Firewall.Blacklist.IPs.Add("10.0.0.0/8");
```

Whitelist

La whitelist concede accesso incondizionato agli indirizzi IP specificati. **Gli IP in whitelist bypassano tutti gli altri controlli del firewall**, inclusi blacklist, ban per brute force, rate limit e filtraggio dei messaggi.

Proprietà	Descrizione
Whitelist.Enabled	Abilita o disabilita il controllo della whitelist. Per impostazione predefinita: False.
Whitelist.IPs	TStringList contenente indirizzi IP o intervalli CIDR attendibili.

Esempio

```
// Allow internal network unconditionally
server.Firewall.Whitelist.Enabled = true;
server.Firewall.Whitelist.IPs.Add("192.168.1.0/24");
server.Firewall.Whitelist.IPs.Add("127.0.0.1");
```

Ordine di priorità

Quando sono abilitate sia la blacklist sia la whitelist, il firewall le valuta in questo ordine:

1. Se l'IP è nella whitelist, la connessione viene **autorizzata immediatamente**. Non vengono eseguiti ulteriori controlli.
2. Se l'IP è nella lista nera, la connessione viene **negata**.
3. Se l'IP non è presente in nessuna delle due liste, la connessione prosegue verso gli altri controlli (brute force, rate limiting, ecc.).

Ciò significa che una voce in whitelist ha sempre la priorità su una voce in blacklist per lo stesso IP.

Notazione CIDR

Sia la blacklist che la whitelist supportano la notazione CIDR (Classless Inter-Domain Routing) per specificare gli intervalli di IP:

CIDR	Range	Indirizzi
192.168.1.0/24	192.168.1.0 - 192.168.1.255	256
192.168.0.0/16	192.168.0.0 - 192.168.255.255	65,536
10.0.0.0/8	10.0.0.0 - 10.255.255.255	16,777,216
172.16.0.0/12	172.16.0.0 - 172.31.255.255	1,048,576

È possibile combinare IP esatti e intervalli CIDR nella stessa lista:

```
server.Firewall.Blacklist.IPs.Add("203.0.113.50"); // single IP
server.Firewall.Blacklist.IPs.Add("198.51.100.0/24"); // entire subnet
```

Combinazione di Blacklist e Whitelist

Un pattern comune consiste nel bloccare un intervallo ampio ma consentire IP specifici all'interno di tale intervallo:

```
// Block the entire 10.x.x.x range
server.Firewall.Blacklist.Enabled = true;
server.Firewall.Blacklist.IPs.Add("10.0.0.0/8");

// But allow the monitoring server
server.Firewall.Whitelist.Enabled = true;
server.Firewall.Whitelist.IPs.Add("10.1.1.50");
```

In questo esempio, tutti gli IP nell'intervallo 10.x.x.x sono bloccati eccetto 10.1.1.50, che è inserito nella whitelist e aggira tutti i controlli.

Firewall: Protezione contro la forza bruta

Il modulo di protezione dalla forza bruta rileva tentativi di autenticazione falliti ripetuti dallo stesso indirizzo IP e blocca automaticamente l'IP in questione per una durata configurabile.

Come Funziona

Il firewall tiene traccia dei tentativi falliti per indirizzo IP all'interno di una finestra temporale scorrevole. Quando il numero di errori supera la soglia configurata, l'IP viene automaticamente bannato. Gli IP bannati vengono rifiutati a livello di connessione, prima che si attivino qualsiasi evento del server.

1. Un client si connette e tenta l'autenticazione.
2. Se l'autenticazione fallisce, il codice chiama **RegisterFailedAttempt(IP)** per notificarlo al firewall.
3. Il firewall incrementa il contatore degli errori per quell'IP nella finestra temporale corrente.
4. Se il contatore raggiunge **MaxAttempts**, l'IP viene bannato per **BanDurationSec** secondi.
5. Qualsiasi nuova connessione da un IP bannato viene rifiutata immediatamente.
6. Alla scadenza del periodo di ban, l'IP viene automaticamente rimosso dal ban e il contatore degli errori viene azzerato.

Proprietà

Proprietà	Predefinito	Descrizione
BruteForce.Enabled	False	Abilita o disabilita la protezione contro gli attacchi brute force.
BruteForce.MaxAttempts	5	Numero massimo di tentativi falliti consentiti nella finestra temporale prima che l'IP venga bannato.
BruteForce.TimeWindowSec	60	Durata della finestra temporale scorrevole in secondi. I tentativi falliti più vecchi di questo valore vengono eliminati.
BruteForce.BanDurationSec	300	Durata del blocco automatico in secondi. Impostare a 0 per un blocco permanente (fino alla rimozione manuale o al riavvio del server).

RegisterFailedAttempt

Il firewall non sa quali connessioni rappresentano tentativi di login falliti. È necessario chiamare **RegisterFailedAttempt** dal proprio gestore di eventi di autenticazione ogni volta che un tentativo di login fallisce:

```
server.OnAuthentication += (sender, e) =>
{
    e.Authenticated = CheckCredentials(e.User, e.Password);
    if (!e.Authenticated)
        server.Firewall.RegisterFailedAttempt(e.Connection.IP);
};
```

Gestione manuale del Ban

Oltre al ban automatico, è possibile gestire manualmente i ban:

Metodo	Descrizione
BanIP(IP, DurationSec)	Blocca manualmente un IP. Usare DurationSec = 0 per un blocco permanente.
UnbanIP(IP)	Rimuove un ban dall'IP specificato.
IsBanned(IP)	Restituisce True se l'IP è attualmente bannato.
ClearBans	Rimuove tutti i ban attivi (sia automatici che manuali).

Esempio: Ban manuale

```
// Ban an IP for 1 hour
server.Firewall.BanIP("203.0.113.50", 3600);

// Permanent ban
server.Firewall.BanIP("198.51.100.25", 0);

// Check and unban
if (server.Firewall.IsBanned("203.0.113.50"))
    server.Firewall.UnbanIP("203.0.113.50");
```

Esempio di configurazione

```
// Strict configuration: 3 attempts in 30 seconds, ban for 10 minutes
server.Firewall.BruteForce.Enabled = true;
server.Firewall.BruteForce.MaxAttempts = 3;
server.Firewall.BruteForce.TimeWindowSec = 30;
server.Firewall.BruteForce.BanDurationSec = 600;
```

```
// Lenient configuration: 10 attempts in 5 minutes, ban for 1 minute
server.Firewall.BruteForce.Enabled = true;
server.Firewall.BruteForce.MaxAttempts = 10;
server.Firewall.BruteForce.TimeWindowSec = 300;
server.Firewall.BruteForce.BanDurationSec = 60;
```

Note

- Gli IP nella whitelist non vengono mai bannati, anche se RegisterFailedAttempt viene chiamato per essi.
- I ban vengono memorizzati in memoria per impostazione predefinita. Utilizzare **SaveBansToFile** e **LoadBansFromFile** per persistere i ban tra i riavvii del server.
- La finestra temporale è scorrevole, non fissa. Ogni tentativo fallito viene tracciato individualmente e scade dopo TimeWindowSec secondi.

Firewall: Rilevamento SQL Injection e XSS

TsgcWebSocketFirewall include il rilevamento integrato basato su pattern per attacchi di SQL injection e Cross-Site Scripting (XSS) nei messaggi WebSocket. Questi moduli analizzano il contenuto dei messaggi in arrivo e intervengono quando viene rilevato un pattern di minaccia.

Rilevamento SQL Injection

Il modulo SQL injection analizza tutti i messaggi WebSocket in arrivo alla ricerca di pattern comuni di SQL injection. Il rilevamento non distingue tra maiuscole e minuscole.

Proprietà

Proprietà	Descrizione
SQLInjection.Enabled	Abilita o disabilita il rilevamento di SQL injection. Predefinito: False.
SQLInjection.Action	Azione da intraprendere quando viene rilevato uno schema: faDeny (blocca il messaggio e disconnette, predefinito), faLog (attiva l'evento OnViolation ma consente il messaggio), faAllow (nessuna azione, rilevamento disabilitato).
SQLInjection.CustomPatterns	TStringList di pattern regex aggiuntivi da verificare. Questi vengono valutati in aggiunta ai pattern integrati.

Pattern integrati

Per impostazione predefinita vengono controllati i seguenti pattern (senza distinzione tra maiuscole e minuscole):

Pattern	Tipo di attacco
' OR ', ' AND '	Iniezione basata su booleano
UNION SELECT	Iniezione basata su union
'; DROP, '; DELETE, '; INSERT, '; UPDATE	Iniezione di statement / manipolazione di dati
-- (doppio trattino)	Iniezione di commenti SQL
EXEC(, EXECUTE(	Esecuzione di comandi
xp_cmdshell	Abuso di stored procedure estese
CAST(, CONVERT(	Abuso di conversione di tipo
1=1, 1='1	Iniezione per tautologia

Pattern personalizzati

È possibile aggiungere pattern specifici dell'applicazione utilizzando la proprietà CustomPatterns:

```
server.Firewall.SQLInjection.Enabled = true;
server.Firewall.SQLInjection.CustomPatterns.Add("WAITFOR DELAY");
server.Firewall.SQLInjection.CustomPatterns.Add("BENCHMARK\\(");
server.Firewall.SQLInjection.CustomPatterns.Add("SLEEP\\(");
```

XSS Detection

Il modulo XSS rileva i pattern di Cross-Site Scripting nei messaggi WebSocket, prevenendo l'iniezione di script malevoli che potrebbero essere resi da altri client connessi.

Proprietà

Proprietà	Descrizione
XSS.Enabled	Abilita o disabilita il rilevamento XSS. Predefinito: False.
XSS.Action	Azione da intraprendere quando viene rilevato un pattern: faDeny (blocca e disconnette, predefinito), faLog (genera OnViolation ma consente), faAllow (nessuna azione).

Pattern integrati

I seguenti pattern XSS vengono rilevati (senza distinzione tra maiuscole e minuscole):

Pattern	Tipo di attacco
<script	Iniezione di tag script
javascript:	XSS basato su protocollo
onerror=, onload=, onclick=, onmouseover=	Iniezione di handler di eventi
eval(	Valutazione di codice JavaScript
document.cookie	Furto di cookie
<iframe, <object, <embed	Iniezione di elementi
<svg onload	XSS basato su SVG
expression(	Iniezione di espressioni CSS

Proprietà Action

Sia i moduli SQLInjection che XSS supportano una proprietà Action di tipo TsgcFirewallAction:

Valore	Comportamento
faDeny	Il messaggio viene bloccato e il client viene disconnesso. L'evento OnViolation viene attivato. Questo è il comportamento predefinito.
faLog	Il messaggio viene lasciato passare, ma l'evento OnViolation viene generato a fini di registrazione.
faAllow	Non viene intrapresa alcuna azione. Disabilita di fatto il rilevamento mantenendo il flag Enabled su True.

OnViolation Event

Quando viene rilevato un pattern di SQL injection o XSS (con Action impostata su faDeny o faLog), l'evento OnViolation si attiva con i dettagli della violazione:

```
server.Firewall.OnViolation += (sender, e) =>
{
    switch (e.ViolationType)
    {
        case TsgcFirewallViolationType.fvSQLInjection:
            LogWarning("SQL injection attempt from " + e.IP + ": " + e.Details);
            break;
        case TsgcFirewallViolationType.fvXSS:
            LogWarning("XSS attempt from " + e.IP + ": " + e.Details);
            break;
    }
};
```

Esempio di configurazione

```
// Enable SQL injection detection with deny action
server.Firewall.SQLInjection.Enabled = true;
server.Firewall.SQLInjection.Action = TsgcFirewallAction.faDeny;

// Add custom patterns for time-based SQL injection
server.Firewall.SQLInjection.CustomPatterns.Add("WAITFOR DELAY");
server.Firewall.SQLInjection.CustomPatterns.Add("SLEEP\\(");

// Enable XSS detection in log-only mode (monitor without blocking)
server.Firewall.XSS.Enabled = true;
server.Firewall.XSS.Action = TsgcFirewallAction.faLog;

// Assign violation handler
server.Firewall.OnViolation += FirewallViolation;
```

Note

- Il rilevamento dei pattern viene applicato a tutti i messaggi di testo in arrivo. I messaggi binari non vengono analizzati.
- Gli IP nella whitelist bypassano completamente il filtraggio dei messaggi.
- I pattern integrati coprono i vettori di attacco più comuni. Per le minacce specifiche dell'applicazione, aggiungere pattern personalizzati tramite SQLInjection.CustomPatterns.
- Utilizzare la modalità faLog durante la distribuzione iniziale per monitorare i rilevamenti senza impattare i client, quindi passare a faDeny una volta convalidati i pattern per la propria applicazione.

Firewall: Limitazione della velocità e protezione contro il flooding

TsgcWebSocketFirewall fornisce due meccanismi complementari per il controllo del traffico: la limitazione del tasso di connessione (controllo di quante connessioni un IP può aprire) e la protezione contro il flood di messaggi (controllo di quanti messaggi un client connesso può inviare al secondo).

Limitazione della frequenza delle connessioni

La limitazione della frequenza limita il numero di connessioni che un singolo indirizzo IP può stabilire in una finestra temporale. Ciò impedisce a un singolo client di consumare tutte le connessioni server disponibili.

Proprietà

Proprietà	Predefinito	Descrizione
RateLimit.Enabled	False	Abilita o disabilita la limitazione della velocità di connessione.
RateLimit.MaxConnectionsPerIP	10	Numero massimo di connessioni simultanee consentite da un singolo indirizzo IP.
RateLimit.TimeWindowSec	60	Finestra temporale in secondi per il monitoraggio delle connessioni. Le connessioni più vecchie di questo valore vengono rimosse dal conteggio.

Come Funziona

1. Quando arriva una nuova connessione, il firewall conta il numero di connessioni attive da quell'IP.
2. Se il conteggio è pari o superiore a MaxConnectionsPerIP, la nuova connessione viene rifiutata.
3. Quando una connessione si disconnette, viene rimossa dal conteggio di tracciamento.
4. Le connessioni più vecchie di TimeWindowSec vengono automaticamente rimosse dal tracciamento.

Esempio

```
// Allow maximum 5 connections per IP
server.Firewall.RateLimit.Enabled = true;
server.Firewall.RateLimit.MaxConnectionsPerIP = 5;
server.Firewall.RateLimit.TimeWindowSec = 60;
```

Protezione contro il flood di messaggi

La protezione contro le inondazioni limita il numero di messaggi che un singolo IP può inviare al secondo. Ciò impedisce ai client abusivi di sovraccaricare il server con raffiche rapide di messaggi.

Proprietà

Proprietà	Predefinito	Descrizione
FloodProtection.Enabled	False	Abilita o disabilita la protezione contro le inondazioni di messaggi.
FloodProtection.MaxMessagesPerSec	100	Numero massimo di messaggi consentiti al secondo da un singolo indirizzo IP.
FloodProtection.Action	faDeny	Azione quando viene superato il limite: faDeny (disconnette il client), faLog (genera OnViolation ma consente), faAllow (nessuna azione).

Come Funziona

1. Il firewall mantiene un contatore di messaggi per IP che si resetta ogni secondo.
2. Ogni messaggio in arrivo incrementa il contatore per l'IP del mittente.
3. Quando il contatore supera MaxMessagesPerSec, viene eseguita l'Azione configurata.
4. Con l'azione faDeny, il client viene disconnesso immediatamente. Con faLog, l'evento OnViolation viene generato ma il messaggio viene elaborato normalmente.

Esempio

```
// Limit to 50 messages per second per IP
server.Firewall.FloodProtection.Enabled = true;
server.Firewall.FloodProtection.MaxMessagesPerSec = 50;
server.Firewall.FloodProtection.Action = TsgcFirewallAction.faDeny;
```

Combinare entrambe le protezioni

La limitazione della frequenza e la protezione contro le inondazioni di richieste operano insieme per fornire un controllo completo del traffico:

```
// Connection limiting: max 5 connections per IP
server.Firewall.RateLimit.Enabled = true;
server.Firewall.RateLimit.MaxConnectionsPerIP = 5;

// Message limiting: max 50 messages/sec per IP
server.Firewall.FloodProtection.Enabled = true;
server.Firewall.FloodProtection.MaxMessagesPerSec = 50;
```

Con questa configurazione, ogni IP può aprire al massimo 5 connessioni simultanee e, su tutte quelle connessioni, può inviare un massimo combinato di 50 messaggi al secondo.

Linee guida per l'ottimizzazione

Scenario	MaxConnectionsPerIP	MaxMessagesPerSec
Applicazione di chat	3-5	10-30
Feed di dati in tempo reale	2-3	100-500
Game server	1-2	50-100

API gateway	10-20	200-1000

Iniziare con limiti conservativi e aumentarli in base al monitoraggio. Utilizzare inizialmente la modalità faLog per osservare i pattern di traffico prima di abilitare faDeny.

Note

- Gli IP nella whitelist bypassano sia il rate limiting che la protezione flood.
- I dati di tracciamento del limite di velocità vengono cancellati automaticamente alla chiusura delle connessioni. Utilizzare ClearTracking per reimpostare manualmente tutti i contatori.
- Dietro un reverse proxy, tutte le connessioni possono sembrare provenire dall'IP del proxy. Configurare il proxy in modo che trasmetta l'IP reale del client (ad es. tramite l'intestazione X-Forwarded-For) e assicurarsi che il server lo risolva correttamente.

TsgcWSRateLimiter

TsgcWSRateLimiter implementa un componente di rate limiting completo che protegge gli endpoint del server da traffico eccessivo, abusi e scraping.

Introduzione

TsgcWSRateLimiter implementa un componente di rate limiting completo che protegge gli endpoint del server da traffico eccessivo, abusi e scraping. Supporta tre algoritmi (Token Bucket, Sliding Window, Fixed Window), regole con scope (per IP, per chiave API, per utente, per endpoint), quote a lungo termine (ora / giorno / mese), protezione dai burst, una forma configurabile della risposta HTTP 429 e statistiche in tempo reale. Lo stato interno può essere reso persistente su file o stream in modo che i contatori sopravvivano ai riavvii.

Il rate limiter si integra automaticamente con i componenti server tramite la proprietà **RateLimiter** disponibile su TsgcWebSocketHTTPServer e TsgcWSServer_HTTPAPI. Una volta assegnato, valuta ogni connessione e ogni messaggio senza richiedere la configurazione manuale degli eventi.

Avvio rapido

1. Inserire un componente **TsgcWSRateLimiter** nel form.
2. Abilitare una o più strategie e scope (vedere la documentazione per sezione di seguito).
3. Assegnare il rate limiter a un componente server:

```
sgcWebSocketHTTPServer1.RateLimiter := sgcWSRateLimiter1;
```

4. Avviare il server. Il limitatore di frequenza valuta automaticamente ogni connessione e messaggio, rifiuta il traffico in eccesso con HTTP 429 e aggiorna i propri contatori statistici.

Proprietà di livello superiore

Funzione: Interruttore principale per il componente e file di archiviazione opzionale utilizzato dai metodi di persistenza dello stato. Quando Enabled è False, ogni metodo pubblico viene cortocircuitato su "consentito" e nessun contatore viene aggiornato.

Quando utilizzarlo: Disabiliti il rate limiter a runtime durante un test di carico controllato o durante una finestra di manutenzione senza disassociarlo dal server. Imposti StorageFile affinché i contatori sopravvivano a un riavvio dell'applicazione — per esempio un Token Bucket continua a ricaricarsi dal punto in cui si era fermato invece di ripartire da zero e accettare un'enorme raffica.

Pro-prietà	Type	Predefi-nito	Descrizione
Abilita-to	Boo-lean	True	Interruttore principale. Quando è False, IsAllowed / Consume / IsConnectionAllowed / IsMessageAllowed restituiscono sempre True e nessun contatore viene aggiornato.

StorageFile	string	"	Percorso file opzionale utilizzato da SaveStateToFile / LoadStateFromFile per la persistenza dei contatori tra i riavvii.

```
rateLimiter.Enabled = true;
rateLimiter.StorageFile = "ratelimit.dat";
rateLimiter.LoadStateFromFile(rateLimiter.StorageFile);
```

Strategia TokenBucket

Funzionamento: Mantiene un bucket di massimo *Capacity* token per chiave. Il bucket si riempie a *RefillRate* token ogni *RefillIntervalMs* millisecondi. Ogni richiesta consuma un token (o il valore *Cost* passato a *Consume*). Quando il bucket è vuoto la richiesta viene rifiutata e viene calcolato un suggerimento *RetryAfterSec* in base alla pianificazione di ricarica.

Quando utilizzarlo: L'API dichiara una velocità media di 10 req/sec, ma si desidera consentire ai client ben comportati di eseguire fino a 50 richieste consecutive senza essere limitati. Un Token Bucket con *Capacity*=50 e *RefillRate*=10 per 1000 ms consente a un client di inviare 50 richieste istantaneamente, quindi lo limita a 10 req/sec fino al riempimento del bucket.

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	False	Abilita l'algoritmo Token Bucket come strategia predefinita globale.
Capacity	Integer	100	Numero massimo di token che il bucket può contenere. Controlla la dimensione massima del burst.
RefillRate	Integer	10	Numero di token aggiunti al bucket ogni <i>RefillIntervalMs</i> .
RefillIntervalMs	Integer	1000	Periodo di ricarica in millisecondi. Insieme a <i>RefillRate</i> determina la frequenza di richieste sostenuta.

```
// Average 10 req/sec, bursts up to 50
rateLimiter.TokenBucket.Enabled = true;
rateLimiter.TokenBucket.Capacity = 50;
rateLimiter.TokenBucket.RefillRate = 10;
rateLimiter.TokenBucket.RefillIntervalMs = 1000;
// First 50 requests succeed immediately; afterwards throttled to 10/sec until refill
```

Strategia SlidingWindow

Funzionamento: Memorizza un timestamp per ogni richiesta e conta i timestamp che rientrano in una finestra mobile di *WindowSec* secondi. Rifiuta quando il conteggio supererebbe *MaxRequests*. La finestra scorre in modo continuo, quindi non esiste un momento di reset rigido in cui un burst potrebbe passare.

Quando utilizzarlo: Si pubblica un SLA rigido che garantisce "non più di 100 richieste in qualsiasi finestra temporale mobile di 60 secondi". La Finestra Fissa consentirebbe a un client di inviare 100 richieste alle 12:00:59 e altre 100 alle 12:01:00 (200 in un secondo). La Finestra Scorrevole lo impedisce e applica lo SLA con precisione.

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	False	Abilita l'algoritmo Sliding Window come strategia predefinita globale.
Window-Sec	Integer	60	Dimensione della finestra scorrevole in secondi.
MaxRequests	Integer	1000	Numero massimo di richieste consentite all'interno di qualsiasi finestra mobile ampia WindowSec.

```
// Strict SLA: max 100 requests in any rolling 60 seconds
rateLimiter.SlidingWindow.Enabled = true;
rateLimiter.SlidingWindow.WindowSec = 60;
rateLimiter.SlidingWindow.MaxRequests = 100;
// Client that sends a burst of 100 requests near t=59s will be throttled at t=60s
```

Strategia FixedWindow

Cosa fa: Conta le richieste all'interno di una finestra di orologio fissa di *WindowSec* secondi e reimposta il contatore a zero al limite della finestra. L'algoritmo più semplice e quello più economico in termini di memoria; tuttavia può consentire fino a $2 \times \text{MaxRequests}$ in un breve periodo intorno all'attraversamento di un limite.

Quando utilizzarlo: Quote in stile billing come "1000 richieste per ora, il contatore si azzerà all'ora". I clienti si aspettano un orario di reset chiaramente visibile e il comportamento di burst al confine è accettabile perché l'unità di fatturazione è l'ora, non il minuto mobile.

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	False	Abilita l'algoritmo Fixed Window come strategia predefinita globale.
Window-Sec	Integer	60	Dimensione della finestra fissa in secondi. Il contatore si azzerà quando la finestra scade.
MaxRequests	Integer	1000	Numero massimo di richieste consentite all'interno di una singola finestra fissa.

```
// Billing-style: 1000 requests per hour, resets on the hour
rateLimiter.FixedWindow.Enabled = true;
rateLimiter.FixedWindow.WindowSec = 3600;
rateLimiter.FixedWindow.MaxRequests = 1000;
```

PerIP

Cosa fa: Definisce un rate limit con ambito l'indirizzo IP di origine. Quando una chiave di richiesta non corrisponde a PerAPIKey / PerUser / PerEndpoint, PerIP diventa la regola attiva e sovrascrive le impostazioni predefinite globali Token/Sliding/Fixed. Ogni IP ottiene il proprio contatore indipendente.

Quando utilizzarlo: Un'API pubblica con molti client. Senza PerIP un singolo client rumoroso può consumare il budget globale e affamare tutti gli altri. Assegnare a ogni IP il proprio bucket (ad esempio 100 richieste al minuto) in modo che un client non possa influenzare gli altri.

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	False	Abilita la limitazione della frequenza per IP.
MaxRequests	Integer	100	Numero massimo di richieste per finestra (o capacità del bucket quando la Strategy è rlsTokenBucket).
WindowSec	Integer	60	Dimensione della finestra in secondi. Per Token Bucket viene utilizzata anche per derivare la velocità di ricarica (MaxRequests / WindowSec token al secondo).
Strategia	TsgcRateLimitStrategy	rlsTokenBucket	Algoritmo utilizzato per questo scope: rlsTokenBucket, rlsSliding o rlsFixed.

```
// Every client IP gets its own 100 req/min bucket
rateLimiter.PerIP.Enabled = true;
rateLimiter.PerIP.Strategy = RateLimitStrategy.TokenBucket;
rateLimiter.PerIP.MaxRequests = 100;
rateLimiter.PerIP.WindowSec = 60;
```

PerAPIKey

Funzionamento: Stessa semantica di PerIP ma con scope limitato a un identificatore di chiave API. Il componente riconosce le chiavi che iniziano con il prefisso apikey: come limitate all'ambito della chiave API. Progettato per integrarsi con TsgcWSAPIKeyManager in modo che ogni cliente disponga di un budget di frequenza indipendente indipendentemente dall'IP sorgente.

Quando utilizzarlo: SaaS con prezzi a livelli. Ogni cliente ha la propria API key e la propria quota indipendentemente da dove si connette. Utilizzare PerAPIKey come livello predefinito e differenziare i livelli del cliente (Free / Pro / Enterprise) tramite una regola PerEndpoint con chiave per prefisso di percorso o sovrascrivendo la decisione all'interno del gestore dell'evento OnThrottled.

Proprietà	Type	Predefinito	Descrizione

Abilitato	Boolean	False	Abilita il rate limiting per chiave API.
MaxRequests	Integer	100	Richieste massime per finestra (o capacità del bucket per Token Bucket).
WindowSec	Integer	60	Dimensione della finestra in secondi.
Strategia	TsgcRate-LimitStrategy	rlsTokenBucket	Algoritmo utilizzato per questo ambito.

```
// Default API-key tier: 1000 req/min
rateLimiter.PerAPIKey.Enabled = true;
rateLimiter.PerAPIKey.Strategy = RateLimitStrategy.Sliding;
rateLimiter.PerAPIKey.MaxRequests = 1000;
rateLimiter.PerAPIKey.WindowSec = 60;
// Tier-differentiation can be done via PerEndpoint rules or OnThrottled
```

PerUser

Cosa fa: Stessa struttura di PerIP / PerAPIKey ma indicizzata da un identificatore utente definito dall'applicazione. Il componente riconosce come indicizzate per utente le chiavi che iniziano con il prefisso `user:.` Utile quando un singolo utente può connettersi da più dispositivi o IP e si desidera un unico budget condiviso per quell'utente.

Quando utilizzarlo: App web / mobile autenticata. All'utente `alice@example.com` sono consentite 1000 richieste al giorno indipendentemente dal fatto che effettui le chiamate dal telefono, dal laptop o dal desktop di lavoro. L'aggregazione per IP contenebbe due volte; l'aggregazione per utente fornisce il budget che si desidera effettivamente fatturare.

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	False	Abilita il rate limiting per utente.
MaxRequests	Integer	100	Richieste massime per finestra (o capacità del bucket per Token Bucket).
WindowSec	Integer	60	Dimensione della finestra in secondi.
Strategia	TsgcRate-LimitStrategy	rlsTokenBucket	Algoritmo utilizzato per questo ambito.

```
// 1000 requests/day per authenticated user across all their devices
rateLimiter.PerUser.Enabled = true;
rateLimiter.PerUser.Strategy = RateLimitStrategy.Fixed;
rateLimiter.PerUser.MaxRequests = 1000;
rateLimiter.PerUser.WindowSec = 86400;
// Pass keys like "user:alice@example.com" to Consume / IsAllowed
```


PerEndpoint

Cosa fa: Collezione di regole basate su pattern. Ogni elemento ha il proprio *Pattern* (sono supportati i caratteri jolly * e ?), *Strategy*, *MaxRequests* e *WindowSec*. Vince il primo elemento abilitato il cui pattern corrisponde alla chiave — nessun ulteriore elemento viene valutato. Ha la precedenza su PerAPIKey, PerUser e PerIP.

Quando utilizzarlo: Si dispone di un endpoint costoso `/api/v1/expensive-report` che deve essere limitato a 10 req/min per client, mentre un endpoint di stato economico `/api/v1/status` consente 1000 req/min. Endpoint diversi hanno budget diversi — configurare un elemento PerEndpoint per ogni schema di percorso.

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	False	Abilita il set di regole PerEndpoint. Se False, nessun elemento viene valutato.
Regole	TCollection	vuoto	Raccolta di TsgcRateLimitRuleItem. L'ordine è importante — vince la prima corrispondenza.

Ogni **TsgcRateLimitRuleItem** espone:

Proprietà	Type	Predefinito	Descrizione
Nome	string	"	Nome descrittivo mostrato nell'ispettore oggetti.
Abilitato	Boolean	True	Disabilita questo elemento senza eliminarlo.
Pattern	string	"	Pattern con caratteri jolly confrontato con la chiave (senza distinzione tra maiuscole e minuscole). Supporta * e ?.
MaxRequests	Integer	100	Numero massimo di richieste per finestra / capacità del bucket.
WindowSec	Integer	60	Dimensione della finestra in secondi.
Strategia	TsgcRateLimitStrategy	rlsTokenBucket	Algoritmo da utilizzare per le richieste corrispondenti.

```
rateLimiter.PerEndpoint.Enabled = true;

var ruleReport = rateLimiter.PerEndpoint.Rules.Add();
ruleReport.Name = "expensive-report";
ruleReport.Pattern = "*/api/v1/expensive-report*";
ruleReport.Strategy = RateLimitStrategy.Sliding;
ruleReport.MaxRequests = 10;
ruleReport.WindowSec = 60;

var ruleStatus = rateLimiter.PerEndpoint.Rules.Add();
```

```
ruleStatus.Name = "status";
ruleStatus.Pattern = "*/api/v1/status*";
ruleStatus.Strategy = RateLimitStrategy.TokenBucket;
ruleStatus.MaxRequests = 1000;
ruleStatus.WindowSec = 60;
```

Quote

Cosa fa: Limiti a lungo termine che persistono oltre la finestra mobile / di ricarica della strategia principale. Ogni voce limita quante richieste una chiave (o l'intero sistema) può effettuare entro un periodo di calendario (*qpHour*, *qpDay* o *qpMonth*). Una volta raggiunto il *Limit*, la richiesta viene rifiutata e viene generato l'evento *OnQuotaExceeded*.

Quando utilizzarlo: Piano gratuito limitato a 10.000 richieste al mese come limite massimo. Il token bucket con scorrimento continuo attenua comunque il traffico a breve termine, ma quando la quota mensile è esaurita ogni ulteriore chiamata fallisce fino al mese di calendario successivo.

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	False	Abilita il set di regole di quota. Quando è False non viene valutato alcun elemento.
Quote	TCollection	vuoto	Raccolta di <i>TsgcRateLimitQuotaltem</i> . Tutti gli elementi abilitati vengono valutati nell'ordine.

Ogni *TsgcRateLimitQuotaltem* espone:

Proprietà	Type	Predefinito	Descrizione
Nome	string	"	Nome descrittivo riportato in <i>OnQuotaExceeded</i> e in <i>Result.Reason</i> ("quota:Name").
Abilitato	Boolean	True	Disabilita questa quota senza eliminarla.
Scope	<i>TsgcRateLimitQuotaScope</i>	<i>qscPerKey</i>	<i>qscPerKey</i> conta in modo indipendente per ogni chiave; <i>qscGlobal</i> conta su tutte le chiavi combinate.
Periodo	<i>TsgcRateLimitQuotaPeriod</i>	<i>qpHour</i>	<i>qpHour</i> , <i>qpDay</i> o <i>qpMonth</i> . Il contatore si azzer automaticamente all'inizio di ogni periodo.
Limite	Int64	10000	Numero massimo di richieste consentite entro il periodo.

```
rateLimiter.Quotas.Enabled = true;

var quota = rateLimiter.Quotas.Quotas.Add();
quota.Name = "free-tier-monthly";
quota.Scope = QuotaScope.PerKey;
```

```
quota.Period = QuotaPeriod.Month;
quota.Limit = 10000;
```

BurstProtection

Cosa fa: Un rilevatore di picchi a breve termine che opera in parallelo con la strategia principale. Se più di *BurstThreshold* richieste arrivano dalla stessa chiave entro *BurstWindowMs* millisecondi, la chiave viene posta in cooldown per *CooldownSec* secondi durante i quali ogni richiesta viene rifiutata — anche se la strategia principale le avrebbe consentite.

Quando utilizzarlo: Rilevare client scraper e scanner che inondano un server con più di 100 richieste in burst da 100 ms ma rimangono sotto un limite al minuto. La protezione burst rileva questo pattern a raffica automatica che un utente umano non produce mai.

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	False	Abilita il rilevatore di burst.
BurstThreshold	Integer	50	Numero di richieste nell'intervallo <i>BurstWindowMs</i> che attiva un periodo di raffreddamento.
BurstWindowMs	Integer	500	Finestra di osservazione del burst in millisecondi.
CooldownSec	Integer	30	Durata del periodo di cooldown dopo il rilevamento di un burst. Durante il cooldown ogni richiesta viene rifiutata.

```
// More than 50 requests in 500ms triggers a 30-second cooldown
rateLimiter.BurstProtection.Enabled = true;
rateLimiter.BurstProtection.BurstThreshold = 50;
rateLimiter.BurstProtection.BurstWindowMs = 500;
rateLimiter.BurstProtection.CooldownSec = 30;
```

Risposta

Cosa fa: Configura la forma della risposta HTTP restituita ai client quando una richiesta viene limitata. Controlla il codice di stato, il messaggio nel corpo e se le intestazioni *Retry-After* e *X-RateLimit-** vengono aggiunte alla risposta.

Quando utilizzarlo: Si desidera che i client ben configurati tornino a rallentare correttamente. Restituire lo stato standard 429 con un'intestazione *Retry-After* che indica quanto tempo devono attendere, e includere *X-RateLimit-Limit* / *X-RateLimit-Remaining* / *X-RateLimit-Reset* in modo che i client possano regolare il proprio ritmo in modo proattivo invece di reagire ai rifiuti.

Proprietà	Type	Predefinito	Descrizione
StatusCo- de	Inte- ger	429	Codice di stato HTTP restituito per le richieste limitate dalla frequenza. Utilizzare 429 (standard) o 503 per il rifiuto in stile manutenzione.
Messag- gio	string	"Too Many Requests"	Corpo della risposta HTTP per le richieste limitate dalla velocità.
RetryAfte- rHeader	Boo- lean	True	Quando True aggiunge un'intestazione <code>Retry-After</code> con il ritardo di retry calcolato in secondi.
Include- RateLimi- tHeaders	Boo- lean	True	Quando True aggiunge le intestazioni <code>X-RateLimit-Limit</code> / <code>X-RateLimit-Remaining</code> / <code>X-RateLimit-Reset</code> a ogni risposta.

```
rateLimiter.Response.StatusCode = 429;
rateLimiter.Response.Message = "Too Many Requests - please slow down";
rateLimiter.Response.RetryAfterHeader = true;
rateLimiter.Response.IncludeRateLimitHeaders = true;
```

Statistiche

Funzionamento: Oggetto contatore in sola lettura che espone statistiche in tempo reale sul limitatore di velocità. Tutti i contatori vengono aggiornati atomicamente all'interno della sezione critica interna del componente.

Quando usarlo: Dashboard live in un'interfaccia amministrativa. Eseguire il polling di Stats da un TTimer e visualizzare TotalRequests / TotalThrottled / ActiveKeys correnti in etichette, indicatori o grafici affinché gli operatori possano vedere la pressione del traffico in tempo reale.

Proprietà	Type	Descrizione
TotalRequests	Int64	Numero totale di richieste valutate dall'ultimo Reset.
TotalThrottled	Int64	Numero totale di richieste rifiutate dalla strategia o dalla protezione burst.
TotalQuotaExceeded	Int64	Numero totale di richieste rifiutate perché è stata raggiunta una quota.
ActiveKeys	Inte- ger	Numero di chiavi attualmente tracciate (bucket + contatori finestra + connessioni attive).
Uptime	Int64	Secondi trascorsi dall'ultimo reset dell'oggetto delle statistiche.

```
// Poll from a Timer and update a dashboard
private void StatsTimer_Tick(object sender, EventArgs e)
{
```

```

labelRequests.Text = $"Requests: {rateLimiter.Stats.TotalRequests}";
labelThrottled.Text = $"Throttled: {rateLimiter.Stats.TotalThrottled}";
labelActiveKeys.Text = $"Active keys: {rateLimiter.Stats.ActiveKeys}";
labelUptime.Text = $"Uptime: {rateLimiter.Stats.Uptime} s";
}

```

Eventi

OnThrottled: Si attiva quando una richiesta viene rifiutata dalla strategia o dalla protezione burst. Fornisce la chiave (IP, chiave API o utente) e la stringa del motivo ("token_bucket", "sliding_window", "fixed_window", "burst", "connection_rate" o "message_rate"). Il parametro *Allow* (var Boolean) consente di sovrascrivere la decisione e consentire comunque la richiesta.

OnQuotaExceeded: Attivato quando viene raggiunta una quota a lungo termine. Fornisce la chiave e il Name della quota superata.

OnStateChange: Generato quando la dimensione del Token Bucket cambia per una chiave. Fornisce la chiave insieme alle dimensioni vecchia e nuova del bucket. Utile per la telemetria, il debug e le visualizzazioni in tempo reale.

Metodi pubblici

Metodo	Descrizione
IsAllowed(Key, Cost)	Restituisce True se una richiesta del Cost specificato (predefinito 1) è consentita per la chiave indicata.
Consume(Key, Cost)	Tenta di consumare i token Cost e restituisce un TsgcRateLimitResult (Allowed, Remaining, RetryAfterSec, Reason).
Reset(Key)	Azzera i contatori per la chiave specificata.
ResetAll	Azzera tutti i contatori interni per ogni chiave tracciata e reimposta l'oggetto Stats.
GetRemaining(Key)	Restituisce il numero di richieste ancora disponibili per la chiave specificata.
GetRetryAfter(Key)	Restituisce il numero di secondi che il chiamante deve attendere prima di riprovare.
SaveStateToFile(FileName)	Persiste tutti i bucket e i contatori nel file indicato.
LoadStateFromFile(FileName)	Ripristina i bucket e i contatori precedentemente salvati in un file.
SaveStateToStream(Stream)	Mantiene lo stato interno in uno stream arbitrario.
LoadStateFromStream(Stream)	Ripristina lo stato interno da un flusso arbitrario.

IsConnectionAllowed(IP)	Hook di integrazione con il server. Restituisce True se una nuova connessione da IP supera il rate limiting. Chiamato automaticamente dal server.
IsMessageAllowed(IP, Message)	Hook di integrazione con il server. Restituisce True se un messaggio proveniente da un IP supera il rate limiting. Viene chiamato automaticamente dal server.
RegisterConnection(IP)	Registra una nuova connessione per il tracciamento. Chiamato automaticamente.
UnregisterConnection(IP)	Rimuove una connessione dal tracciamento. Chiamato automaticamente alla disconnessione.

Integrazione Automatica

Quando viene assegnato alla proprietà RateLimiter di un server, il componente automaticamente:

- **Alla nuova connessione:** chiama IsConnectionAllowed per valutare PerIP, la protezione burst e la strategia globale. Rifiuta la connessione con HTTP 429 prima dell'evento OnConnect se il limite viene superato. Chiama RegisterConnection per tracciare il client.
- **Alla ricezione del messaggio:** Chiama IsMessageAllowed per verificare PerIP, PerEndpoint, PerAPIKey, PerUser, le quote e la protezione burst. Disconnette o rifiuta il messaggio quando il limite viene superato.
- **Alla disconnessione:** Chiama UnregisterConnection per rilasciare le risorse di tracciamento.

Esempio completo

```
// Enable rate limiter on server
server.RateLimiter.Enabled = true;

// Global Token Bucket: 50 burst, 10/sec sustained
server.RateLimiter.TokenBucket.Enabled = true;
server.RateLimiter.TokenBucket.Capacity = 50;
server.RateLimiter.TokenBucket.RefillRate = 10;
server.RateLimiter.TokenBucket.RefillIntervalMs = 1000;

// Per-IP sliding window: 60 requests per minute per IP
server.RateLimiter.PerIP.Enabled = true;
server.RateLimiter.PerIP.Strategy = RateLimitStrategy.Sliding;
server.RateLimiter.PerIP.MaxRequests = 60;
server.RateLimiter.PerIP.WindowSec = 60;

// Per-endpoint: expensive report endpoint = 10 req/min
server.RateLimiter.PerEndpoint.Enabled = true;
var rule = server.RateLimiter.PerEndpoint.Rules.Add();
rule.Name = "expensive-report";
rule.Pattern = "*/api/v1/expensive-report*";
rule.Strategy = RateLimitStrategy.Sliding;
rule.MaxRequests = 10;
rule.WindowSec = 60;

// Burst protection: 50 requests in 500ms triggers 30s cooldown
server.RateLimiter.BurstProtection.Enabled = true;
server.RateLimiter.BurstProtection.BurstThreshold = 50;
server.RateLimiter.BurstProtection.BurstWindowMs = 500;
server.RateLimiter.BurstProtection.CooldownSec = 30;

// Monthly quota per API key: 10000 requests/month
server.RateLimiter.Quotas.Enabled = true;
var quota = server.RateLimiter.Quotas.Quotas.Add();
quota.Name = "free-tier-monthly";
quota.Scope = RateLimitQuotaScope.PerKey;
```

```

quota.Period = RateLimitQuotaPeriod.Month;
quota.Limit = 10000;

// HTTP 429 response with Retry-After and X-RateLimit-* headers
server.RateLimiter.Response.StatusCode = 429;
server.RateLimiter.Response.Message = "Too Many Requests";
server.RateLimiter.Response.RetryAfterHeader = true;
server.RateLimiter.Response.IncludeRateLimitHeaders = true;

// Persist state across restarts
server.RateLimiter.StorageFile = "ratelimit.dat";
server.RateLimiter.LoadStateFromFile("ratelimit.dat");

// Handle throttled events
server.RateLimiter.OnThrottled += (sender, e) =>
{
    Log("Throttled " + e.Key + ": " + e.Reason);
};

// Handle quota exhausted events
server.RateLimiter.OnQuotaExceeded += (sender, e) =>
{
    Log("Quota " + e.QuotaName + " exceeded for " + e.Key);
};

```

Sicurezza dei thread

Il rate limiter è completamente thread-safe. Tutti i metodi pubblici utilizzano sezioni critiche interne e contatori thread-safe per proteggere l'accesso concorrente. Il componente può essere condiviso da più istanze server e accessibile da qualsiasi thread (gestori di eventi server, thread timer, worker in background) senza sincronizzazione esterna.

Riferimento

- Properties — 13 proprietà disponibili sul componente.
- Metodi — 12 metodi pubblici.
- Eventi — 3 eventi gestibili.

TsgcWSCircuitBreaker

TsgcWSCircuitBreaker implementa il pattern di resilienza Circuit Breaker per le chiamate lato client alle API HTTP.

Introduzione

TsgcWSCircuitBreaker implementa il pattern di resilienza Circuit Breaker per le chiamate lato client alle API HTTP. Previene i guasti a cascata quando un endpoint remoto diventa non disponibile, monitorando i guasti in una finestra mobile, aprendo il circuito una volta raggiunte le soglie configurabili e rifiutando immediatamente le chiamate durante il periodo di cooldown. Supporta soglie basate sul conteggio dei guasti e sul tasso di guasti, il rilevamento delle chiamate lente, le chiamate di prova half-open, la classificazione delle eccezioni (record / ignore), gli override per endpoint, le risposte di fallback e le metriche in tempo reale.

Il circuit breaker si integra automaticamente con qualsiasi discendente di **TsgcHTTPAPI_client** (OpenAI, Anthropic, Gemini, Grok, DeepSeek, Mistral, Ollama, Stripe, Binance, Bybit, Coinbase, Kraken e gli altri wrapper API) tramite la proprietà **CircuitBreaker**. Una volta assegnato, ogni chiamata Get / Post / Put / Patch / Delete viene gestita automaticamente. Viene mantenuto un circuito indipendente per ogni hostname univoco.

Avvio rapido

1. Trascini un componente **TsgcWSCircuitBreaker** sul form.

2. Configuri Thresholds (quando aprire) e Recovery (come ripristinare):

```
sgcWSCircuitBreaker1.Thresholds.FailureCount := 5;
sgcWSCircuitBreaker1.Recovery.CooldownSec := 30;
```

3. Assegnare il circuit breaker a un client API HTTP (qualsiasi discendente di TsgcHTTPAPI_client):

```
sgcAI_OpenAI1.CircuitBreaker := sgcWSCircuitBreaker1;
```

4. Chiamare Get / Post / Delete / Patch / Put sul client. Il breaker registra automaticamente successi e fallimenti e rifiuta le nuove chiamate mentre il circuito è Open.

Stati

Cosa fa: Il circuit breaker è una macchina a tre stati. Ogni host (chiave) ha il proprio stato indipendente che controlla se le chiamate in uscita sono consentite, rifiutate o inviate come tentativi.

Stato	Comportamento	Transizione
csClosed	Tutte le chiamate vengono elaborate. I fallimenti vengono conteggiati nella finestra temporale mobile.	Passa a csOpen quando viene raggiunto FailureCount, FailureRatePercent o Slow-CallRatePercent.
csOpen	Ogni chiamata viene rifiutata immediatamente. OnCallRejected si attiva per ogni tentativo.	Passa a csHalfOpen dopo che sono trascorsi Recovery.CooldownSec secondi dall'apertura.

csHal-fOpen	Sono consentite fino a Recovery.HalfOpenTrialCalls chiamate di prova per verificare il ripristino.	Passa a csClosed quando i tentativi riescono; torna a csOpen se un tentativo fallisce.

Proprietà di livello superiore

Funzione: Interruttore principale per il componente, oltre ai due nomi di chiave utilizzati quando non viene fornita una chiave per singola chiamata. L'integrazione lato client utilizza automaticamente il nome host di destinazione come chiave — DefaultKey e ServerKey si applicano solo alle chiamate manuali di Execute/RecordSuccess/RecordFailure e agli helper opzionali di integrazione server.

Quando utilizzarlo: Disabilitare il circuit breaker in fase di esecuzione durante un test di carico controllato o una finestra di manutenzione senza dissociarlo dal client HTTP. Impostare un DefaultKey significativo (ad esempio 'openai-api') quando si esegue Execute manualmente all'interno di codice personalizzato.

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	True	Interruttore principale. Quando è False, ogni metodo pubblico si chiude subito con "consentito" e nessun contatore viene aggiornato.
Default-Key	string	'default'	Chiave utilizzata dai sovraccarichi senza parametri di Execute, RecordSuccess e RecordFailure.
Server-Key	string	'server'	Chiave utilizzata dagli hook di integrazione server opzionali (IsConnectionAllowed, IsMessageAllowed, RecordMessageError).

```
circuitBreaker.Enabled = true;
circuitBreaker.DefaultKey = "openai-api";
circuitBreaker.ServerKey = "server";
```

Soglie

Cosa fa: Definisce le condizioni che spostano il circuito dallo stato Closed a Open. Il breaker si apre quando si raggiungono FailureCount errori grezzi, oppure quando FailureRatePercent delle chiamate nella finestra fallisce (dopo MinCalls), oppure quando SlowCallRatePercent delle chiamate supera SlowCallDurationMs.

Quando utilizzarlo: Il server chiama l'API di OpenAI. Quando si verificano 5 errori in 60 secondi (o il 50% delle chiamate fallisce dopo che almeno 10 chiamate sono state osservate), il circuito si apre e smette di sovraccaricare OpenAI — lasciando che il servizio remoto si riprenda invece di accumulare tentativi di riconnessione.

Proprietà	Type	Predefinito	Descrizione

Abilitato	Boolean	True	Abilita l'apertura basata su soglia. Quando è False il circuito non si apre mai automaticamente.
FailureCount	Integer	5	Numero assoluto di errori nella finestra mobile che fa scattare il circuit breaker.
FailureRatePercent	Integer	50	Percentuale di fallimento (0-100) che fa scattare il breaker una volta raggiunto MinCalls.
SlowCallDurationMs	Integer	3000	Soglia (ms) oltre la quale una chiamata riuscita viene conteggiata anche come "lenta".
SlowCallRatePercent	Integer	0	Percentuale (0-100) di chiamate lente che attiva il circuit breaker. 0 disabilita il rilevamento delle chiamate lente.
MinCalls	Integer	10	Numero minimo di chiamate osservate prima che vengano valutate le soglie di frequenza (evita di scattare su campioni ridotti).

```
// Open after 5 failures, or 50% failure rate on at least 10 calls
circuitBreaker.Thresholds.Enabled = true;
circuitBreaker.Thresholds.FailureCount = 5;
circuitBreaker.Thresholds.FailureRatePercent = 50;
circuitBreaker.Thresholds.MinCalls = 10;
// Also treat any call slower than 3 seconds as a "slow" call
circuitBreaker.Thresholds.SlowCallDurationMs = 3000;
circuitBreaker.Thresholds.SlowCallRatePercent = 60;
```

TimeWindow

Cosa fa: Definisce la finestra temporale scorrevole utilizzata per la valutazione della soglia. La finestra è divisa in sotto-bucket BucketCount in modo che i contatori vecchi vengano eliminati progressivamente con l'avanzare del tempo. Le soglie vengono sempre valutate rispetto alla somma dei bucket attivi.

Quando utilizzarlo: Utilizzare una finestra mobile di 30 secondi con 6 bucket in modo che i picchi transitori non aprano il circuit breaker, ma i guasti prolungati sì. Finestre più lunghe attenuano il rumore; finestre più brevi reagiscono più rapidamente ai malfunzionamenti. Un numero maggiore di bucket produce un aging più uniforme ma consuma leggermente più memoria per host.

Proprietà	Type	Predefinito	Descrizione
RollingWindowSec	Integer	60	Larghezza totale della finestra temporale in secondi. I campioni più vecchi vengono scartati.
BucketCount	Integer	10	Numero di sotto-bucket in cui è suddivisa la finestra. Ogni bucket contiene i contatori di successo / fallimento / lentezza.

```
// 30-second window split into 6 buckets (5 seconds each)
circuitBreaker.TimeWindow.RollingWindowSec = 30;
circuitBreaker.TimeWindow.BucketCount = 6;
```

Ripristino

Cosa fa: Controlla il modo in cui il circuit breaker si ripristina dallo stato Aperto. Dopo che è trascorso CooldownSec, il breaker passa a HalfOpen e consente il transito di HalfOpenTrialCalls chiamate di verifica. Se le prove hanno esito positivo, il breaker si chiude; in caso di fallimento si riapre e il timer di cooldown riparte.

Quando usarlo: dopo l'apertura, attendere 30 secondi e poi lasciar passare una chiamata di prova per verificare il ripristino prima di chiudere nuovamente. Un cooldown più lungo protegge un upstream in difficoltà da un'altra ondata di traffico; un cooldown breve riapre rapidamente per servizi che hanno solo brevi interruzioni.

Proprietà	Type	Predefinito	Descrizione
CooldownSec	Integer	30	Secondi di attesa nello stato Open prima di passare a HalfOpen.
HalfOpenTrialCalls	Integer	1	Numero di chiamate di prova consentite in HalfOpen prima di decidere di chiudere o riaprire il circuito.
AutoReset	Boolean	True	Quando True, i contatori vengono azzerati automaticamente alle transizioni di stato.
MaxRetries	Integer	3	Limite informativo utilizzato dagli helper di ripetizione. Il circuit breaker stesso non esegue alcun tentativo automatico.

```
// After opening: wait 30s, then allow 1 trial call, then decide
circuitBreaker.Recovery.CooldownSec = 30;
circuitBreaker.Recovery.HalfOpenTrialCalls = 1;
circuitBreaker.Recovery.AutoReset = true;
circuitBreaker.Recovery.MaxRetries = 3;
```

Fallback

Funzione: Configura un payload alternativo servito mentre il circuito è Aperto. Quando Fallback.Enabled è True ed Execute / ExecuteWithResult rifiuta una chiamata, l'evento OnFallback viene attivato con una risposta inizializzata da CachedResponse (o CustomMessage). I gestori possono sostituire questa stringa prima che venga restituita al chiamante.

Quando utilizzarlo: Restituire una risposta last-known-good memorizzata in cache quando OpenAI non è disponibile, in modo che gli utenti vedano un comportamento degradato ma funzionale (per esempio il riepilogo generato in precedenza) anziché un errore. Utilizzi CustomMessage per un body JSON statico di "modalità manutenzione".

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	False	Abilita il percorso di fallback. Quando è False, le chiamate rifiutate ritornano semplicemente senza attivare OnFallback.
CachedResponse	string	"	Payload statico restituito mentre il circuito è Aperto. Ha la precedenza su CustomMessage.
CustomMessage	string	"	Payload di fallback utilizzato quando CachedResponse è vuoto.
UseLastSuccess	Boolean	False	Quando True, viene restituita come fallback l'ultima risposta corretta osservata per la chiave.

```
// When api.openai.com is down, serve a safe JSON body instead of erroring
circuitBreaker.Fallback.Enabled = true;
circuitBreaker.Fallback.CachedResponse =
  "{\"error\":\"service unavailable\",\"retry_after\":30}";
circuitBreaker.Fallback.CustomMessage =
  "{\"error\":\"ai service temporarily offline\"}";
circuitBreaker.Fallback.UseLastSuccess = true;
```

Classificazione

Funzione: Classifica le eccezioni sollevate come "registrate come errore" o "ignorate". Solo le eccezioni registrate incrementano i contatori degli errori. MatchMode seleziona il modo in cui le stringhe in RecordAsFailure / IgnoreExceptions vengono confrontate con il testo dell'eccezione (ClassName: Message).

Quando utilizzarlo: Trattare `EidHttpException` con 500/502/503/504 come errori, ma ignorare `EidHttpException` 400/401/404 (gli errori di business non devono attivare il circuito). Lasciare RecordAsFailure vuoto significa che ogni eccezione non ignorata viene registrata.

Proprietà	Type	Predefinito	Descrizione
RecordAsFailure	TStringList	vuoto	Pattern confrontati con il testo delle eccezioni; le eccezioni corrispondenti vengono conteggiate come fallimenti. Elenco vuoto = registra tutto ciò che non viene ignorato.
IgnoreExceptions	TStringList	vuoto	Pattern confrontati con il testo dell'eccezione; le eccezioni corrispondenti vengono eliminate (né come successo né come errore).
Match-Mode	TsgcCircuitBreakerExceptionMatch	cem-Contains	Modalità di confronto: cemExact, cemContains, cemStartsWith o cemWildcard (* / ?).

```
// Record HTTP 5xx and timeouts as failures; ignore client-side 4xx
circuitBreaker.Classification.MatchMode = ExceptionMatch.Contains;
circuitBreaker.Classification.RecordAsFailure.Add("HTTP/1.1 500");
circuitBreaker.Classification.RecordAsFailure.Add("HTTP/1.1 502");
circuitBreaker.Classification.RecordAsFailure.Add("HTTP/1.1 503");
circuitBreaker.Classification.RecordAsFailure.Add("HTTP/1.1 504");
circuitBreaker.Classification.RecordAsFailure.Add("timeout");
circuitBreaker.Classification.IgnoreExceptions.Add("HTTP/1.1 400");
circuitBreaker.Classification.IgnoreExceptions.Add("HTTP/1.1 401");
circuitBreaker.Classification.IgnoreExceptions.Add("HTTP/1.1 404");
```

PerEndpoint

Cosa fa: Raccolta di override basati su pattern. Ogni `TsgcCircuitBreakerEndpointItem` ha il proprio Pattern (sono supportati i caratteri jolly * e ?), un blocco `OverrideThresholds` e un blocco `OverrideRecovery`. Quando la chiave attiva (hostname) corrisponde a un elemento abilitato, le impostazioni di quell'elemento sostituiscono Thresholds / Recovery di livello superiore per la durata della chiamata.

Quando utilizzarlo: Circuiti separati per ogni API: `api.openai.com` ha soglie permissive (occasionalmente rallenta sotto carico), ma `api.stripe.com` è critica per i pagamenti e dovrebbe aprirsi più rapidamente per fallire in modo sicuro (fail closed). Aggiunga un elemento endpoint per ogni host con soglie ottimizzate.

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	False	Abilita la raccolta di override degli endpoint. Quando è False non viene valutato alcun elemento.
Elementi	TCollection	vuoto	Raccolta di <code>TsgcCircuitBreakerEndpointItem</code> . Il primo elemento abilitato corrispondente ha la precedenza.

Ogni `TsgcCircuitBreakerEndpointItem` espone:

Proprietà	Type	Predefinito	Descrizione
Pattern	string	"	Pattern wildcard confrontato con la chiave (nome host). Supporta * e ?.
Abilitato	Boolean	True	Disabilita questo elemento senza eliminarlo.
OverrideThresholds	TsgcCircuitBreakerThresholds	defaults	Soglie applicate quando il pattern corrisponde. Attive solo quando il proprio flag <code>Enabled</code> è True.
OverrideRecovery	TsgcCircuitBreakerRecovery	defaults	Impostazioni di recovery applicate quando il pattern corrisponde.

```

circuitBreaker.PerEndpoint.Enabled = true;

// OpenAI - lenient: 10 failures, 60s cooldown
var openai = circuitBreaker.PerEndpoint.Items.Add();
openai.Pattern = "api.openai.com";
openai.Enabled = true;
openai.OverrideThresholds.Enabled = true;
openai.OverrideThresholds.FailureCount = 10;
openai.OverrideThresholds.FailureRatePercent = 70;
openai.OverrideRecovery.CooldownSec = 60;

// Stripe - payment critical: open fast, recover slow
var stripe = circuitBreaker.PerEndpoint.Items.Add();
stripe.Pattern = "api.stripe.com";
stripe.Enabled = true;
stripe.OverrideThresholds.Enabled = true;
stripe.OverrideThresholds.FailureCount = 3;
stripe.OverrideThresholds.FailureRatePercent = 30;
stripe.OverrideRecovery.CooldownSec = 120;

```

Metriche

Cosa fa: Contatori di sola lettura aggiornati atomicamente man mano che le chiamate fluiscono attraverso il breaker. Aggregati su tutte le chiavi / hostname. CurrentOpenBreakers viene aggiornato dopo ogni transizione di stato; AverageLatencyMs è una media mobile delle durate osservate da Execute.

Quando utilizzarlo: Dashboard in tempo reale che mostra gli stati del circuito per host, i tassi di errore e i conteggi dei rifiuti. Faccia il polling di Metrics da un TTimer per disegnare indicatori per gli operatori, oppure li emetta verso Prometheus / StatsD per gli alert.

Proprietà	Type	Descrizione
TotalCalls	Int64	Totale chiamate elaborate (successi + errori registrati).
TotalFailures	Int64	Totale dei fallimenti registrati su tutte le chiavi.
TotalSuccesses	Int64	Totale delle chiamate riuscite registrate su tutte le chiavi.
TotalRejected	Int64	Numero totale di chiamate rifiutate perché il circuito era aperto.
CurrentOpen-Breakers	Integer	Numero di chiavi attualmente nello stato Aperto.
AverageLatencyMs	Double	Latenza media in esecuzione (ms) misurata attorno alle chiamate Execute / ExecuteWithResult.

```

// Poll from a Timer and update a dashboard
private void MetricsTimer_Tick(object sender, EventArgs e)
{
    labelCalls.Text = $"Calls: {circuitBreaker.Metrics.TotalCalls} " +
        $"(ok {circuitBreaker.Metrics.TotalSuccesses} / " +
        $"fail {circuitBreaker.Metrics.TotalFailures} / " +
        $"rejected {circuitBreaker.Metrics.TotalRejected})";
    labelOpen.Text = $"Open breakers: {circuitBreaker.Metrics.CurrentOpenBreakers}";
    labelLatency.Text = $"Avg latency: {circuitBreaker.Metrics.AverageLatencyMs:0.0} ms";
}

```

Eventi

OnStateChange: Generato quando un circuit breaker cambia stato. Fornisce la chiave, lo stato precedente e quello nuovo (csClosed, csOpen o csHalfOpen). Utile per la registrazione, gli avvisi e l'aggiornamento dell'interfaccia utente in tempo reale.

OnCallRejected: Attivato quando una chiamata viene rifiutata perché il circuito è Open o il budget di prova HalfOpen è esaurito. Fornisce la chiave e una stringa di motivazione (per esempio `circuit open`).

OnFallback: Attivato appena prima che venga restituita una risposta di fallback. Il parametro *aResponse* (var string) viene inizializzato da `Fallback.CachedResponse / CustomMessage` e può essere sostituito dal gestore.

OnFailureRecorded: Attivato ogni volta che viene registrato un errore. Fornisce la chiave e il testo dell'eccezione (`ClassName: Message`). Da utilizzare per il logging strutturato o per correlarlo al tracciamento degli incidenti a monte.

OnSlowCall: Generato quando una chiamata riuscita ha superato `Thresholds.SlowCallDurationMs`. Fornisce la chiave e la durata misurata in millisecondi.

Metodi pubblici

Metodo	Descrizione
<code>Execute(Key, Action)</code>	Esegue il TProc sotto la chiave specificata. Controlla prima <code>IsCallAllowed</code> , registra automaticamente successo / fallimento / lentezza e restituisce True se l'azione è stata eseguita.
<code>Execute(Action)</code>	Overload che utilizza <code>DefaultKey</code> .
<code>ExecuteWithResult(Key, Action, out Result)</code>	(D2009+) Esegue un <code>TFunc<TObject></code> e restituisce il risultato tramite il parametro out; stessa contabilità di <code>Execute</code> .
<code>RecordSuccess(Key)</code>	Registra una chiamata riuscita. Avanza da <code>HalfOpen</code> a <code>Closed</code> quando i tentativi hanno successo.
<code>RecordSuccess</code>	Overload che utilizza <code>DefaultKey</code> .
<code>RecordFailure(Key, Exception)</code>	Registra un errore con testo di eccezione opzionale; attiva la valutazione della transizione.
<code>RecordFailure(Exception)</code>	Overload che utilizza <code>DefaultKey</code> .
<code>ForceOpen(Key)</code>	Forza manualmente il circuit in stato Open.
<code>ForceClose(Key)</code>	Forza manualmente il circuit in stato Closed.
<code>Reset(Key)</code>	Azzera i contatori e lo stato per la chiave specificata.
<code>ResetAll</code>	Cancella ogni circuito tracciato e azzera le Metriche.
<code>GetState(Key)</code>	Restituisce il <code>TsgcCircuitState</code> corrente per la chiave.

IsCallAllowed(Key)	Restituisce True se è attualmente consentita una nuova chiamata. Viene chiamato automaticamente dal client API HTTP prima di ogni richiesta.
GetFailureRate(Key)	Restituisce la percentuale di errori corrente (0,0 - 100,0) osservata nella finestra temporale scorrevole.
GetTotalCalls(Key)	Restituisce il numero di chiamate osservate nella finestra temporale scorrevole per la chiave.
GetLastStateChange(Key)	Restituisce il TDateTime dell'ultima transizione di stato.
GetOpenBreakers	Restituisce un TArray<string> delle chiavi attualmente nello stato Open.
SaveStateToFile(FileName)	Salva lo stato e i contatori di ogni circuito su un file.
LoadStateFromFile(FileName)	Ripristina gli stati dei circuiti da un file precedentemente salvato.
IsConnectionAllowed(IP)	Hook di auto-protezione lato server. Secondario rispetto all'utilizzo principale lato client. Restituisce True se il circuito ServerKey non è aperto.
IsMessageAllowed(IP, Message)	Hook di auto-protezione lato server. Restituisce True se il circuito ServerKey non è aperto.
RegisterConnection(IP)	Hook lato server riservato per future metriche per IP (attualmente non operativo).
UnregisterConnection(IP)	Hook lato server riservato per future metriche per IP (attualmente non operativo).
RecordMessageError(IP, Exception)	Hook di auto-protezione lato server. Registra un errore su Server-Key.
RecordMessageSuccess(IP)	Hook di auto-protezione lato server. Registra un successo su Server-Key.

Integrazione Automatica

Quando assegnato a un discendente di TsgcHTTPAPI_client tramite la proprietà CircuitBreaker, il componente automaticamente:

- **Prima di ogni Get / Post / Put / Patch / Delete:** Ricava la chiave dal nome host dell'URL di destinazione (ad esempio `api.openai.com`) e chiama `IsCallAllowed(host)`. Se il circuito è aperto, la chiamata viene rifiutata con l'eccezione *"Circuit breaker open for <host> - call rejected"* e viene generato `OnCallRejected`.
- **In caso di risposta HTTP riuscita:** Chiama `RecordSuccess(host)` per aggiornare i contatori, consentendo ai tentativi `HalfOpen` di chiudere il circuito quando il servizio remoto si ripristina.
- **In caso di eccezione:** Chiama `RecordFailure(host, exception.Message)`, applica le regole di `Classification` per decidere se conteggiare o ignorare, valuta le transizioni e quindi rilancia l'eccezione al chiamante.
- **Un circuito per host:** Le chiamate a host diversi (ad esempio `api.openai.com`, `api.anthropic.com`, `api.stripe.com`) tracciano circuiti completamente indipendenti.

Esempio completo

```
// Enable the circuit breaker
circuitBreaker.Enabled = true;

// Thresholds: open after 5 failures, or 50% failure rate on 10+ calls
circuitBreaker.Thresholds.Enabled = true;
circuitBreaker.Thresholds.FailureCount = 5;
circuitBreaker.Thresholds.FailureRatePercent = 50;
circuitBreaker.Thresholds.SlowCallDurationMs = 3000;
circuitBreaker.Thresholds.MinCalls = 10;

// Rolling window: 60 seconds / 10 buckets
circuitBreaker.TimeWindow.RollingWindowSec = 60;
circuitBreaker.TimeWindow.BucketCount = 10;

// Recovery: wait 30s in Open, then 1 trial call in HalfOpen
circuitBreaker.Recovery.CooldownSec = 30;
circuitBreaker.Recovery.HalfOpenTrialCalls = 1;

// Classification: record HTTP 5xx and timeouts, ignore 4xx
circuitBreaker.Classification.MatchMode = ExceptionMatch.Contains;
circuitBreaker.Classification.RecordAsFailure.Add("HTTP/1.1 5");
circuitBreaker.Classification.RecordAsFailure.Add("timeout");
circuitBreaker.Classification.IgnoreExceptions.Add("HTTP/1.1 404");

// Fallback: serve a cached payload while the circuit is Open
circuitBreaker.Fallback.Enabled = true;
circuitBreaker.Fallback.CustomMessage = "{\"error\":\"service unavailable\"}";

// Per-endpoint overrides
circuitBreaker.PerEndpoint.Enabled = true;
var stripe = circuitBreaker.PerEndpoint.Items.Add();
stripe.Pattern = "api.stripe.com";
stripe.OverrideThresholds.Enabled = true;
stripe.OverrideThresholds.FailureCount = 3;
stripe.OverrideRecovery.CooldownSec = 120;

// Persist circuit state across restarts
circuitBreaker.LoadStateFromFile("circuit.dat");

// Attach to an HTTP API client (any TsgcHTTPAPI_client descendant)
client.CircuitBreaker = circuitBreaker;

// Use the client normally; the breaker intercepts Get / Post / etc.
try
{
    string response = client.Get("https://api.openai.com/v1/models");
}
catch (Exception ex)
{
    if (ex.Message.Contains("Circuit breaker open"))
        Log("Circuit open for api.openai.com - using fallback");
    else
        Log("Call failed: " + ex.Message);
}

// Handle state changes
circuitBreaker.OnStateChange += (sender, e) =>
{
    Log($"Circuit {e.Key}: {e.OldState} -> {e.NewState}");
};

// Handle fallback responses
circuitBreaker.OnFallback += (sender, e) =>
{
    Log("Fallback used for " + e.Key);
};
```

Sicurezza dei thread

Il circuit breaker è completamente thread-safe. Tutti i metodi pubblici utilizzano sezioni critiche interne e liste thread-safe per proteggere l'accesso concorrente allo stato per chiave, alla finestra mobile e ai contatori delle metri-

che. Un singolo componente può essere condiviso da più client API HTTP e a cui si accede da qualsiasi thread (gestori di eventi, timer, worker in background) senza sincronizzazione esterna.

Riferimento

- Proprietà — 10 proprietà disponibili sul componente.
- Metodi — 17 metodi pubblici.
- Eventi — 5 eventi che Lei può gestire.

TsgcWSAPIKeyManager

TsgcWSAPIKeyManager fornisce una gestione completa del ciclo di vita per le chiavi API emesse dai server sgc-WebSockets.

Introduzione

TsgcWSAPIKeyManager fornisce una gestione completa del ciclo di vita per le chiavi API emesse dai server sgc-WebSockets. Genera chiavi casuali crittograficamente sicure con un prefisso leggibile dall'utente configurabile, le esegue l'hashing a riposo con SHA-256, SHA-512 o Bcrypt, convalida le richieste in arrivo tramite intestazione o stringa di query, ruota le chiavi con un periodo di grazia configurabile, revoca e rinnova le chiavi, applica l'autorizzazione basata su scope, mantiene un registro di audit persistente e supporta un elenco IP consentiti opzionale. Le chiavi vengono memorizzate in memoria o in un file crittografato che sopravvive ai riavvii.

Il manager si collega a TsgcWebSocketHTTPServer e TsgcWSServer_HTTPAPI tramite la proprietà **APIKeyManager**. Poiché gli hook automatici del server vedono solo l'IP del peer, l'autenticazione effettiva delle richieste viene eseguita chiamando **IsRequestAuthorized** dall'evento OnConnect del server — questo fornisce accesso al blocco completo delle intestazioni HTTP e all'URL/stringa di query, in modo che l'intestazione **X-API-Key** configurata (o il parametro di query) possa essere analizzata, validata e, se richiesto, confrontata con un ambito richiesto.

Avvio rapido

1. Rilasciare un componente **TsgcWSAPIKeyManager** sul form.
2. Configurare Generation (prefisso, lunghezza), Hashing (algoritmo, salt) e Validation (nome dell'intestazione, allowlist di IP, FailClosed). Veda la documentazione per sezione qui sotto.
3. Assegnare il manager a un componente server:

```
sgcWebSocketHTTPServer1.APIKeyManager := sgcWSAPIKeyManager1;
```

4. Collegare l'evento OnConnect del server per chiamare **IsRequestAuthorized** utilizzando le intestazioni, l'URL e l'IP della connessione. Disconnettere il client quando restituisce False.
5. Emettere la prima chiave per un cliente e consegnare il valore in chiaro esattamente una volta — solo la forma hash rimane in archivio in seguito.

Proprietà di livello superiore

Cosa fa: Interruttore principale dell'intero componente. Quando Enabled è False, ogni metodo pubblico viene cortocircuitato su "consentito" e gli hook automatici del server diventano no-op, quindi il key manager è di fatto bypassato.

Quando utilizzarlo: Disabilitare temporaneamente l'applicazione delle chiavi durante una finestra di manutenzione controllata o una distribuzione di staging senza disassociare il componente dal server. Lo riabiliti quando la modifica è completata — nessuna chiave viene persa, nessun ambito cambia e il log di audit continua a funzionare nel momento in cui lo riattiva.

Proprietà	Type	Predefinito	Descrizione

Abilitato	Boolean	True	Interruttore principale. Quando è False, ValidateKey / IsRequestAuthorized restituiscono sempre True e i ganci automatici del server diventano operazioni nulle.

```
apiKeyManager.Enabled = true;
```

Generazione

Funzione: Controlla come viene costruita la stringa di chiave in testo normale da *IssueKey*. La chiave è un *Key-Prefix* seguito da *KeyLength* caratteri casuali tratti dal *Charset*. Quando *IncludeChecksum* è True, viene aggiunto un breve checksum in modo che le chiavi malformate possano essere rifiutate rapidamente prima di una ricerca hash.

Quando utilizzarlo: Chiavi con brand per il proprio SaaS — utilizzare *sgc_live_aB3...* in produzione e *sgc_test_...* nel cluster di staging. Il prefisso rende immediatamente riconoscibile al personale di supporto a quale ambiente appartiene una chiave compromessa e consente agli strumenti di redazione dei log di oscurarla.

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	True	Quando False, <i>IssueKey</i> genera un'eccezione anziché creare una nuova chiave.
KeyPrefix	string	"	Prefisso leggibile dall'utente anteposto al corpo casuale (ad esempio <i>sgc_live_</i>).
Key-Length	Integer	32	Numero di caratteri casuali nel corpo della chiave.
Charset	string	A-Za-z0-9	Alfabeto usato per generare il body casuale. Se vuoto, per impostazione predefinita usa caratteri alfanumerici URL-safe.
Include-Checksum	Boolean	False	Se True, aggiunge un breve suffisso di checksum in modo che le chiavi palesemente non valide possano essere rifiutate senza una ricerca tramite hash.

```
// Branded production keys: sgc_live_aB3...
apiKeyManager.Generation.Enabled = true;
apiKeyManager.Generation.KeyPrefix = "sgc_live_";
apiKeyManager.Generation.KeyLength = 32;
apiKeyManager.Generation.IncludeChecksum = true;
```

Storage

Funzione: Controlla dove risiedono le chiavi con hash e il log di audit. *kstMemory* mantiene tutto nel processo (cancellato all'arresto). *kstFile* salva in *FileName*, opzionalmente cifrato con *EncryptionKey*, e si risalva ogni *AutoSaveSeconds* secondi. *kstCustom* consente di implementare hook personalizzati *SaveToFile* / *LoadFromFile*.

Quando utilizzarlo: Conservare le chiavi in memoria durante lo sviluppo per iterazioni rapide. Passare a *kstFile* con *EncryptAtRest* := *True* in produzione, in modo che le chiavi sopravvivano a un riavvio del server e non vengano mai esposte in testo normale su disco.

Proprietà	Type	Predefinito	Descrizione
Storage-Type	TsgcAPIKey-StorageType	kstMemory	kstMemory (volatile), kstFile (persistente su FileName) o kstCustom (SaveToFile / LoadFromFile forniti dall'utente).
FileName	string	"	Percorso del file utilizzato da SaveToFile / LoadFromFile quando StorageType è kstFile.
EncryptAtRest	Boolean	False	Quando è True cifra l'archivio delle chiavi su disco utilizzando EncryptionKey.
EncryptionKey	string	"	Segreto utilizzato per cifrare il file della chiave persistente. Deve essere fornito quando EncryptAtRest è True.
AutoSaveSeconds	Integer	0	Quando > 0, il manager salva nuovamente il key store su disco ogni AutoSaveSeconds secondi.

```
// Production: encrypted file that survives restarts, auto-saved every 60s
apiKeyManager.Storage.StorageType = StorageType.File;
apiKeyManager.Storage.FileName = "apikeys.dat";
apiKeyManager.Storage.EncryptAtRest = true;
apiKeyManager.Storage.EncryptionKey = "my-very-secret-master-key";
apiKeyManager.Storage.AutoSaveSeconds = 60;
apiKeyManager.LoadFromFile("apikeys.dat");
```

Hashing

Cosa fa: Le chiavi in chiaro non vengono mai memorizzate. *IssueKey* esegue l'hash della chiave generata utilizzando l'*Algorithm* con un *Salt* opzionale e un conteggio di *Iterations*; *ValidateKey* ricalcola l'hash del valore in arrivo ed esegue un confronto a tempo costante rispetto al digest memorizzato.

Quando usarlo: Non memorizzare mai in testo normale. SHA-256 è il valore predefinito ragionevole per la maggior parte delle distribuzioni. Utilizzare SHA-512 per profili di sicurezza più elevati (FIPS 140-2 Level 3). Utilizzare Bcrypt quando è necessario uno stretching deliberato della chiave — la validazione è più lenta ma rende molto più difficile il brute-force offline di un archivio chiavi compromesso.

Proprietà	Type	Predefinito	Descrizione

Algoritmo	TsgcAPIKeyHashAlgorithm	kha-SHA256	Funzione hash utilizzata per l'archiviazione a riposo: kha-SHA256, khaSHA512 o khaBcrypt.
Salt	string	"	Salt statico mescolato in ogni hash. La modifica invalida tutte le chiavi esistenti.
Iterazioni	Integer	1	Numero di round di hashing. Aumenti il valore per il key-stretching a scapito della latenza di validazione.

```
// SHA-512 with a static salt and 10k iterations for key stretching
apiKeyManager.Hashing.Algorithm = HashAlgorithm.SHA512;
apiKeyManager.Hashing.Salt = "my-salt";
apiKeyManager.Hashing.Iterations = 10000;
```

Scopes

Cosa fa: Catalogo delle stringhe di scope consentite. Quando *Enabled* è True, ogni scope associato a una chiave emessa deve esistere in questo elenco; gli scope sconosciuti vengono rifiutati. Le singole voci del catalogo possono essere disabilitate senza eliminarle. *ValidateKey* / *IsRequestAuthorized* può richiedere uno scope specifico per richiesta.

Quando utilizzarlo: Permessi granulari. La chiave di un cliente contiene solo *read:orders* e verrà rifiutata su un endpoint di scrittura. Un'integrazione partner riceve *read:orders* + *write:shipments* ma mai *admin:**. Ogni richiesta presenta un suggerimento di scope e il gestore lo applica in modo coerente.

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	True	Abilita l'applicazione degli scope. Quando è False, i parametri degli scope vengono ignorati.
Scopes	TCollection	vuoto	Raccolta di voci <i>TsgcAPIKeyScopeItem</i> che definiscono gli scope consentiti.

Ogni **TsgcAPIKeyScopeItem** espone:

Proprietà	Type	Predefinito	Descrizione
Nome	string	"	Identificatore di scope (ad esempio <i>read:orders</i>). Distingue tra maiuscole e minuscole.
Descrizione	string	"	Descrizione leggibile dall'utente mostrata nell'object inspector e nelle interfacce di amministrazione.
Abilitato	Boolean	True	Disabilita questo ambito senza eliminarlo. Gli ambiti disabilitati non soddisfano mai <i>HasScope</i> .

```
apiKeyManager.Scopes.Enabled = true;

var scopeRead = apiKeyManager.Scopes.Scopes.Add();
scopeRead.Name = "read:orders";
scopeRead.Description = "List and read order history";

var scopeWrite = apiKeyManager.Scopes.Scopes.Add();
scopeWrite.Name = "write:shipments";
scopeWrite.Description = "Create and update shipment records";
```

Rotazione

Cosa fa: Rende trasparente la rotazione delle chiavi. `RotateKey` emette una nuova chiave per lo stesso proprietario e gli stessi scope e contrassegna quella vecchia come *kksRotated*. Durante la finestra *GracePeriodSec* entrambe le chiavi sono valide, così il cliente può effettuare lo scambio senza tempi di inattività. Quando *AutoRotateDays* è > 0, le chiavi più vecchie di tale soglia vengono ruotate dalla scansione in background; *NotifyBeforeExpirySec* attiva `OnKeyExpired` in anticipo, così può inviare un'e-mail al cliente.

Quando utilizzarlo: Un cliente segnala che la propria chiave è stata divulgata, ma la sua integrazione è ancora attiva. Rinnovi la chiave: ne riceverà una nuova, quella vecchia continuerà a funzionare per 24 ore in modo che possa aggiornarla nella propria infrastruttura, dopodiché quella vecchia verrà invalidata automaticamente.

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	False	Abilita le funzionalità di rotazione (periodo di tolleranza e scansione di rotazione automatica).
GracePeriodSec	Integer	86400	Secondi durante i quali la chiave precedente rimane valida dopo <code>RotateKey</code> . Il valore predefinito è 24 ore.
AutoRotateDays	Integer	0	Quando > 0, la scansione in background ruota le chiavi più vecchie di questo numero di giorni.
NotifyBeforeExpirySec	Integer	604800	Attiva <code>OnKeyExpired</code> questo numero di secondi prima che una chiave scada effettivamente. Il valore predefinito è 7 giorni.

```
// 24h grace period, auto-rotate keys older than 90 days, notify 7 days ahead
apiKeyManager.Rotation.Enabled = true;
apiKeyManager.Rotation.GracePeriodSec = 86400;
apiKeyManager.Rotation.AutoRotateDays = 90;
apiKeyManager.Rotation.NotifyBeforeExpirySec = 7 * 86400;
```

Scadenza

Cosa fa: controlla come le chiavi emesse scadono. *DefaultTTLSec* viene applicato a `IssueKey` quando il chiamante passa 0 come `ExpiresInSec`. *EnforceExpiry* indica se le chiavi scadute vengono rifiutate. *CheckIntervalSec* con-

trolla la frequenza con cui la scansione in background cerca le voci scadute, genera OnKeyExpired e aggiorna Stats.

Quando utilizzarlo: Le chiavi di integrazione partner devono scadere dopo 90 giorni — i partner devono re-firmare l'accordo e recuperare una nuova chiave. Impostare DefaultTTLSec a 90×86400 significa che ogni chiamata IssueKey rispetta tale policy, a meno che il chiamante non la sostituisca esplicitamente.

Proprietà	Type	Predefinito	Descrizione
DefaultTTLSec	Integer	0	Tempo di vita predefinito per le chiavi appena emesse, in secondi. 0 significa che non scadono mai.
EnforceExpiry	Boolean	True	Quando True, ValidateKey rifiuta le chiavi il cui ExpiresAt è nel passato.
CheckIntervalSec	Integer	60	Con quale frequenza la scansione in background controlla la presenza di chiavi scadute e attiva OnKeyExpired.

```
// Partner keys expire after 90 days, enforced strictly, swept every minute
apiKeyManager.Expiration.DefaultTTLSec = 90 * 86400;
apiKeyManager.Expiration.EnforceExpiry = true;
apiKeyManager.Expiration.CheckIntervalSec = 60;
```

RateLimit

Cosa fa: Memorizza i limiti di velocità predefiniti per chiave (numero massimo di richieste per finestra temporale) accanto a ogni chiave. Questi valori sono suggerimenti informativi che un'istanza TsgcWSRateLimiter può leggere tramite lo scope PerAPIKey per applicare una limitazione appropriata al livello — il gestore delle chiavi API stesso non limita il traffico.

Quando utilizzarlo: SaaS a livelli. I clienti del livello gratuito sono configurati a 100 req/ora, Pro a 10.000 req/ora, Enterprise a 1.000.000 req/ora. Configurare DefaultMaxRequests / DefaultWindowSec sul gestore delle chiavi API e abbinarlo a TsgcWSRateLimiter.PerAPIKey in modo che il rate limiter legga automaticamente il budget per chiave.

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	False	Abilita metadati di limite di frequenza per chiave insieme a ogni chiave emessa.
DefaultMaxRequests	Integer	100	Numero massimo predefinito di richieste per finestra assegnato alle nuove chiavi.
DefaultWindowSec	Integer	60	Dimensione predefinita della finestra temporale in secondi associata a DefaultMaxRequests.


```
// Default tier: 100 requests per hour per API key
apiKeyManager.RateLimit.Enabled = true;
apiKeyManager.RateLimit.DefaultMaxRequests = 100;
apiKeyManager.RateLimit.DefaultWindowSec = 3600;
// Pair with TsgcWSRateLimiter.PerAPIKey for actual throttling
```

Audit

Cosa fa: Registra ogni azione del ciclo di vita delle chiavi (emessa, convalidata, revocata, ruotata, scaduta, ambito concesso/revocato, convalida fallita) nel buffer circolare in memoria e, quando *LogFile* è impostato, aggiunge le stesse voci a un file. *IncludeIP* registra l'IP del richiedente; *IncludePayload* acquisisce dettagli aggiuntivi. *Retention-Days* elimina le voci più vecchie; *MaxMemoryEntries* limita il buffer circolare.

Quando utilizzarlo: Requisito di conformità: ogni operazione sulle chiavi deve essere registrata e conservata per 12 mesi. Abilitare il controllo con *LogFile* che punta alla propria zona di raccolta SIEM e *RetentionDays* impostato a 365. *OnAuditEvent* viene generato per ogni voce, in modo da poterle inviare anche a un bus di messaggi.

Proprietà	Type	Predefinito	Descrizione
Abilitato	Boolean	True	Abilita il registro di audit. Se False, non viene registrata alcuna voce e <i>OnAuditEvent</i> non viene mai generato.
LogFile	string	"	Percorso file opzionale a cui viene aggiunto il log di audit.
IncludeIP	Boolean	True	Se True, ogni voce registra l'IP del richiedente.
Include-Payload	Boolean	False	Quando True, il campo Details di ogni voce di audit contiene un contesto aggiuntivo.
RetentionDays	Integer	90	Le voci più vecchie di questo numero di giorni vengono eliminate dalla scansione in background.
MaxMemoryEntries	Integer	10000	Numero massimo di voci mantenute nel buffer circolare in memoria.

```
// Compliance: 12-month retention, log to file, include IP + payload
apiKeyManager.Audit.Enabled = true;
apiKeyManager.Audit.LogFile = "apikey-audit.log";
apiKeyManager.Audit.IncludeIP = true;
apiKeyManager.Audit.IncludePayload = true;
apiKeyManager.Audit.RetentionDays = 365;
apiKeyManager.Audit.MaxMemoryEntries = 50000;
```

Validazione

Funzione: Indica al manager come i client presentano le proprie chiavi e in quali condizioni le richieste vengono accettate. *HeaderName* (predefinito `X-API-Key`) e *QueryParamName* (predefinito `api_key`) sono i due punti in cui *ExtractKeyFromHeaders* / *ExtractKeyFromQuery* effettuano la ricerca. *RequireHTTPS* rifiuta il traffico in chiaro. *IPAllowlist* limita le chiavi a specifici IP sorgente. *FailClosed* (predefinito `True`) rifiuta qualsiasi richiesta che il manager non è in grado di autorizzare positivamente.

Quando utilizzarlo: API solo aziendale: richiedere HTTPS per ogni richiesta e limitare le chiavi all'intervallo CIDR di uscita dell'ufficio, in modo che anche una chiave compromessa non possa essere abusata dall'esterno della rete aziendale.

Proprietà	Type	Predefinito	Descrizione
HeaderName	string	X-API-Key	Nome dell'intestazione HTTP utilizzata per leggere la chiave in arrivo.
QueryParamName	string	api_key	Parametro query-string utilizzato come fallback quando l'intestazione è assente.
Require-HTTPS	Boolean	False	Quando True, rifiuta ogni richiesta non HTTPS indipendentemente dalla validità della chiave.
IPAllowlist	TStringList	vuoto	Quando non è vuoto, solo le richieste il cui IP compare nell'elenco sono autorizzate.
FailClosed	Boolean	True	Quando True, qualsiasi richiesta che il manager non riesce ad autorizzare positivamente viene rifiutata.

```
// Corporate API: HTTPS-only, allow only office egress IPs, fail closed
apiKeyManager.Validation.HeaderName = "X-API-Key";
apiKeyManager.Validation.QueryParamName = "api_key";
apiKeyManager.Validation.RequireHTTPS = true;
apiKeyManager.Validation.FailClosed = true;
apiKeyManager.Validation.IPAllowlist.Add("203.0.113.10");
apiKeyManager.Validation.IPAllowlist.Add("203.0.113.11");
```

Statistiche

Cosa fa: Oggetto contatore di sola lettura che espone statistiche in tempo reale sull'archivio delle chiavi. Tutti i contatori vengono aggiornati atomicamente all'interno della sezione critica interna del componente.

Quando utilizzarlo: Dashboard di amministrazione in tempo reale. Eseguire il polling di Stats da un `TTimer` e visualizzare i valori correnti di `ActiveKeys` / `RevokedKeys` / `TotalValidations` / `TotalValidationFailures` in modo che gli operatori possano monitorare la pressione di autenticazione in tempo reale.

Proprietà	Type	Descrizione
TotalKeys	Int64	Numero totale di chiavi mai emesse da questa istanza del manager.
ActiveKeys	Int64	Numero di chiavi attualmente nello stato <code>kksActive</code> .

RevokedKeys	Int64	Numero di chiavi attualmente in stato kksRevoked.
ExpiredKeys	Int64	Numero di chiavi attualmente nello stato kksExpired.
TotalValidations	Int64	Numero totale di chiamate a ValidateKey / IsRequestAuthorized.
TotalValidationFailures	Int64	Numero totale di chiamate di validazione che hanno rifiutato la chiave.
LastIssuedAt	TDa- teTi- me	Timestamp della chiamata IssueKey più recente.

```
// Poll from a Timer and update a dashboard
private void StatsTimer_Tick(object sender, EventArgs e)
{
    labelActive.Text = $"Active: {apiKeyManager.Stats.ActiveKeys}";
    labelRevoked.Text = $"Revoked: {apiKeyManager.Stats.RevokedKeys}";
    labelValidations.Text = $"Validations: {apiKeyManager.Stats.TotalValidations}";
    labelFailures.Text = $"Failures: {apiKeyManager.Stats.TotalValidationFailures}";
}
```

Eventi

OnKeyIssued: Generato quando viene emessa una nuova chiave. Fornisce il proprietario, la chiave in testo normale (ultima possibilità di osservarla) e l'array degli ambiti. Consegnare il testo normale al cliente da questo gestore.

OnKeyValidated: Attivato ogni volta che ValidateKey viene completato. Fornisce la chiave, un flag booleano *aValid* e una stringa di motivo leggibile dalla macchina.

OnKeyRevoked: Generato quando una chiave è stata revocata. Fornisce la chiave e il motivo della revoca.

OnKeyRotated: Attivato quando una chiave è stata ruotata. Fornisce la vecchia chiave e la nuova chiave in modo da poter notificare il cliente.

OnKeyExpired: Generato dalla scansione di scadenza in background quando è trascorso l'ExpiresAt di una chiave (o NotifyBeforeExpirySec secondi prima).

OnValidation: Hook di decisione finale generato durante ValidateKey. Impostare *Allow* := False e fornire un *Reason* per rifiutare una chiave altrimenti valida (regole di business personalizzate).

OnAuditEvent: Generato per ogni voce del registro di audit. Fornisce il record TsgcAPIKeyAuditEntry (Timestamp, KeyPrefix, Action, IP, Details) — ideale per l'inoltro a un SIEM esterno.

Metodi pubblici

Metodo	Descrizione

IssueKey(Owner, Scopes, ExpiresInSec)	Genera, esegue l'hash e archivia una nuova chiave; restituisce il testo in chiaro (unico momento in cui può essere visualizzato). Passare ExpiresInSec = 0 per ereditare Expiration.DefaultTTLSec.
ValidateKey(Key, RequiredScope, IP)	Convalida una chiave grezza. Facoltativamente, impone un ambito richiesto e registra l'IP del richiedente. Restituisce True in caso di successo.
RevokeKey(Key, Reason)	Contrassegna una chiave come revocata; le successive chiamate a ValidateKey restituiscono False.
RotateKey(OldKey, out NewKey)	Emette una nuova chiave per lo stesso proprietario e ambiti; la vecchia chiave viene contrassegnata come kksRotated e rispettata durante GracePeriodSec.
RenewKey(Key, NewExpiresInSec)	Estende la scadenza della chiave del numero di secondi specificato a partire da ora.
GetKeyInfo(Key)	Restituisce un record TsgcAPIKeyInfo con proprietario, scope, stato, timestamp e conteggio di utilizzo.
GetKeysByOwner(Owner)	Restituisce gli identificatori delle chiavi con hash associati al proprietario specificato.
ListScopes(Key)	Restituisce gli scope associati a una chiave.
IsExpired(Key)	Restituisce True se la chiave ha superato la sua scadenza.
IsRevoked(Key)	Restituisce True se la chiave è revocata.
KeyExists(Key)	Restituisce True se la chiave è nota (attiva, revocata o scaduta).
GrantScope(Key, Scope)	Aggiunge uno scope a una chiave esistente.
RevokeScope(Key, Scope)	Rimuove un ambito da una chiave esistente.
HasScope(Key, Scope)	Restituisce True se la chiave contiene lo scope specificato.
GetAuditLog(Key, MaxEntries)	Restituisce le voci di audit; quando Key è vuota, restituisce le voci per tutte le chiavi.
ClearAuditLog(Key)	Cancella il log di audit per una chiave specifica o (se vuoto) per tutte le chiavi.
SaveToFile(FileName)	Salva le chiavi e il registro di audit nel file specificato (ricorre a Storage.FileName se vuoto).
LoadFromFile(FileName)	Ripristina chiavi e registro di controllo dal file.
ExportKeys(Stream)	Esportazione basata su stream di chiavi e log di audit.
ImportKeys(Stream)	Importazione basata su stream di chiavi e registro di audit.
PurgeExpired	Rimuove tutte le chiavi il cui ExpiresAt è nel passato.

Conteggio	Restituisce il numero di chiavi attualmente memorizzate.
IsRequestAuthorized(Headers, QueryString, IP, RequiredScope)	Verifica di autorizzazione una-tantum: estrae la chiave dalle intestazioni o dalla query string, la convalida e applica facoltativamente uno scope.
ExtractKeyFromHeaders(Headers)	Analizza il valore Validation.HeaderName da un blocco di intestazione HTTP grezzo.
ExtractKeyFromQuery(QueryString)	Analizza il valore Validation.QueryParamName da una stringa di query grezza.
IsConnectionAllowed(IP)	Hook del server: restituisce False quando FailClosed è True, IPAllowlist non è vuota e l'IP non è nell'elenco. Viene chiamato automaticamente.
IsMessageAllowed(IP, Message)	Hook del server chiamato automaticamente per ogni messaggio in entrata. Per impostazione predefinita, non esegue operazioni a livello IP.
RegisterConnection(IP)	Registra una nuova connessione. Chiamato automaticamente dal server.
UnregisterConnection(IP)	Rilascia il tracking per una connessione. Viene chiamato automaticamente alla disconnessione.

Integrazione Automatica

Quando assegnato alla proprietà `APIKeyManager` di un server, il componente automaticamente:

- **Alla nuova connessione:** Il server chiama `IsConnectionAllowed(IP)` che applica la `IPAllowlist` opzionale quando `FailClosed` è `True` (altrimenti è un'operazione nulla). `RegisterConnection(IP)` viene invocato per tenere traccia della connessione attiva.
- **Al ricevimento del messaggio:** Il server chiama `IsMessageAllowed(IP, Message)`, anch'esso un no-op a livello IP a meno che la allowlist non rifiuti l'IP. Questi hook *non* ispezionano le intestazioni HTTP, quindi l'autenticazione tramite chiave API non viene eseguita qui.
- **Autorizzazione della richiesta (il cablaggio è a carico dell'utente):** Chiamare `IsRequestAuthorized(Connection.HeadersRequest.Text, Connection.URL, Connection.IP, RequiredScope)` dall'evento `OnConnect` del server. Questo è il punto in cui sono disponibili le intestazioni HTTP raw, quindi è qui che l'intestazione `X-API-Key` viene estratta, validata e opzionalmente verificata per l'ambito. Disconnette la connessione quando la funzione restituisce `False`.
- **Lato client:** Impostare l'intestazione `X-API-Key` (o il `HeaderName` configurato) dall'evento `OnHandshake` / `OnBeforeConnect` del client in modo che il server possa leggerla durante l'upgrade HTTP.
- **Alla disconnessione:** `UnregisterConnection(IP)` viene chiamato per rilasciare il tracciamento interno.

Esempio completo

```
// Enable and configure the API key manager
server.APIKeyManager.Enabled = true;

// Branded keys
server.APIKeyManager.Generation.KeyPrefix = "sgc_live_";
server.APIKeyManager.Generation.KeyLength = 32;
```

```
// Persist to an encrypted file that survives restarts
server.APIKeyManager.Storage.StorageType = StorageType.File;
server.APIKeyManager.Storage.FileName = "apikeys.dat";
server.APIKeyManager.Storage.EncryptAtRest = true;
server.APIKeyManager.Storage.EncryptionKey = "master-secret";
server.APIKeyManager.Storage.AutoSaveSeconds = 60;

// SHA-256 with a static salt
server.APIKeyManager.Hashing.Algorithm = HashAlgorithm.SHA256;
server.APIKeyManager.Hashing.Salt = "my-salt";

// Scope catalog
server.APIKeyManager.Scopes.Enabled = true;
var scopeRead = server.APIKeyManager.Scopes.Scopes.Add();
scopeRead.Name = "read:orders";
scopeRead.Description = "Read orders";
var scopeWrite = server.APIKeyManager.Scopes.Scopes.Add();
scopeWrite.Name = "write:orders";
scopeWrite.Description = "Create orders";

// 90-day default TTL and 24-hour rotation grace period
server.APIKeyManager.Expiration.DefaultTTLSec = 90 * 86400;
server.APIKeyManager.Rotation.Enabled = true;
server.APIKeyManager.Rotation.GracePeriodSec = 86400;

// Full audit log, 12-month retention
server.APIKeyManager.Audit.Enabled = true;
server.APIKeyManager.Audit.LogFile = "apikeys-audit.log";
server.APIKeyManager.Audit.IncludeIP = true;
server.APIKeyManager.Audit.RetentionDays = 365;

// Validation: header-based, fail closed
server.APIKeyManager.Validation.HeaderName = "X-API-Key";
server.APIKeyManager.Validation.QueryParamName = "api_key";
server.APIKeyManager.Validation.FailClosed = true;

// Load previously saved keys and start the server
server.APIKeyManager.LoadFromFile("apikeys.dat");
server.Active = true;

// Issue a new key for a customer (plaintext returned only here)
string apiKey = server.APIKeyManager.IssueKey("customer-123",
    new[] { "read:orders", "write:orders" }, 30 * 86400);
Console.WriteLine("Deliver to customer: " + apiKey);

// Authorize every incoming connection in OnConnect
server.OnConnect += (sender, e) =>
{
    if (!server.APIKeyManager.IsRequestAuthorized(
        e.Connection.HeadersRequest.Text,
        e.Connection.URL, e.Connection.IP, "read:orders"))
        e.Connection.Disconnect();
};

// Rotate a leaked key (old one valid 24h during grace)
string newKey;
server.APIKeyManager.RotateKey(apiKey, out newKey);

// Revoke a key on subscription cancellation
server.APIKeyManager.RevokeKey(newKey, "customer cancelled subscription");
```

Sicurezza dei thread

Il gestore delle chiavi API è completamente thread-safe. Tutti i metodi pubblici utilizzano sezioni critiche interne e liste thread-safe per proteggere l'accesso concorrente all'archivio chiavi, all'indice degli scope, all'indice dei proprietari e al log di audit. Un singolo componente può essere condiviso da più istanze server e accessibile da qualsiasi thread (gestori di eventi server, thread timer, worker in background) senza sincronizzazione esterna.

Riferimento

- Proprietà — 10 proprietà disponibili sul componente.

- Methods — 26 metodi pubblici.
- Eventi — 7 eventi gestibili.

TsgcWebSocketLoadBalancerServer

Componente *TsgcWebSocketLoadBalancerServer*.

Introduzione

Il componente **TsgcWebSocketLoadBalancerServer** consente di bilanciare il carico dei protocolli **WebSocket** e **HTTP**. Per il protocollo WebSocket, distribuisce i messaggi su un gruppo di server e distribuisce le connessioni client utilizzando una sequenza casuale o l'algoritmo delle connessioni minime.

Il Load Balancer Server eredita tutti i metodi e le proprietà da [TsgcWebSocketHTTPServer](#).

Configurazione Load Balancer

Il server Load Balancer è un discendente di [TsgcWebSocketHTTPServer](#); consultare la documentazione relativa a [TsgcWebSocketHTTPServer](#) per sapere come configurarlo.

Inoltre, il Load Balancer dispone della proprietà **LoadBalancer**, che contiene le seguenti proprietà:

- **LoadBalancing**: configurare qui come distribuire le connessioni
 - **IbRandom**: (predefinito) ogni volta che un nuovo client richiede una connessione, viene restituito un server casuale.
 - **IbConnections**: ogni volta che un nuovo client richiede una connessione, viene restituito il server con il minor numero di client connessi.
- **Protocols**: configura quali protocolli sono abilitati
 - **WebSocket**: se true, le connessioni websocket saranno gestite dal Load Balancer Server.
 - **HTTP**: se true, le connessioni http saranno gestite dal Load Balancer Server.

Configurazione del server di backup

I server di backup (i server dietro il load balancer) possono essere un [TsgcWebSocketServer](#), un [TsgcWebSocketHTTPServer](#) o un [Dataspap Server](#).

Quei server hanno una proprietà chiamata **LoadBalancer** dove può configurare la connessione tra il LoadBalancer Server e i Backup Server.

- **Enabled**: impostare a true se si desidera utilizzarlo come server di backup.
- **Host**: l'host in cui si trova il LoadBalancer.
- **Port**: la porta di ascolto del LoadBalancer.
- **Guid**: ID univoco che identifica questo server.
- **Binding**: gli indirizzi pubblici a cui verranno inoltrate le connessioni. Esempio: se il server WebSocket di backup è in ascolto sulla porta 8000 e l'indirizzo IP è 1.1.1.1, utilizzare: ws://1.1.1.1:8000;
- **AutoRegisterBindings**: se abilitato, il Server LoadBalancer utilizzerà la proprietà Bindings del server di backup per configurare i binding pubblici.
- **AutoRestart**: in secondi, se maggiore di zero, il client del bilanciamento del carico del server di backup abiliterà un watchdog interno che ogni x secondi verificherà se la connessione è attiva; se è chiusa, tenterà di riconnettersi.

Riferimento

- Proprietà — 18 proprietà disponibili nel componente.
- Metodi — 9 metodi pubblici.
- Eventi — 22 eventi gestibili.

TsgcWebSocketProxyServer

TsgcWebSocketProxyServer implementa un componente Server WebSocket che rimane in ascolto delle connessioni WebSocket dei client e reindirizza le connessioni dati a un normale server TCP/IP.

Introduzione

TsgcWebSocketProxyServer implementa un componente server WebSocket che ascolta le connessioni WebSocket dei client e inoltra le connessioni dati a un normale server TCP/IP. Ciò è particolarmente utile per le connessioni browser perché consente a un browser di connettersi virtualmente a qualsiasi server.

Riferimento

- Proprietà — 19 proprietà disponibili sul componente.
- Metodi — 7 metodi pubblici.
- Eventi — 22 eventi che è possibile gestire.

TsgclWWebSocketClient

TsgclWWebSocketClient implementa un componente WebSocket IntraWeb che può connettersi a un server WebSocket.

Introduzione

TsgclWWebSocketClient implementa un componente WebSocket IntraWeb che può connettersi a un server WebSocket. Seguire i passaggi seguenti per configurare questo componente:

1. Inserire un componente TsgclWWebSocketClient nel form
2. Imposti Host e Port (l'impostazione predefinita è 80) per connettersi a un server WebSocket disponibile. Può impostare la proprietà URL e Host, Port, Parameters... verranno aggiornati dall'URL. Esempio: wss://127.0.0.1:8080/ws/ produrrà:

3. Può selezionare se desidera TLS (connessione sicura) o no, per impostazione predefinita non è attivato.
4. Set Transports allowed.

WebSockets: utilizzerà l'implementazione WebSocket standard

Emulation: se il browser non supporta i WebSocket, utilizzerà una connessione callback AJAX in loop

6. Create an Async Procedure and set property Active := True

Methods

Open: Opens a WebSocket Connection.

Close: Chiude una connessione WebSocket.

WriteData: invia un messaggio al server WebSocket.

Properties

Connected: is a read-only variable and returns True if the connection is Active, otherwise returns False.

JSOpen: here you can include JavaScript Code on the client side when a connection is opened.

JSClose: qui è possibile includere codice JavaScript lato client quando una connessione viene chiusa.

JSMessage: here you can include JavaScript Code on the client side when clients receive a message from the server. You can get Message String, using Javascript variable "text".

JSError: qui è possibile includere codice JavaScript lato client quando viene generato un errore. È possibile ottenere il messaggio di errore utilizzando la variabile JavaScript "text".

Riferimento

- Proprietà — 11 proprietà disponibili sul componente.
- Methods — 4 metodi pubblici.
- Eventi — 5 eventi che è possibile gestire.

TsgcWSConnection

TsgcWSConnection è un wrapper per le connessioni WebSocket client. È possibile accedere a questo oggetto negli eventi Server o Client.

Metodi

WriteData: invia un messaggio al client.

Close: invia un messaggio di chiusura all'altro peer. È possibile specificare opzionalmente un "CloseCode". Per impostazione predefinita, il valore inviato è il codice di chiusura NORMAL. Se si invia un codice di chiusura negativo, il motivo della chiusura non verrà inviato.

Disconnect: chiude la connessione del client dal lato server. È possibile specificare facoltativamente un "CloseCode".

Ping: invia un ping al client.

AddTCPEndOfFrame: se la connessione è TCP semplice, consente di impostare quale byte/i definisce la fine del messaggio. Il messaggio viene memorizzato nel buffer finché non viene ricevuto completamente.

Subscribed: restituisce se la connessione è iscritta a un canale personalizzato.

Subscribe: sottoscrive questa connessione a un canale. In seguito è possibile trasmettere un messaggio dal componente server a tutte le connessioni sottoscritte a questo canale.

UnSubscribe: annulla l'iscrizione di questa connessione a un canale.

Proprietà

Protocol: restituisce il sotto-protocollo utilizzato su questa connessione.

IP: restituisce l'indirizzo IP del Peer.

Port: restituisce la porta del peer.

LocalIP: restituisce l'indirizzo IP dell'Host.

LocalPort: restituisce la porta dell'Host.

URL: restituisce l'URL richiesto dal client.

Guid: restituisce l'ID di connessione.

HeadersRequest: restituisce un elenco delle intestazioni ricevute nella Request.

HeadersResponse: restituisce un elenco degli Header inviati come risposta.

RecBytes: numero di byte ricevuti.

SendBytes: numero di byte inviati.

Transport: restituisce il tipo di trasporto della connessione:

trpRFC6455: una connessione WebSocket normale.

trpHixie76: una connessione WebSocket che utilizza la bozza delle specifiche WebSocket.

trpFlash: una connessione WebSocket che utilizza Flash come FallBack.

trpSSE: una connessione Server-Sent Events.

trpTCP: connessione TCP semplice.

TCPEndOfFrameScanBuffer: consente di definire quale metodo utilizzare per trovare la fine del messaggio (se si utilizza trpTCP come trasporto).

eofScanNone: ogni volta che arriva un nuovo pacchetto, viene chiamato l'evento OnBinary.

eofScanLatestBytes: se gli ultimi byte sono uguali ai byte aggiunti con il metodo AddTCPEndOfFrame, viene chiamato il messaggio OnBinary, altrimenti questo pacchetto viene memorizzato nel buffer

eofScanAllBytes: cerca nell'intero pacchetto i byte uguali ai byte aggiunti con il metodo AddTCPEndOfFrame. Se trovati, viene chiamato l'evento OnBinary, altrimenti il pacchetto viene bufferizzato

Data: oggetto dati della sessione utente; qui è possibile passare un oggetto e accedervi ogni volta che è necessario, ad esempio: è possibile passare una connessione a un database, proprietà della sessione utente...

Protocolli

Con i WebSocket è possibile implementare sottoprotocolli, consentendo di creare comunicazioni personalizzate. **Ad esempio**, può implementare un sottoprotocollo sopra il protocollo WebSocket per comunicare con un'applicazione personalizzata utilizzando messaggi JSON, e può implementare un altro sottoprotocollo utilizzando messaggi XML.

Quando una connessione viene aperta sul lato Server, verrà convalidato se il subprotocollo inviato dal client è supportato dal server; in caso contrario, la connessione verrà chiusa. Un server può implementare diversi subprotocolli, ma su una singola connessione ne può essere utilizzato solo uno.

I sottoprotocolli sono molto utili per creare applicazioni personalizzate e garantire che tutti i client supportino la stessa interfaccia di comunicazione.

Sebbene il nome del protocollo sia arbitrario, si consiglia di utilizzare nomi univoci come "dataset.esegce.com"

Con il pacchetto `sgcWebSockets` è possibile costruire i propri protocolli e utilizzare i sottoprotocolli integrati forniti:

- 1. Protocollo MQTT:** MQTT è un protocollo di trasporto messaggi publish/subscribe client-server. È leggero, aperto, semplice e progettato per essere facile da implementare.
- 2. Protocol AppRTC:** è un'applicazione demo webrtc sviluppata da Google e Mozilla; consente a entrambi i browser di "comunicare" tra loro utilizzando l'API WebRTC.
- 3. Protocol WebRTC:** progetto open source volto ad abilitare il web con capacità di comunicazione in tempo reale (RTC).
- 4. File di Protocollo:** implementato tramite messaggi binari, fornisce supporto per l'invio di file: dimensione del pacchetto, autorizzazione, QoS, conferma dei messaggi e altro.
- 5. Protocol SGC:** implementato usando messaggi JSON-RPC 2.0, fornisce i seguenti pattern: RPC, PubSub, Transactional Messages, Messages Acknowledgment e altro.
- 7. Protocol Presence:** consente di sapere chi è iscritto a un canale, ad esempio: stanze di chat, collaboratori su un documento, persone che visualizzano la stessa pagina web, avversari in un gioco...
- 8. Protocollo WAMP 1.0:** sottoprotocollo WebSocket aperto che fornisce due pattern di messaggistica asincroni: RPC e PubSub.
- 9. Protocol WAMP 2.0:** sottoprotocollo WebSocket aperto che fornisce due pattern di messaggistica asincrona: RPC e PubSub.
- 10. Protocollo STOMP:** STOMP è il Simple (or Streaming) Text Orientated Messaging Protocol. STOMP fornisce un formato wire interoperabile in modo che i client STOMP possano comunicare con qualsiasi message broker STOMP, garantendo un'interoperabilità di messaggistica semplice e diffusa tra molti linguaggi, piattaforme e broker.
 - 10.1 STOMP for RabbitMQ:** client per il broker RabbitMQ.
 - 10.2 STOMP per ActiveMQ:** client per il broker ActiveMQ.
- 11. Protocol AMQP:** Advanced Message Queuing Protocol (AMQP 0.9.1) è stato creato come protocollo standard aperto che consente l'interoperabilità dei messaggi tra sistemi, indipendentemente dal fornitore del broker di messaggi o dalla piattaforma utilizzata.
- 12. Protocol AMQP1:** Advanced Message Queuing Protocol (AMQP 1.0.0) è uno standard aperto creato per consentire l'interoperabilità di messaggistica tra sistemi, indipendentemente dal fornitore o dalla piattaforma del broker di messaggi utilizzata.
- 13. Protocol E2EE:** I messaggi inviati tramite una connessione WebSocket sono cifrati end-to-end a livello applicativo, così solo i client comunicanti possono leggerli, anche se il server WebSocket li inoltra.

I protocolli possono essere registrati a **runtime**: è sufficiente chiamare il metodo **RegisterProtocol** e passare il componente protocollo come parametro.

Riferimento JavaScript

Qui può ottenere [maggiori informazioni](#) sulla libreria JavaScript comune usata in sgcWebSockets.

Protocolli Javascript

Di default sgcWebSockets utilizza il file **sgcWebSocket.js** per Javascript.

Qui è possibile trovare i metodi disponibili; è necessario sostituire le variabili `{%host%}` e `{%port%}` secondo le proprie esigenze. Esempio: se il server sgcWebSocket è configurato per ascoltare sulla porta 80 del sito `www.example.com`, è necessario configurare l'accesso al file `sgcWebSocket.js` come segue:

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
```

Apri Connessione

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  var socket = new sgcWebSocket('ws://{%host%}:{%port%}');
</script>
```

sgcWebSocket ha 3 parametri, solo il primo è obbligatorio:

```
sgcWebSocket(url, protocol, transport)
```

- **URL:** posizione del server WebSocket, può utilizzare "ws:" per le normali connessioni WebSocket e "wss:" per le connessioni WebSocket sicure.


```
sgcWebSocket('ws://127.0.0.1')
```

```
sgcWebSocket('wss://127.0.0.1')
```
- **Protocol:** se il server accetta uno o più protocolli, è possibile definire quale protocollo si desidera utilizzare.


```
sgcWebSocket('ws://127.0.0.1', 'esegece.com')
```
- **Transport:** per impostazione predefinita, tenta prima di connettersi tramite connessione WebSocket e, se non supportata dal browser, utilizza Server Sent Events come trasporto.

Utilizzi WebSocket se implementato, in caso contrario, utilizzi Server Sent Events:

```
sgcWebSocket('ws://127.0.0.1')
```

Utilizzare solo WebSocket come trasporto:

```
sgcWebSocket('ws://127.0.0.1', '', ['websocket'])
```

Usa solo Server Sent Events come trasporto:

```
sgcWebSocket('ws://127.0.0.1', '', ['sse'])
```

Aprire la Connessione Con Autenticazione

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  var socket = new sgcWebSocket({ "host": "ws://{%host%}:{%port%}", "user": "admin", "password": "1234" });
</script>
```


Invia messaggio

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  var socket = new sgcWebSocket('ws://{%host%}:{%port%}');
  socket.send('Hello sgcWebSockets!');
</script>
```

Mostra avviso con messaggio ricevuto

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  var socket = new sgcWebSocket('ws://{%host%}:{%port%}');
  socket.on('message', function(event)
  {
    alert(event.message);
  }
</script>
```

Messaggio Binario Ricevuto

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  var socket = new sgcWebSocket('ws://{%host%}:{%port%}');
  socket.on('stream', function(event)
  {
    document.getElementById('image').src = URL.createObjectURL(event.stream);
    event.stream = "";
  }
</script>
```

Messaggio binario ricevuto (intestazione + immagine)

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  var socket = new sgcWebSocket('ws://{%host%}:{%port%}');
  socket.on('stream', function(event)
  {
    sgcWSStreamRead(evt.stream, function(header, stream) {
      document.getElementById('text').innerHTML = header;
      document.getElementById('image').src = URL.createObjectURL(event.stream);
      event.stream = "";
    })
  }
</script>
```

Mostra avviso negli eventi OnConnect, OnDisconnect e OnError

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  var socket = new sgcWebSocket('ws://{%host%}:{%port%}');
```

```
socket.on('open', function(event)
{
    alert('sgcWebSocket Open!');
});
socket.on('close', function(event)
{
    alert('sgcWebSocket Closed!');
});
socket.on('error', function(event)
{
    alert('sgcWebSocket Error: ' + event.message);
});
</script>
```

Chiudi connessione

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
    socket.close();
</script>
```

Ottieni lo stato della connessione

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
    socket.state();
</script>
```

Protocollo MQTT

MQTT è un protocollo di trasporto messaggi client-server di tipo publish/subscribe. È leggero, aperto, semplice e progettato per essere facile da implementare. Queste caratteristiche lo rendono ideale per l'uso in molte situazioni, inclusi ambienti con risorse limitate come la comunicazione Machine to Machine (M2M) e l'Internet of Things (IoT), dove è richiesta un'impronta di codice ridotta e/o la larghezza di banda di rete è scarsa.

Il protocollo viene eseguito su TCP/IP, o su altri protocolli di rete che forniscono connessioni ordinate, senza perdita di dati e bidirezionali. Le sue caratteristiche includono:

- Utilizzo del modello di messaggistica publish/subscribe che fornisce distribuzione dei messaggi uno-a-molti e disaccoppiamento delle applicazioni.
- Un trasporto di messaggistica indipendente dal contenuto del payload.
- Tre qualità di servizio per la consegna dei messaggi:
 - "Al massimo una volta", dove i messaggi vengono consegnati in base ai migliori sforzi dell'ambiente operativo. Può verificarsi la perdita di messaggi. Questo livello potrebbe essere utilizzato, ad esempio, con dati di sensori ambientali dove non importa se una singola lettura viene persa poiché la successiva verrà pubblicata poco dopo.
 - "Almeno una volta", dove i messaggi sono garantiti ma possono verificarsi duplicati.
 - "Exactly once", dove si garantisce che i messaggi arrivino esattamente una volta. Questo livello potrebbe essere utilizzato, ad esempio, con sistemi di fatturazione dove messaggi duplicati o persi potrebbero portare all'applicazione di addebiti errati.
- Un overhead di trasporto ridotto e scambi di protocollo minimizzati per ridurre il traffico di rete.
- Un meccanismo per notificare le parti interessate quando si verifica una disconnessione anomala.

Funzionalità

- Supporta le versioni MQTT **3.1.1** e **5.0**.
- **Publish/subscribe** pattern di messaggi per la distribuzione di messaggi uno-a-molti e il disaccoppiamento delle applicazioni.
- **Conferma** dei messaggi inviati.
- Implementa **QoS** (Quality of Service) per la consegna dei messaggi (tutti i livelli: Al più una volta, Almeno una volta ed Esattamente una volta)
- **Last Will Testament**.
- **Sicure** connessioni.
- **HeartBeat** e **Watchdog**.
- **Autenticazione** al server.

Componenti

[TsgcWSPClient_MQTT](#): Componente Client MQTT.

Usi più comuni

- **Connessione**
 - [Connessione MQTT del client](#)
 - [Connetti Server MQTT Mosquitto](#)
 - [Sessioni MQTT del client](#)
 - [Versione MQTT Client](#)
- **Pubblica & Sottoscrivi**
 - [MQTT Publish Subscribe](#)
 - [MQTT Argomenti](#)

- [MQTT Subscribe](#)
- [MQTT Publish Message](#)
- [MQTT Receive Messages](#)
- [MQTT Pubblica e attendi risposta](#)
- **Altro**
 - [MQTT Clear Retained Messages](#)

TsgcWSPClient_MQTT

Il componente MQTT fornisce un'implementazione client MQTT leggera e completa con supporto per le versioni 3.

Introduzione

Il componente MQTT fornisce un'implementazione di client MQTT leggera e completa, con supporto per le versioni 3.1.1 e 5.0. Il componente supporta connessioni in chiaro e sicure sia su TCP standard sia su WebSocket.

La connessione a un server MQTT è semplice: è sufficiente inserire questo componente nel form e selezionare un componente [TsgcWebSocketClient](#) tramite la proprietà Client. Impostare host e porta in TsgcWebSocketClient e impostare Active := True per connettersi.

MQTT v5.0 non è retrocompatibile (come la v3.1.1). Ovviamente vengono introdotte troppe novità, quindi le implementazioni esistenti devono essere riviste.

Secondo la specifica, MQTT v5.0 aggiunge un numero significativo di nuove funzionalità a MQTT mantenendo gran parte del core invariato.

- La funzionalità del flag Clean Session è suddivisa in 2 proprietà per consentire un controllo più fine sui dati di stato della sessione: il parametro CleanStart e il nuovo SessionExpInterval.
- Disconnessione server: consente l'invio di DISCONNECT da parte del Server per indicare il motivo della chiusura della connessione.
- Tutti i pacchetti di risposta (CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK, DISCONNECT) ora contengono un reason code e una reason string che descrivono perché le operazioni sono riuscite o fallite.
- Autenticazione avanzata: fornisce un meccanismo per abilitare l'autenticazione in stile challenge/response, inclusa l'autenticazione reciproca. Ciò consente di utilizzare l'autenticazione in stile SASL se supportata sia dal client che dal server, e include la possibilità per un client di ri-autenticarsi all'interno di una connessione.
- Il pattern Request / Response viene formalizzato dall'aggiunta del ResponseTopic.
- Shared Subscriptions: aggiunge il supporto per le sottoscrizioni condivise consentendo consumer con bilanciamento del carico di una sottoscrizione.
- Gli alias di topic possono essere inviati sia dal client che dal server per riferirsi ai filtri di topic tramite identificatori numerici più brevi al fine di risparmiare larghezza di banda.
- I server possono comunicare quali funzionalità supportano in ConnectionProperties.
- Riferimento al server: consente al server di specificare un server alternativo da utilizzare su CONNACK o DISCONNECT. Può essere utilizzato come reindirizzamento o per il provisioning.
- Altro: scadenza dei messaggi, Receive Maximum e Maximum Packet Size, e un intervallo Will Delay sono tutti supportati.

Riferimento

- Proprietà — 10 proprietà disponibili sul componente.
- Metodi — 9 metodi pubblici.
- Eventi — 13 eventi gestibili.

TsgcWSPClient_MQTT | Client MQTT Connect

Per connettersi a un server MQTT, occorre prima creare un [TsgcWebSocketClient](#) e un [TsgcWSPClient_MQTT](#). Successivamente è necessario collegare il componente MQTT al client WebSocket.

Utilizzo di base

Connettere al server MQTT Mosquitto utilizzando il protocollo websocket. Iscrivere al topic: "topic1" dopo la connessione.

```
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");
}
```

Identificatore Client

MQTT richiede un **Identificatore Client** per identificare la connessione del client. Il componente imposta automaticamente un **valore casuale**, ma è possibile impostare il proprio Identificatore Client se necessario; per farlo, gestire l'evento **OnBeforeConnect** e impostare il proprio valore nel parametro aClientIdentifier.

```
void OnMQTTBeforeConnect(TsgcWSConnection Connection, ref bool aCleanSession,
    ref string aClientIdentifier)
{
    aClientIdentifier = "your client id";
}
```

Autenticazione

Alcuni server richiedono nome utente e password per **autorizzare le connessioni MQTT**. Utilizzare la proprietà **Authentication** per impostare il nome utente e la password prima di connettersi al server.

```
oMQTT = new TsgcWSPClient_MQTT();
oMQTT.Authentication.Enabled = true;
oMQTT.Authentication.UserName = "your user";
oMQTT.Authentication.Password = "your passwd";
```

TsgcWSPClient_MQTT | Connetti MQTT Mosquitto

Utilizzare le seguenti configurazioni di esempio per connettersi a un server MQTT Mosquitto.

MOSQUITTO MQTT WebSockets

```
oClient = new TsgcWebSocketClient();
oClient.Host = "test.mosquitto.org";
oClient.Port = 8080;
oMQTT = TsgcWSPClient_MQTT.Create(nil);
oMQTT.Client = oClient;
oClient.Active = true;
```

MOSQUITTO MQTT WebSockets TLS

```
oClient = new TsgcWebSocketClient();
oClient.Host = "test.mosquitto.org";
oClient.Port = 8081;
oClient.TLS = true;
oClient.TLSOptions.Version = TwstLSVersions.tls1_2;
oMQTT = TsgcWSPClient_MQTT.Create(nil);
oMQTT.Client = oClient;
oClient.Active = true;
```

MOSQUITTO MQTT Plain TCP

```
oClient = new TsgcWebSocketClient();
oClient.Host = "test.mosquitto.org";
oClient.Port = 1883;
oClient.Specifications.RFC6455 := False;
oMQTT = TsgcWSPClient_MQTT.Create(nil);
oMQTT.Client = oClient;
oClient.Active = true;
```

MOSQUITTO MQTT Plain TCP TLS

```
oClient = new TsgcWebSocketClient();
oClient.Host = "test.mosquitto.org";
oClient.Port = 8083;
oClient.Specifications.RFC6455 := False;
oClient.TLS = true;
oClient.TLSOptions.Version = TwstLSVersions.tls1_2;
oMQTT = TsgcWSPClient_MQTT.Create(nil);
oMQTT.Client = oClient;
oClient.Active = true;
```

TsgcWSPClient_MQTT | Sessioni client MQTT

Clean Start

Nell'evento **OnMQTTBeforeConnect**, c'è un parametro chiamato **aCleanSession**. Se il valore di questo parametro è **True**, significa che il client **vuole avviare una nuova sessione**, quindi se il server ha una sessione memorizzata, deve eliminarla. Pertanto, quando viene generato l'evento **OnMQTTConnect**, il parametro **aSession** sarà **false**. Se il valore di questo parametro è **False** e c'è una sessione associata a questo identificatore client, il server deve riprendere le comunicazioni con il client nello stato della sessione esistente.

Pertanto, se il client subisce una **disconnessione imprevista** e si desidera **ripristinare la sessione** nel punto in cui è avvenuta la disconnessione, in **OnMQTTBeforeConnect** impostare **aCleanSession = True** e **aClientIdentifier = Client ID of Session**.

Sessione

Dopo la connessione avvenuta con successo, verificare l'evento **OnMQTTConnect** e il valore del parametro **Session**.

Session = true, significa che la sessione è stata ripresa.

Session = false, significa che si tratta di una nuova sessione.

```
void OnMQTTBeforeConnect(TsgcWSConnection Connection, ref bool aCleanSession,
    ref string aClientIdentifier)
{
    aCleanSession = false;
    aClientIdentifier = "previous client id";
}

void OnMQTTConnect(TsgcWSConnection Connection, bool Session, int ReasonCode,
    string ReasonName, TsgcWSMQTTCONNACKProperties ConnectProperties);
{
    if (Session == true)
    {
        Console.WriteLine("Session resumed");
    }
    else
    {
        Console.WriteLine("New Session");
    }
}
```


TsgcWSPClient_MQTT | Versione Client MQTT

Attualmente, il Client MQTT supporta le seguenti specifiche:

- **MQTT 3.1.1:** <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>
- **MQTT 5.0:** <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>

È possibile selezionare la versione che utilizzerà il componente MQTT Client tramite la proprietà MQTTVersion.

MQTT 3.1.1: TsgcWSPClient_MQTT.Version = mqtt311

MQTT 5.0: sgcWSPClient_MQTT.Version = mqtt5

TsgcWSPClient_MQTT | MQTT Publish Subscribe

Il pattern publish/subscribe (noto anche come pub/sub) fornisce un'alternativa all'architettura client-server tradizionale. Nel modello client-server, un client comunica direttamente con un endpoint. Il modello pub/sub **disaccoppia il client che invia un messaggio (il publisher) dal client o dai client che ricevono i messaggi (i subscriber)**. I publisher e i subscriber non si contattano mai direttamente. In realtà, non sono nemmeno a conoscenza dell'esistenza dell'altro. **La connessione tra di loro è gestita da un terzo componente (il broker)**. Il compito del broker è filtrare tutti i messaggi in entrata e distribuirli correttamente ai subscriber.

Con **TsgcWSPClient_MQTT** è possibile **pubblicare messaggi** e **sottoscrivere Topic**.

Iscriviti al Topic

Iscriversi al Topic "topic1" dopo una connessione riuscita.

```
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");
}
```

Pubblica messaggio

Pubblica un messaggio a tutti i subscriber di "topic1"

```
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.Publish("topic1", "Hello Subscribers topic1");
}
```

TsgcWSPClient_MQTT | MQTT Topics

Topics

In MQTT, il termine topic si riferisce a una stringa UTF-8 che il broker utilizza per filtrare i messaggi per ogni client connesso. Il topic è composto da uno o più livelli di topic. Ogni livello di topic è separato da una barra obliqua (separatore di livello topic)

myHome / groundfloor / livingroom / temperature

Rispetto a una coda di messaggi, i topic MQTT sono molto leggeri. Il client non deve creare il topic desiderato prima di pubblicarlo o sottoscriverlo. Il broker accetta ogni topic valido senza alcuna inizializzazione preliminare. Si noti che ogni topic deve contenere almeno 1 carattere e che la stringa del topic ammette spazi vuoti. I topic fanno distinzione tra maiuscole e minuscole.

WildCards

Quando un client si sottoscrive a un topic, può sottoscrivere all'esatto topic di un messaggio pubblicato oppure può utilizzare caratteri jolly per sottoscrivere a più topic contemporaneamente. Un carattere jolly può essere utilizzato solo per sottoscrivere ai topic, non per pubblicare un messaggio. Esistono due tipi diversi di caratteri jolly: `_single-level` e `_multi-level`.

Livello Singolo: +

Come suggerisce il nome, un wildcard a livello singolo sostituisce un livello di argomento. Il simbolo più rappresenta un wildcard a livello singolo in un argomento.

myHome / groundfloor / + / temperature

Qualsiasi argomento corrisponde a un argomento con un wildcard a singolo livello se contiene una stringa arbitraria al posto del wildcard. Ad esempio, una sottoscrizione a `_myhome/groundfloor+/temperature` può produrre i seguenti risultati:

```
YES => myHome / groundfloor / livingroom / temperature
YES => myHome / groundfloor / kitchen / temperature
NO => myHome / groundfloor / livingroom / brightness
NO => myHome / firstfloor / livingroom / temperature
NO => myHome / groundfloor / kitchen / fridge / temperature
```

Multi Livello:

La wildcard multilivello copre molti livelli di topic. Il simbolo cancelletto rappresenta la wildcard multilivello nel topic. Affinché il broker possa determinare quali topic corrispondono, la wildcard multilivello deve essere posizionata come ultimo carattere nel topic e preceduta da una barra in avanti.

myHome / groundfloor / #

```
YES => myHome / groundfloor / livingroom / temperature
YES => myHome / groundfloor / kitchen / temperature
YES => myHome / groundfloor / kitchen / brightness
NO => myHome / firstfloor / kitchen / temperature
```

Quando un client si sottoscrive a un topic con un wildcard multi-livello, riceve tutti i messaggi di un topic che inizia con il pattern precedente al carattere wildcard, indipendentemente da quanto sia lungo o profondo il topic. Se si specifica solo il wildcard multi-livello come topic (`_#`), si ricevono tutti i messaggi inviati al broker MQTT.

TsgcWSPClient_MQTT | MQTT Subscribe

È possibile sottoscrivere a un Topic utilizzando il metodo Subscribe di TsgcWSPClient_MQTT. Questo metodo dispone dei seguenti parametri:

Topic: è il nome del topic a cui sottoscrivere.

QoS: uno dei 3 livelli QoS (non tutti i broker supportano tutti e 3 i livelli). Se non specificato, utilizza mtqsAtMostOnce. Per ulteriori informazioni, consultare Livelli QoS.

SubscribeProperties: se MQTT 5.0, sono proprietà aggiuntive relative alle sottoscrizioni.

Subscribe QoS = At Least Once

```
MQTT.Subscribe("topic1", TmqttQoS.mtqsAtLeastOnce);
```

Subscribe MQTT 5.0

```
oProperties = new TsgcWSMQTTSubscribe_Properties();  
oProperties.SubscriptionIdentifier = 1234;  
oProperties.UserProperties = "name=value";  
  
MQTT->Subscribe("topic1", TmqttQoS.mtqsAtMostOnce, oProperties);
```

TsgcWSPClient_MQTT | Pubblicazione Messaggio MQTT

È possibile pubblicare messaggi a tutti i sottoscrittori di un Topic usando il metodo **Publish**, che ha i seguenti parametri:

Topic: è il nome del topic in cui verrà pubblicato il messaggio.

Text: è il testo del messaggio.

QoS: uno dei 3 livelli QoS (non tutti i broker supportano tutti e 3 i livelli). Se non specificato, utilizza `mtqsAtMostOnce`. Per ulteriori informazioni, consultare Livelli QoS.

Retain: se `true`, questo messaggio verrà mantenuto. E ogni volta che un nuovo client si iscrive a questo topic, questo messaggio verrà inviato a quel client.

PublishProperties: se MQTT 5.0, queste sono le proprietà del messaggio.

Pubblicare un messaggio semplice

```
MQTT.Publish("topic1", "Hello Subscribers topic1");
```

QoS di pubblicazione = At Least Once

```
MQTT.Publish("topic1", "Hello Subscribers topic1", TmqttQoS.mtqsAtLeastOnce);
```

Pubblicare un messaggio Retained

```
MQTT.Publish("topic1", "Hello Subscribers topic1", TmqttQoS.mtqsAtMostOnce, true);
```

TsgcWSPClient_MQTT | Ricezione Messaggi MQTT

I messaggi inviati dal server vengono ricevuti nell'evento **OnMQTTPublish**. Questo evento ha i seguenti parametri:

Topic: è il nome del topic associato a questo messaggio.

Text: è il testo del messaggio.

PublishProperties: se MQTT 5.0, queste sono le proprietà del messaggio pubblicato.

Leggi messaggi pubblicati

```
void OnMQTTPublish(TsgcWSConnection Connection, string aTopic, string aText,
    TsgcWSMQTTPublishProperties PublishProperties)
{
    WriteLine("Topic: " + aTopic + ". Message: " + aText);
}
```

TsgcWSPClient_MQTT | Ricezione messaggi MQTT (Estesa)

L'evento **OnMQTTPublishEx** fornisce il payload del messaggio pubblicato in più formati tramite un oggetto **TsgcWSMQTTPublishData**. Questo evento ha i seguenti parametri:

Topic: è il nome del topic associato a questo messaggio.

Data: contiene il payload del messaggio pubblicato. Ha le seguenti proprietà:

- **Value:** il payload come stringa.
 - **Bytes:** il payload non elaborato come TBytes.
 - **Stream:** il payload non elaborato come TMemoryStream.
- PublishProperties:** se MQTT 5.0, queste sono le proprietà del messaggio pubblicato.

Leggere i Messaggi pubblicati (Esteso)

```
void OnMQTTPublishEx(TsgcWSConnection Connection, string aTopic,
    TsgcWSMQTTPublishData aData, TsgcWSMQTTPublishProperties PublishProperties)
{
    // read as string
    WriteLine("Topic: " + aTopic + ". Message: " + aData.Value);
    // read as bytes
    WriteLine("Bytes Length: " + aData.Bytes.Length.ToString());
    // read as stream
    WriteLine("Stream Size: " + aData.Stream.Size.ToString());
}
```

TsgcWSPClient_MQTT | Publish and Wait Response

Il client MQTT consente l'utilizzo di alcuni tipi di livelli QoS, ognuno dei quali funziona in modo diverso per garantire che i messaggi siano stati elaborati come previsto.

Esistono i seguenti livelli QoS:

- **mtqsAtMostOnce:** (per impostazione predefinita) il messaggio viene consegnato secondo il massimo impegno della rete TCP/IP sottostante. Non è prevista una risposta e nessuna semantica di retry è definita nel protocollo. Il messaggio arriva al server una volta o non arriva affatto.
- **mtqsAtLeastOnce:** la ricezione di un messaggio da parte del server è confermata da un messaggio di ACKNOWLEDGMENT. Se si verifica un errore identificato nel collegamento di comunicazione o nel dispositivo mittente, oppure se il messaggio di conferma non viene ricevuto entro un periodo di tempo specificato, il mittente reinvia il messaggio. Il messaggio arriva al server almeno una volta. Un messaggio con livello QoS 1 ha un parametro ID nel messaggio.
- **mtqsExactlyOnce:** dove si garantisce che i messaggi arrivino esattamente una volta. Questo livello potrebbe essere utilizzato, per esempio, con sistemi di fatturazione dove messaggi duplicati o persi potrebbero portare a addebiti errati. Se si verifica un guasto identificato del collegamento di comunicazione o del dispositivo di invio, oppure il messaggio di conferma non viene ricevuto dopo un periodo di tempo specificato, il mittente reinvia il messaggio.

Può gestire gli eventi OnPubAck o OnPubComp per sapere se il messaggio è stato elaborato dal server, oppure può usare il metodo **PublishAndWait** per sapere se il messaggio è stato elaborato dal server.

L'utilizzo di **PublishAndWait** è uguale al normale metodo Publish, ma ora si dispone di un nuovo parametro chiamato Timeout, dove il metodo restituirà false se trascorso un determinato periodo di tempo, non vi è risposta dal server. Per impostazione predefinita questo valore è 10 secondi.

```
if mqtt->PublishAndWait("topic", "text")
{
    MessageBox.Show("Message processed")
}
else
{
    MessageBox.Show("Message error");
}
```


TsgcWSPClient_MQTT | MQTT Clear Retained Messages

Per impostazione predefinita, ogni topic MQTT può avere un messaggio retained. Il meccanismo MQTT standard per eliminare i messaggi retained consiste nell'inviare un messaggio retained con payload vuoto a un topic. Questo rimuoverà il messaggio retained.

```
MQTT.Publish("topic1", "", TmqttQoS.mtqsAtMostOnce, true);
```

Protocol AMQP 0.9.1

L'**Advanced Message Queuing Protocol** (AMQP) è un protocollo standard aperto a livello di applicazione per il middleware orientato ai messaggi. Le caratteristiche distintive di AMQP sono l'orientamento ai messaggi, il queuing, il routing (incluso il point-to-point e il publish-and-subscribe), l'affidabilità e la sicurezza.

AMQP è un protocollo binario a livello applicativo, progettato per supportare in modo efficiente un'ampia varietà di applicazioni di messaggistica e pattern di comunicazione. Fornisce comunicazione orientata ai messaggi con controllo del flusso e garanzie di consegna dei messaggi come: al-più-una-volta (dove ogni messaggio viene consegnato una volta o mai), almeno-una-volta (dove la consegna di ogni messaggio è garantita, ma può avvenire più volte) ed esattamente-una-volta (dove il messaggio arriverà sempre e una sola volta), oltre ad autenticazione e/o cifratura basate su SASL e/o TLS. Presuppone un protocollo di trasporto affidabile sottostante come il Transmission Control Protocol (TCP).

Funzionalità

AMQP può essere utilizzato in qualsiasi situazione in cui vi sia necessità di una consegna dei messaggi di alta qualità e sicura tra client e broker.

AMQP fornisce le seguenti funzionalità:

- Monitoraggio e condivisione degli aggiornamenti.
- Garanzia di una risposta rapida del server alle richieste e di trasmissione di attività dispendiose in termini di tempo per ulteriore elaborazione.
- Distribuire messaggi a più destinatari.
- Connessione client offline per ulteriori recuperi di dati.
- Aumentare l'affidabilità e il funzionamento ottimale delle applicazioni.
- Affidabilità della consegna dei messaggi.
- Consegna dei messaggi ad alta velocità.
- Accettazione dei messaggi.

Componenti

TsgcWSPClient_AMQP: è il componente client che implementa il protocollo **AMQP 0.9.1**.

Usi più comuni

- **Connessione**
 - [Client AMQP Connect](#)
 - [Client AMQP Disconnect](#)
- **Comandi**
 - [Canali AMQP](#)
 - [Exchange AMQP](#)
 - [Code AMQP](#)
 - [Pubblicazione messaggi AMQP](#)
 - [Consumo messaggi AMQP](#) (Asincrono)
 - [Recupero messaggi AMQP](#) (Sincrono)
 - [AMQP QoS](#)
 - [Transazioni AMQP](#)

TsgcWSPClient_AMQP

AMQP 0.

Introduzione

Il client **TsgcWSPClient_AMQP** implementa il protocollo completo **AMQP 0.9.1** seguendo la specifica OASIS. Il client supporta connessioni TCP semplici e WebSocket; sono supportate anche le connessioni TLS (sicure).

Connessione

Il protocollo AMQP 0.9.1 definisce il concetto di canali, che consente di condividere una singola connessione socket con diversi canali virtuali; il client implementa un thread interno che legge i byte ricevuti e distribuisce ogni messaggio al canale corretto (che già opera nel proprio thread). Pertanto, se si esegue una connessione AMQP con 5 canali, il client avvierà 6 thread (5 thread che gestiscono i dati di ogni canale e 1 thread che gestisce i dati della connessione).

Prima di connettersi a un server AMQP, configurare le seguenti proprietà del protocollo AMQP

- **AMQPOptions.Locale:** è il locale del messaggio da utilizzare, è un valore negoziato e può cambiare rispetto ai locale supportati dal server. Il valore predefinito è "en_US".
- **AMQPOptions.MaxChannels:** è il numero massimo di canali che possono essere aperti, è un valore negoziato, quindi può cambiare rispetto alla configurazione del server. Il valore predefinito è 65535.
- **AMQPOptions.MaxFrameSize:** è la dimensione massima in byte del frame AMQP; è un valore negoziato, quindi può variare rispetto alla configurazione del server. Il valore predefinito è 2147483647.
- **AMQPOptions.VirtualHost:** è il nome dell'host virtuale. Il valore predefinito è "/".

L'AMQP HeartBeat può essere configurato anche prima di connettersi al server; è possibile abilitare o disabilitare l'uso degli heartbeat.

- **HeartBeat.Enabled:** impostare su true se il client supporta gli HeartBeat.
- **HeartBeat.Interval:** l'intervallo desiderato in secondi.

Una volta configurato il client AMQP, collegarlo a un [TsgcWebSocketClient](#) e quindi configurare le proprietà di connessione al server per connettersi al Server AMQP.

Impostare il valore della proprietà **Specifications.RFC6455** su false se si utilizza una connessione TCP normale anziché una connessione WebSocket.

```
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;
```

Canali

Una volta connesso il client AMQP, può aprire il primo canale.

```
oAMQP.OpenChannel("channel_name");
```

Exchange

Quando un Channel viene aperto, il client può dichiarare nuovi exchange, verificarne l'esistenza... utilizzare il metodo **DeclareExchange** per dichiarare un nuovo exchange.

```
oAMQP.DeclareExchange("channel_name", "exchange_name");
```

Code

Quando viene aperto un canale, il client può dichiarare nuove code, verificarne l'esistenza... utilizzare il metodo **DeclareQueue** per dichiarare una nuova coda. Le code non vengono fornite di default dal server (a differenza degli exchange), quindi è sempre necessario dichiararne una nuova (a meno che una coda non sia già stata creata da un altro client).

```
oAMQP.DeclareQueue("channel_name", "queue_name");
```

Binding delle code

Una volta configurati Exchange e Queue, potrebbe essere necessario associare le code agli exchange, in modo che gli exchange sappiano quali messaggi devono essere inviati alle code.

I server AMQP collegano automaticamente le code all'exchange "direct" utilizzando il nome della coda come routing key. Questo consente di inviare un messaggio a una coda specifica senza la necessità di dichiarare un binding (basta chiamare il metodo **PublishMessage** e passare l'argomento Exchange come valore vuoto e il nome della coda nell'argomento RoutingKey).

```
oAMQP.BindQueue("channel_name", "queue_name", "exchange_name", "routing_key");
```

Invia Messaggi

Chiamare il metodo **PublishMessage** per pubblicare un nuovo messaggio AMQP. Il metodo consente di pubblicare un messaggio di tipo **String** o **TStream**.

```
oAMQP.PublishMessage("channel_name", "exchange_name", "routing_key", "Hello from sgcWebSockets!!!");
```

Ricezione Messaggi

AMQP consente di ricevere i messaggi in 2 modalità:

- **Richiesta dal Client:** utilizzando il metodo **GetMessage**. Se non ci sono messaggi in coda, verrà chiamato l'evento **OnAMQPBasicGetEmpty**.
- **Inviato dal Server:** utilizzando il metodo **Consume**.

Richiesta dal Client

```
oAMQP.GetMessage("channel_name", "queue_name");

private void OnAMQPGetOk(TObject Sender, string aChannel,
    TsgcAMQPFramePayload_Method_BasicGetOk aGetOk, TsgcAMQPMessageContent aContent)
{
```

```
    DoLog("#AMQP_basic_GetOk: " + aChannel + " " + IntToStr(aGetOk.MessageCount) + " " + aContent.Body.AsString
  }
```

Inviato dal server

```
oAMQP.Consume("channel_name", "queue_name");

private void OnAMQPGetOk(TObject Sender, string aChannel,
  TsgcAMQPFramePayload_Method_BasicGetOk aGetOk, TsgcAMQPMessageContent aContent)
{
  DoLog("#AMQP_basic_GetOk: " + aChannel + " " + IntToStr(aGetOk.MessageCount) + " " + aContent.Body.AsString
}
```

Riferimento

- Proprietà — 6 proprietà disponibili nel componente.
- Metodi — 41 metodi pubblici.
- Eventi — 25 eventi gestibili.

Connessione | Client AMQP Connect

Per connettersi a un server AMQP, è necessario creare prima un [TsgcWebSocketClient](#) e un [TsgcWSPClient_AMQP](#). Successivamente è necessario collegare il componente AMQP al client WebSocket.

Utilizzo di base

Connette al server AMQP senza autenticazione. Definire i valori della proprietà AMQPOptions, l'host virtuale e quindi impostare in TsgcWebSocketClient l'Host e la Porta del server.

Se si utilizza una connessione TCP Plain, impostare la proprietà Specifications.RFC6455 del TsgcWebSocketClient su false.

```
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;
```

Autenticazione

Se il server richiede l'autenticazione, utilizzare l'evento **OnAMQPAuthentication** per selezionare il meccanismo di autenticazione (se richiesto) e impostare l'utente/la password.

```
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;

private void OnAMQPAuthentication(object Sender, TsgcAMQPAuthentications aMechanisms, ref TsgcAMQPAuthentication
    ref string User, ref string Password)
{
    User = "user_value";
    Password = "password_value";
}
```

Connessione | Disconnessione client AMQP

Il client può disconnettere una connessione attiva corrente utilizzando i seguenti metodi:

Invio di un motivo di chiusura

Il client AMQP può informare il server che la connessione verrà chiusa e fornire informazioni sul motivo della chiusura. Utilizzare il metodo `Close` per richiedere la chiusura della connessione al server.

```
oAMQP.Close(541, "Internal Error");
```

Chiusura della connessione socket

Impostare semplicemente la proprietà `Active` di [TsgcWebSocketClient](#) su `False`. È possibile leggere ulteriori informazioni sulla [chiusura delle connessioni](#).

Comandi | Canali AMQP

AMQP è un protocollo multi-canale. I canali forniscono un modo per multiplexare una connessione TCP/IP pesante in più connessioni leggere. Questo rende il protocollo più "firewall friendly" poiché l'utilizzo delle porte è prevedibile. Significa anche che il traffic shaping e altre funzionalità QoS di rete possono essere facilmente adottate.

Ogni canale viene eseguito nel proprio thread; pertanto, ogni volta che viene ricevuto un nuovo messaggio, il client identifica prima il canale e mette il messaggio in coda, che viene elaborata dal thread del canale.

Il ciclo di vita del canale è il seguente:

1. Il client apre un nuovo canale (Open).
2. Il server conferma che il nuovo canale è pronto (Open-Ok).
3. Il client e il server utilizzano il canale come desiderato.
4. Un peer (client o server) chiude il canale (Close).
5. L'altro peer completa l'handshake di chiusura del canale (Close-Ok).

Apri canale

Per creare un nuovo canale, chiamare il metodo **OpenChannel** e passare il nome del canale come argomento. L'evento **OnAMQPChannelOpen** viene generato come conferma inviata dal server che il canale è stato aperto.

```
AMQP.OpenChannel("channel_name");

private void OnAMQPChannelOpen(TObject Sender, const string aChannel)
{
    DoLog("#AMQP_channel_open: " + aChannel);
}
```

È anche possibile effettuare una chiamata sincrona chiamando il metodo **OpenChannelEx**, questo metodo restituisce true se il canale è stato aperto e false se non è arrivata alcuna conferma dal server.

```
if AMQP.OpenChannelEx('channel_name')
{
    DoLog("#AMQP_channel_open channel_name");
}
else
{
    DoLog("#AMQP_channel_open_error");
}
```

Chiudi canale

Per chiudere un canale esistente, chiamare il metodo **CloseChannel** e passare il nome del canale come argomento. L'evento **OnAMQPChannelClose** verrà chiamato quando il client riceve la conferma che il canale è stato chiuso.

Una chiamata sincrona può essere eseguita chiamando il metodo **CloseChannelEx**, questo metodo restituisce true se il canale è stato chiuso e false se non è arrivata alcuna conferma dal server.

Channel Flow

Il controllo di flusso è una procedura di emergenza utilizzata per bloccare il flusso di messaggi da un peer. Funziona allo stesso modo tra client e server ed è implementato dai comandi **EnableChannel** / **DisableChannel**. Il controllo di flusso è l'unico meccanismo in grado di fermare un publisher che produce in eccesso.

Per disabilitare il flusso di un canale, chiamare il metodo **DisableChannel**, l'evento **OnAMQPChannelFlow** verrà chiamato quando il client riceve la conferma che il flusso del canale è stato disabilitato.

Lo stesso vale quando si abilita il flow di un canale, chiami il metodo **EnableChannel**, l'evento **OnAMQPChannelFlow** verrà chiamato quando il client riceve una conferma che il flow del canale è stato abilitato.

Le richieste sincrone sono disponibili tramite le funzioni **EnableChannelEx** e **DisableChannelEx**.

Comandi | Exchange AMQP

La classe `exchange` consente a un'applicazione di gestire gli exchange sul server. Questa classe permette all'applicazione di definire il proprio cablaggio (anziché affidarsi a un'interfaccia di configurazione). Nota: la maggior parte delle applicazioni non necessita di questo livello di sofisticazione, ed è improbabile che il middleware legacy sia in grado di supportare questa semantica.

Il ciclo di vita dell'exchange è:

1. Il client chiede al server di verificare l'esistenza dell'exchange (Declare). Il client può specificare: "crea l'exchange se non esiste", oppure "avvisami ma non crearlo, se non esiste".
2. Il client pubblica i messaggi nell'exchange.
3. Il client può scegliere di eliminare l'exchange (Delete).

Dichiarazione Exchange

Questo metodo crea un nuovo exchange o verifica che un Exchange esista già. Il metodo accetta i seguenti argomenti:

- **ChannelName:** è il nome del canale (deve essere aperto prima di chiamare questo metodo).
- **ExchangeName:** è il nome dell'exchange; non deve superare i 255 caratteri e non deve iniziare con "amq." (eccetto se il parametro `passive` è `true`).
- **ExchangeType:** è il tipo di exchange; tutti i server AMQP supportano gli exchange "direct" e "fanout". Consultare la documentazione del server per conoscere quali tipi di exchange sono supportati.
- **Passive:** se `passive` è `true`, il server verifica solo che l'exchange sia già dichiarato. Se `passive` è `false` e l'exchange non esiste, il server ne creerà uno nuovo.
- **Durable:** se `true`, lo scambio verrà ricreato all'avvio del server. Se `false`, lo scambio verrà eliminato all'arresto del server.
- **AutoDelete:** se `true`, l'exchange verrà eliminato quando tutte le code saranno state scollegate.
- **Internal:** sempre `false`.
- **NoWait:** se `true`, il server non invia una conferma al client.
- **Arguments:** stringa contenente argomenti personalizzati; i valori devono essere passati come stringa json, esempio: `{"x-dead-letter-exchange":"my-dlx"}`.

Per dichiarare un nuovo Exchange, chiamare il metodo **DeclareExchange** passando il nome del canale, il nome dell'exchange e il tipo di exchange come argomenti. L'evento **OnAMQPExchangeDeclare** viene generato come conferma inviata dal server che l'exchange è stato dichiarato.

```
AMQP.DeclareExchange("channel_name", "exchange_name", "direct");

private void OnAMQPExchangeDeclare(TObject Sender, const string aChannel, const string aExchange)
{
    DoLog("#AMQP_exchange_declare: [" + aChannel + "] " + aExchange);
}
```

È possibile effettuare anche una chiamata sincrona richiamando il metodo **DeclareExchangeEx**; questo metodo restituisce `true` se l'Exchange è stato dichiarato e `false` se non è arrivata alcuna conferma dal server.

```
if AMQP.DeclareExchangeEx("channel_name", "exchange_name", "direct")
{
    DoLog("#AMQP_exchange_declare: [" + aChannel + "] " + aExchange);
}
else
{
    DoLog("#AMQP_exchange_declare_error");
}
```

Elimina Exchange

Questo metodo viene utilizzato per eliminare un Exchange esistente. Il metodo ha i seguenti argomenti:

- **ChannelName:** è il nome del canale (deve essere aperto prima di chiamare questo metodo).
- **ExchangeName:** è il nome dell'exchange; non deve superare i 255 caratteri e non deve iniziare con "amq." (eccetto se il parametro `passive` è `true`).
- **IfUnused:** il server elimina l'exchange solo se non vi sono code ad esso associate.
- **NoWait:** se `true`, il server non invia una conferma al client.

Per eliminare uno scambio esistente, chiamare il metodo **DeleteExchange** e passare il nome del canale e il nome dello scambio come argomenti. L'evento **OnAMQPExchangeDelete** viene generato come conferma inviata dal server dell'avvenuta eliminazione dello scambio.

È possibile effettuare anche una chiamata sincrona invocando il metodo **DeleteExchangeEx**; questo metodo restituisce `true` se l'Exchange è stato eliminato e `false` se non è arrivata alcuna conferma dal server.

Comandi | Code AMQP

La classe `queue` consente a un'applicazione di gestire le code di messaggi sul server. Si tratta di un passaggio fondamentale in quasi tutte le applicazioni che consumano messaggi, almeno per verificare che una coda di messaggi attesa sia effettivamente presente.

Il ciclo di vita di una coda di messaggi durabile è abbastanza semplice:

1. Il client verifica l'esistenza della coda messaggi (Declare, con l'argomento "passive").
2. Il server conferma che la coda dei messaggi esiste (Declare-Ok).
3. Il client legge i messaggi dalla coda dei messaggi.

Il ciclo di vita per una coda di messaggi temporanea è più interessante:

1. Il client crea la coda messaggi (Declare, spesso senza nome della coda affinché il server ne assegni uno). Il server conferma (Declare-Ok).
2. Il client avvia un consumer sulla coda di messaggi. La funzionalità precisa di un consumer è definita dalla classe `Basic`.
3. Il client annulla il consumer, esplicitamente o chiudendo il canale e/o la connessione.
4. Quando l'ultimo consumatore scompare dalla coda di messaggi, e dopo un timeout cortese, il server elimina la coda di messaggi.

AMQP implementa il meccanismo di consegna per le sottoscrizioni ai topic come code di messaggi. Ciò consente strutture interessanti in cui una sottoscrizione può essere bilanciata tra un pool di sottoscrittori cooperanti applicazioni.

Il ciclo di vita di una sottoscrizione prevede una fase aggiuntiva di bind:

1. Il client crea la coda messaggi (Declare) e il server conferma (Declare-Ok).
2. Il client associa la coda messaggi a un topic exchange (Bind) e il server conferma (Bind-Ok).
3. Il client utilizza la coda di messaggi come negli esempi precedenti.

Dichiarazione della coda

Questo metodo crea una nuova coda o verifica che una Queue esista già. Il metodo ha i seguenti argomenti:

- **ChannelName:** è il nome del canale (deve essere aperto prima di chiamare questo metodo).
- **QueueName:** è il nome della coda, non deve superare i 255 caratteri e non deve iniziare con "amq." (tranne se il parametro `passive` è true).
- **Passive:** se `passive` è true, il server verifica solo che la coda sia già dichiarata. Se `passive` è false e la coda non esiste, il server ne creerà una nuova.
- **Durable:** se true, la coda verrà ricreata all'avvio del server. Se false, la coda verrà eliminata all'arresto del server.
- **Exclusive:** se true, significa che la coda è accessibile solo dalla connessione corrente.
- **AutoDelete:** se true, la coda verrà eliminata quando tutti i consumer non la utilizzano più.
- **NoWait:** se true, il server non invia una conferma al client.
- **Arguments:** stringa contenente argomenti personalizzati; i valori devono essere passati come stringa json, esempio: `{"x-dead-letter-exchange":"my-dlx"}`.

Per dichiarare una nuova Coda è sufficiente chiamare il metodo **DeclareQueue** e passare come argomenti il nome del canale e il nome della coda. L'evento **OnAMQPQueueDeclare** viene generato come conferma inviata dal server che l'exchange è stato dichiarato.

```
AMQP.DeclareQueue("channel_name", "queue_name");

private void OnAMQPExchangeDeclare(TObject Sender, const string aChannel, const string aQueue,
    int aMessageCount, int aConsumerCount)
{
    DoLog("#AMQP_queue_declare: [" + aChannel + "] " + aQueue));
}
```

È inoltre possibile effettuare una chiamata sincrona richiamando il metodo **DeclareQueueEx**; questo metodo restituisce true se la Queue è stata dichiarata e false se non è arrivata alcuna conferma dal server.

```
if AMQP.DeclareQueueEx("channel_name", "queue_name")
{
    DoLog("#AMQP_queue_declare: [" + aChannel + "] " + aQueue);
}
else
{
    DoLog("#AMQP_queue_declare_error");
}
```

Elimina coda

Questo metodo viene utilizzato per eliminare una Coda esistente. Il metodo ha i seguenti argomenti:

- **ChannelName**: è il nome del canale (deve essere aperto prima di chiamare questo metodo).
- **QueueName**: è il nome della coda, non deve superare i 255 caratteri e non deve iniziare con "amq." (tranne se il parametro passive è true).
- **IfUnused**: il server elimina la coda solo se non vi sono consumer collegati ad essa.
- **IfEmpty**: il server elimina la coda solo se non contiene messaggi.
- **NoWait**: se true, il server non invia una conferma al client.

Per eliminare una Queue esistente, chiamare il metodo **DeleteQueue** passando il nome del canale e il nome della queue come argomenti. L'evento **OnAMQPQueueDelete** viene generato come conferma inviata dal server dell'avvenuta eliminazione della queue.

È possibile effettuare anche una chiamata sincrona richiamando il metodo **DeleteQueueEx**; questo metodo restituisce true se la coda è stata eliminata e false se non è arrivata alcuna conferma dal server.

Associa Coda

Questo metodo viene utilizzato per associare una Coda a un Exchange. Gli Exchange utilizzano i binding per sapere quali code verranno utilizzate per instradare i messaggi.

Tutti i server AMQP associano automaticamente tutte le code all'exchange predefinito (un exchange "direct" senza nome) utilizzando il nome della coda come chiave di routing del binding. Ciò consente di inviare un messaggio a una coda specifica senza dichiarare un binding. È sufficiente chiamare il metodo **PublishMessage**, passare un valore vuoto come nome dell'Exchange e impostare il **RoutingKey** con il valore del **Nome della coda**.

Il metodo ha i seguenti argomenti:

- **ChannelName**: è il nome del canale (deve essere aperto prima di chiamare questo metodo).
- **QueueName**: è il nome della coda, non deve superare i 255 caratteri e non deve iniziare con "amq." (tranne se il parametro passive è true).
- **ExchangeName**: è il nome dell'exchange; non deve superare i 255 caratteri e non deve iniziare con "amq." (eccetto se il parametro passive è true).
- **RoutingKey**: è la chiave di routing del binding.
- **NoWait**: se true, il server non invia una conferma al client.
- **Arguments**: stringa contenente argomenti personalizzati; i valori devono essere passati come stringa json, esempio: {"x-dead-letter-exchange":"my-dlx"}.

Per collegare una coda a un exchange, chiami il metodo **BindQueue** e passi il nome del canale, il nome della coda, l'exchange e la routing key come argomenti. L'evento **OnAMQPQueueBind** viene generato come conferma inviata dal server che la coda è stata collegata.

```
AMQP.BindQueueEx("channel_name", "queue_name", "exchange_name", "routing_key");

private void OnAMQPQueueBind(TObject *Sender, const string aChannel, const string aQueue,
const string aExchange)
{
```

```
DoLog("#AMQP_queue_bind: [" + aChannel + "]" + aQueue + " -->-- " + aExchange);
}
```

È possibile effettuare anche una chiamata sincrona richiamando il metodo **BindQueueEx**; questo metodo restituisce true se la coda è stata associata e false se non è arrivata alcuna conferma dal server.

```
if AMQP.BindQueueEx("channel_name", "queue_name", "exchange_name", "routing_key")
{
    DoLog("#AMQP_queue_bind: [" + aChannel + "]" + aQueue + " -->-- " + aExchange);
}
else
{
    DoLog("#AMQP_queue_bind_error");
}
```

UnBind Queue

Questo metodo elimina un'associazione di coda esistente.

Il metodo ha i seguenti argomenti:

- **ChannelName:** è il nome del canale (deve essere aperto prima di chiamare questo metodo).
- **QueueName:** è il nome della coda, non deve superare i 255 caratteri e non deve iniziare con "amq." (tranne se il parametro passive è true).
- **ExchangeName:** è il nome dell'exchange; non deve superare i 255 caratteri e non deve iniziare con "amq." (eccetto se il parametro passive è true).
- **RoutingKey:** è la chiave di routing del binding.

Per annullare il binding di una coda, chiamare il metodo **UnBindQueue** passando come argomenti il nome del canale, il nome della coda, l'exchange e la routing key. L'evento **OnAMQPQueueUnBind** viene generato come conferma inviata dal server dell'avvenuto annullamento del binding della coda.

È possibile effettuare anche una chiamata sincrona chiamando il metodo **UnBindQueueEx**, questo metodo restituisce true se la Queue è stata Unbound e false se non è arrivata alcuna conferma dal server.

Svuota Coda

Questo metodo elimina tutti i messaggi di una coda. I messaggi già inviati ma in attesa di conferma non vengono interessati.

Il metodo ha i seguenti argomenti:

- **ChannelName:** è il nome del canale (deve essere aperto prima di chiamare questo metodo).
- **QueueName:** è il nome della coda, non deve superare i 255 caratteri e non deve iniziare con "amq." (tranne se il parametro passive è true).
- **NoWait:** se true, il server non invia una conferma al client.

Per svuotare una coda, chiamare il metodo **PurgeQueue** e passare il nome del canale e il nome della coda come argomenti. L'evento **OnAMQPQueuePurge** viene generato come conferma inviata dal server che la coda è stata svuotata.

È possibile effettuare anche una chiamata sincrona chiamando il metodo **PurgeQueueEx**; questo metodo restituisce true se la coda è stata svuotata e false se non è arrivata alcuna conferma dal server.

Comandi | Pubblicazione messaggi AMQP

Pubblicazione di messaggi

Il metodo `PublishMessages` viene utilizzato per inviare un messaggio al server AMQP.

I server AMQP collegano automaticamente le code all'exchange "direct" utilizzando il nome della coda come routing key. Questo consente di inviare un messaggio a una coda specifica senza la necessità di dichiarare un binding (basta chiamare il metodo `PublishMessage` e passare l'argomento `Exchange` come valore vuoto e il nome della coda nell'argomento `RoutingKey`).

Il metodo ha i seguenti argomenti:

- **ChannelName:** è il nome del canale (deve essere aperto prima di chiamare questo metodo).
- **ExchangeName:** è il nome dell'exchange, non deve essere più lungo di 255 caratteri e non deve iniziare con "amq." (tranne se il parametro `passive` è `true`).
- **RoutingKey:** è il nome della chiave di routing del binding.
- **Mandatory:** se `true` e il messaggio non può essere instradato ad alcuna coda, il messaggio viene restituito dal server e viene attivato l'evento `OnAMQPBasicReturn`.
- **Immediate:** se `true` e il messaggio non può essere instradato ad alcuna coda, il messaggio viene restituito dal server e viene generato l'evento `OnAMQPBasicReturn`.

```
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.ReplyText + " " +
}
```

Conferme di pubblicazione

La rete può fallire durante la pubblicazione di un messaggio; l'unico modo per garantire che un messaggio non vada perso è utilizzare le transazioni: per ogni messaggio/i **selezionare la transazione, inviare il messaggio e confermare (commit)**. La conferma di una transazione riuscita si riceve quando viene generato l'evento `OnAMQP-TransactionOk`.

AMQP Consuma Messaggi

I consumer consumano dalle code. Per consumare messaggi deve esistere una coda. Quando viene aggiunto un nuovo consumer, supponendo che siano già presenti messaggi pronti nella coda, le consegne inizieranno immediatamente.

La coda di destinazione può essere vuota al momento della registrazione del consumer. In tal caso le prime consegne avverranno quando verranno accodati nuovi messaggi.

Il consumo di messaggi è un'attività **asincrona**, il che significa che ogni volta che un nuovo messaggio può essere consegnato alla coda del consumatore, viene inviato automaticamente dal server al client. È possibile consultare un metodo alternativo per [ricevere messaggi in modo sincrono](#).

Consuma

Il metodo **Consume** crea un nuovo consumer nella coda e ogni volta che è presente un nuovo messaggio questo verrà recapitato automaticamente al client consumer.

Il metodo ha i seguenti argomenti:

- **ChannelName:** è il nome del canale (deve essere aperto prima di chiamare questo metodo).
- **QueueName:** è il nome della coda, non deve superare i 255 caratteri e non deve iniziare con "amq." (tranne se il parametro **passive** è true).
- **ConsumerTag:** è il nome del consumer e deve essere univoco. Se non è impostato, il server ne crea uno nuovo.
- **NoLocal:** se true, il consumer non consuma mai i messaggi pubblicati sullo stesso canale.
- **NoAck:** se true significa che il server non si aspetta una conferma per ogni messaggio recapitato.
- **Exclusive:** se true impedisce ad altri consumer di consumare messaggi da questa coda.
- **NoWait:** se true, il server non invierà un acknowledgement al client.
- **Arguments:** stringa contenente argomenti personalizzati; i valori devono essere passati come stringa json, esempio: {"x-dead-letter-exchange": "my-dlx"}.

I messaggi vengono consegnati nell'evento **OnAMQPBasicDeliver**.

```
AMQP.Consume("channel_name", "queue_name", "consumer_tag");

private void OnAMQPBasicGetOk(TObject Sender, const string aChannel,
    const TsgcAMQPFramePayload_Method_BasicDeliver aDeliver,
    const TsgcAMQPMessageContent aContent)
{
    DoLog("#AMQP_basic_deliver: " + aChannel + " " + aDeliver.ConsumerTag + " " + aContent.Body.AsString);
}
```

Una chiamata sincrona può essere effettuata chiamando il metodo **ConsumeEx**; questo metodo restituisce true se il Consumer è stato creato e false se non è arrivata alcuna conferma dal server.

Annula Consume

Questo metodo viene utilizzato per annullare una coda consumer esistente.

Il metodo ha i seguenti argomenti:

- **ChannelName:** è il nome del canale (deve essere aperto prima di chiamare questo metodo).
- **ConsumerTag:** è il nome del consumer.
- **NoWait:** se true, il server non invierà un acknowledgement al client.

```
AMQP.CancelConsume("channel_name", "consumer_tag");
```



```
private void OnAMQPBasicCancelConsume(TObject Sender, const string aChannel, const string aConsumerTag)
{
    DoLog("#AMQP_basic_cancel_consume: " + aChannel + " " + aConsumerTag);
}
```

È possibile effettuare una chiamata sincrona chiamando il metodo **CancelConsumeEx**; questo metodo restituisce true se il Consumer è stato annullato e false se non è arrivata alcuna conferma dal server.

Comandi | AMQP Recupera Messaggi

Il recupero dei messaggi è un'operazione **Sincrona**, il che significa che è il client a chiedere al server se ci sono messaggi in coda. È possibile consultare un metodo alternativo per [Ricevere messaggi in modo asincrono](#).

Ottieni Messaggio

Il metodo **GetMessage** invia una richiesta al server AMQP chiedendo se sono disponibili messaggi in una coda. Se ci sono messaggi, questi verranno inviati tramite l'evento **OnAMQPBasicGetOk** e se la coda è vuota, verrà chiamato l'evento **OnAMQPBasicGetEmpty**.

Il metodo ha i seguenti argomenti:

- **ChannelName**: è il nome del canale (deve essere aperto prima di chiamare questo metodo).
- **QueueName**: è il nome della coda, non deve superare i 255 caratteri e non deve iniziare con "amq." (tranne se il parametro **passive** è true).
- **NoWait**: se true, il server non invierà un acknowledgement al client.

```
AMQP.GetMessage("channel_name", "queue_name");

private void OnAMQPBasicGetOk(TObject Sender, const string aChannel,
    const TsgcAMQPFramePayload_Method_BasicGetOk aGetOk,
    const TsgcAMQPMessageContent aContent)
{
    DoLog("#AMQP_basic_GetOk: " + aChannel + " " + IntToStr(aGetOk.MessageCount) + " " + aContent.Body.AsString);
}

private void OnAMQPBasicGetEmpty(TObject Sender, const string aChannel)
{
    DoLog("#AMQP_basic_GetEmpty: " + aChannel);
}
```

È possibile effettuare una chiamata sincrona chiamando semplicemente il metodo **GetMessageEx**; questo metodo restituisce true se la coda contiene messaggi disponibili, altrimenti il risultato sarà false.

Comandi | AMQP QoS

AMQP consente di impostare un livello QoS per limitare il numero di messaggi che il server invia al client prima di attendere il riconoscimento dei messaggi.

Imposta QoS

Il metodo **SetQoS** viene utilizzato per limitare il numero di messaggi che il server invia al client AMQP. Il metodo ha i seguenti argomenti:

- **ChannelName**: è il nome del canale (deve essere aperto prima di chiamare questo metodo).
- **PrefetchSize**: è la dimensione della finestra in byte; il server non invia messaggi al client se la dimensione totale di tutti i messaggi attualmente non confermati già inviati più il messaggio successivo da inviare è superiore all'argomento **PrefetchSize**. Se il valore è zero, significa che non c'è limite.
- **PrefetchCount**: è il numero massimo di messaggi non confermati già inviati e non ancora riconosciuti; se il numero è maggiore, il server smette di inviare messaggi al client.
- **Global**: se true, il QoS si applica a tutti i consumatori esistenti e nuovi della connessione. Se false, il QoS si applica a tutti i consumatori esistenti e nuovi del canale.

La risposta dal server viene ricevuta nell'evento **OnAMQPBasicQoS**.

```
AMQP.SetQoS("channel_name", 1024000, 100, false);

private void OnAMQPBasicQoS(TObject Sender, const string aChannel,
    const TsgcAMQPFramePayload_Method_BasicQoS aQoS)
{
    DoLog("#AMQP_basic_qos: " + aChannel + " " + aQoS.PrefetchSize.ToString() + " "
        + aQoS.PrefetchCount.ToString() + " " + aQoS.Global.ToString());
}
```

Una chiamata sincrona può essere effettuata semplicemente chiamando il metodo **SetQoSEx**, che restituisce true se la richiesta è stata elaborata, altrimenti il risultato sarà false.

Commands | AMQP Transactions

AMQP supporta due tipi di transazioni:

1. Transazioni automatiche, in cui ogni messaggio pubblicato e ogni acknowledgement viene elaborato come una transazione autonoma.
2. Transazioni locali del server, in cui il server bufferizza i messaggi pubblicati e i riconoscimenti e li conferma su richiesta del client.

La classe `Transaction` ("tx") fornisce alle applicazioni l'accesso al secondo tipo, ovvero le transazioni server. La semantica di questa classe è:

1. L'applicazione richiede le transazioni del server in ogni canale per cui desidera tali transazioni (`Select`).
2. L'applicazione svolge il lavoro (`Publish`, `Ack`).
3. L'applicazione esegue il commit o il rollback del lavoro (`Commit`, `Roll-back`).
4. L'applicazione esegue il lavoro, all'infinito.

Le transazioni riguardano i contenuti pubblicati e le conferme, non le consegne. Pertanto, un rollback non reinserisce né riconsegna alcun messaggio, e un client ha il diritto di confermare tali messaggi in una transazione successiva.

I metodi `Transaction` consentono di raggruppare le operazioni di publish e ack in unità di lavoro atomiche. L'intento è che tutte le richieste di publish e ack emesse all'interno di una transazione vengano completate con successo oppure nessuna di esse venga eseguita.

Avvia transazione

Il metodo **`StartTransaction`** avvia una nuova transazione nel server; il client utilizza questo metodo almeno una volta su un canale prima di utilizzare i metodi `Commit` o `Rollback`. L'evento **`OnAMQPTransactionOk`** viene generato quando il server conferma l'uso delle transazioni.

```
AMQP.StartTransaction("channel_name");
```

Una chiamata sincrona può essere effettuata semplicemente chiamando il metodo **`StartTransactionEx`**; questo metodo restituisce `true` se la richiesta è stata elaborata, altrimenti il risultato sarà `false`.

Commit della transazione

Questo metodo conferma tutte le pubblicazioni e i riconoscimenti di messaggi eseguiti nella transazione corrente. Una nuova transazione viene avviata immediatamente dopo un commit. L'evento **`OnAMQPTransactionOk`** viene generato quando il server conferma l'utilizzo delle transazioni.

```
AMQP.CommitTransaction("channel_name");
```

È possibile effettuare una chiamata sincrona semplicemente richiamando il metodo **`CommitTransactionEx`**; questo metodo restituisce `true` se la richiesta è stata elaborata, altrimenti il risultato sarà `false`.

Rollback Transaction

Questo metodo abbandona tutte le pubblicazioni di messaggi e i riconoscimenti eseguiti nella transazione corrente. Una nuova transazione inizia immediatamente dopo un rollback. Si noti che i messaggi non confermati non verranno automaticamente recapitati nuovamente dal rollback; se ciò è necessario, deve essere effettuata una chiamata di recupero esplicita. L'evento **`OnAMQPTransactionOk`** viene generato quando il server conferma l'uso delle transazioni.

```
AMQP.RollbackTransaction("channel_name");
```

Una chiamata sincrona può essere eseguita chiamando semplicemente il metodo **RollbackTransactionEx**, questo metodo restituisce true se la richiesta è stata elaborata, altrimenti il risultato sarà false.

Protocollo AMQP 1.0.0

AMQP (Advanced Message Queuing Protocol) 1.0.0 è un protocollo di messaggistica progettato per la comunicazione affidabile e asincrona tra sistemi distribuiti. Facilita lo scambio di messaggi tra applicazioni o componenti in modo disaccoppiato, consentendo loro di comunicare senza dipendenze dirette. Di seguito è riportata una panoramica tecnica di alcuni aspetti chiave di AMQP 1.0.0:

- **Comunicazione orientata ai messaggi:** AMQP 1.0.0 è incentrato sul concetto di messaggi. I messaggi possono trasportare dati, istruzioni o comandi e sono le unità fondamentali di comunicazione.
- **Message Brokers:** Il protocollo opera su un modello di messaggistica mediata. I broker, che possono essere server o entità intermedie, gestiscono l'instradamento e la consegna dei messaggi tra produttori e consumatori.
- **Code e Exchange:** Le code sono entità di archiviazione all'interno del broker dove i messaggi vengono temporaneamente memorizzati. Gli exchange definiscono le regole per instradare i messaggi dai produttori alle code in base a criteri come il contenuto del messaggio o le chiavi di routing.
- **Indirizzi e Link:** Gli indirizzi identificano le destinazioni dei messaggi all'interno dell'infrastruttura di messaggistica. I link sono canali di comunicazione tra un mittente (produttore) e un destinatario (consumatore) associati a un indirizzo specifico.
- **Sessioni e connessioni:** Le sessioni rappresentano un canale logico per la comunicazione, consentendo più flussi di messaggi all'interno di una singola connessione. Le connessioni gestiscono il collegamento di comunicazione complessivo tra le applicazioni client e il message broker.
- **Security:** AMQP 1.0.0 supporta vari meccanismi di sicurezza, inclusi autenticazione e autorizzazione, per garantire una comunicazione sicura tra client e broker.
- **Indipendenza dal trasporto:** Il protocollo è progettato per essere indipendente dal trasporto, il che significa che può operare su diversi trasporti di rete come TCP, TLS o WebSocket, garantendo flessibilità nel deployment.
- **Controllo del flusso:** AMQP 1.0.0 include meccanismi per il controllo del flusso, consentendo ai consumer di indicare la loro capacità di gestire i messaggi in ingresso a una determinata velocità. Questo aiuta a evitare di sovraccaricare i consumer con un gran numero di messaggi.
- **Gestione degli Errori:** Il protocollo specifica meccanismi per la gestione degli errori, inclusi il riconoscimento e il rifiuto dei messaggi, garantendo robustezza e affidabilità nella consegna dei messaggi.
- **Autenticazione SASL:** Simple Authentication and Security Layer (SASL) viene utilizzato per autenticare e proteggere le connessioni tra client e broker.

Nel complesso, AMQP 1.0.0 fornisce un modo standardizzato e interoperabile affinché diversi componenti software e sistemi comunichino in modo debolmente accoppiato, rendendolo adatto a svariate applicazioni distribuite e di livello enterprise.

Componenti

TsgcWSPClient_AMQP1: è il componente client che implementa il protocollo **AMQP 1.0.0**.

Usi più comuni

- **Connessione**
 - [Client AMQP1 Connect](#)
 - [Client AMQP1 Disconnect](#)
 - [Client AMQP1 Idle Timeout Connection](#)
 - [Client AMQP1 Connection State](#)
 - [Client AMQP1 Authentication](#)
 - [Client AMQP1 Azure Service Bus](#)
- **Comandi**
 - [Sessioni AMQP1](#)
 - [Link AMQP1](#)
 - [Link mittente AMQP1](#)
 - [Link ricevente AMQP1](#)
 - [Invio messaggio AMQP1](#)
 - [Lettura messaggio AMQP1](#)

Connessione | Client AMQP1 Connect

Per connettersi a un server AMQP, è necessario creare prima un [TsgcWebSocketClient](#) e un [TsgcWSPClient_AMQP1](#). Successivamente è necessario collegare il componente AMQP1 al WebSocket Client.

Dopo una connessione riuscita, viene attivato l'evento OnAMQPConnect.

Utilizzo di base

Si connette a un server AMQP 1.0.0 senza autenticazione. Definisca i valori della proprietà AMQPOptions, il virtual host e quindi imposti nel TsgcWebSocketClient l'Host e la Port del server.

Se si utilizza una connessione TCP Plain, impostare la proprietà Specifications.RFC6455 del TsgcWebSocketClient su false.

```
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;
```

Autenticazione

Se il server richiede l'autenticazione, utilizzare le proprietà AMQP:Authentication per impostare i valori di Username/Password e impostare AuthType sul valore "amqp1authSASLPlain".

```
oAMQP = new TsgcWSPClient_AMQP1(this);
// Setting AMQP authentication options
oAMQP.AMQPOptions.Authentication.AuthType = amqp1authSASLPlain;
oAMQP.AMQPOptions.Authentication.Username = "sgc";
oAMQP.AMQPOptions.Authentication.Password = "sgc";
// Creating WebSocket client
oClient = new TsgcWebSocketClient(this);
// Setting WebSocket specifications
oClient.Specifications.RFC6455 = false;
// Setting WebSocket client properties
oClient.Host = "www.esegece.com";
oClient.Port = 5671;
oClient.TLS = true;
// Assigning WebSocket client to AMQP client
oAMQP.Client = oClient;
// Activating WebSocket client
oClient.Active = true;
```


Connessione | Disconnessione Client AMQP1

Il client può disconnettere una connessione attiva corrente utilizzando i seguenti metodi:

Invio di un motivo di chiusura

Il client AMQP può informare il server che la connessione verrà chiusa e fornire informazioni sul motivo della chiusura. Utilizzare il metodo `Close` per richiedere la chiusura della connessione al server.

```
oAMQP.Close('invalid-frame', "The received frame has an invalid format.");
```

Await Close

Per impostazione predefinita, il metodo **Close** è **asincrono**, quindi dopo aver chiamato il metodo, il codice prosegue. Se desidera **attendere il completamento del metodo Close** e la ricezione della conferma inviata dal server, imposti la proprietà **Await** su `True` nel parametro `Options`.

```
public static void Close(string aCondition, string aDescription)
{
    TsgcAMQP1MethodOptions_Close oOptions = new TsgcAMQP1MethodOptions_Close();
    try
    {
        oOptions.ErrorCondition = aCondition;
        oOptions.ErrorDescription = aDescription;
        oOptions.Await = true;
        AMQP1.Close(oOptions);
    }
    finally
    {
        oOptions.Dispose();
    }
}
```

Chiusura della connessione socket

Impostare semplicemente la proprietà `Active` di [TsgcWebsocketClient](#) su `False`. È possibile leggere ulteriori informazioni sulla [chiusura delle connessioni](#).

Connection | Idle Timeout

Le connessioni sono soggette a una soglia di **timeout per inattività**. Il timeout viene attivato dal client quando non vengono ricevuti frame

dal server dopo che viene superato un valore di soglia. Il timeout per inattività è misurato in millisecondi e inizia dal momento in cui viene ricevuto l'ultimo frame. Se la soglia viene superata, il componente invia un Close Frame al server. Se il server non risponde entro 10 secondi, il client chiuderà il socket TCP.

Il valore del timeout di inattività può essere configurato nella proprietà:

AMQPOptions.IdleTimeout

Il valore impostato in questa proprietà verrà inviato al server all'apertura della connessione AMQP. Se il valore è maggiore di zero e minore della metà del valore MaxInt, verrà abilitato un timer interno per verificare che il timeout di inattività non sia stato superato.

Esempio: imposti un valore IdleTimeout di 60 secondi

AMQPOptions.IdleTimeout = 60000

Connessione | Stato della Connessione

AMQP 1.0.0 definisce i seguenti stati di connessione:

- **amqp1csUnknown:** stato iniziale.
- **amqp1csStart:** In questo stato esiste una connessione, ma non è stato inviato né ricevuto nulla. Questo è lo stato in cui si trova un'implementazione immediatamente dopo aver eseguito una connessione socket o un'accettazione socket.
- **amqp1csHeaderReceived:** In questo stato l'intestazione di connessione è stata ricevuta dal peer ma non è stata ancora inviata un'intestazione di connessione.
- **amqp1csHeaderSent:** In questo stato l'intestazione di connessione è stata inviata al peer ma non è stata ricevuta alcuna intestazione di connessione.
- **amqp1csHeaderExchanged:** In questo stato l'intestazione di connessione è stata inviata al peer e un'intestazione di connessione è stata ricevuta dal peer.
- **amqp1csOpenPipe:** In questo stato sono stati inviati sia l'header di connessione sia il frame open, ma non è stato ricevuto nulla.
- **amqp1csOpenClosePipe:** In questo stato, l'intestazione di connessione, il frame open, qualsiasi traffico di connessione in pipeline e il frame close sono stati inviati ma non è stato ricevuto nulla.
- **amqp1csOpenReceived:** In questo stato le intestazioni di connessione sono state scambiate. Un frame open è stato ricevuto dal peer ma non è stato ancora inviato un frame open.
- **amqp1csOpenSent:** In questo stato le intestazioni di connessione sono state scambiate. È stato inviato un frame open al peer ma non è ancora stato ricevuto alcun frame open.
- **amqp1csClosePipe:** In questo stato, le intestazioni di connessione sono state scambiate. Sono stati inviati un frame open, tutto il traffico di connessione con pipeline e il frame close, ma non è ancora stato ricevuto alcun frame open dal peer.
- **amqp1csOpened:** In questo stato sia l'intestazione di connessione che il frame open sono stati inviati e ricevuti.
- **amqp1csCloseReceived:** In questo stato è stato ricevuto un frame di chiusura che indica che il peer ha avviato una chiusura AMQP. Non si prevede che ulteriori frame arrivino sulla connessione; tuttavia, è ancora possibile inviare frame. Se lo si desidera, un'implementazione PUÒ effettuare a questo punto una half-close TCP per chiudere il lato di lettura della connessione.
- **amqp1csCloseSent:** In questo stato è stato inviato un frame di chiusura al peer. Non è consentito scrivere altro sulla connessione, tuttavia potrebbero ancora arrivare frame in ingresso. Se necessario, un'implementazione PUÒ eseguire un TCP half-close a questo punto per chiudere il lato di scrittura della connessione.
- **amqp1csDiscarding:** Lo stato DISCARDING è una variante dello stato CLOSE SENT in cui la chiusura è innescata da un errore. In questo caso tutti i frame in arrivo sulla connessione DEVONO essere silenziosamente scartati finché non viene ricevuto il frame di chiusura del peer.
- **amqp1csEnd:** In questo stato è vietato per entrambi gli endpoint scrivere qualsiasi cosa sulla connessione. La connessione può essere chiusa e eliminata in modo sicuro.

Il client AMQP ha la proprietà **ConnectionState** dove è possibile verificare in quale stato di connessione si trova il componente client.

Connessione | Autenticazione AMQP1

Il componente dispone dei seguenti metodi di autenticazione:

- **amqp1authNone:** non viene utilizzato alcun metodo di autenticazione per la connessione al server.
- **amqp1authSASLAnonymous:** si connette in modo anonimo.
- **amqp1authSASLPlain:** il valore predefinito, utilizza un'autenticazione utente/password.
- **amqp1authSASLExternal:** attualmente non supportato.

Autenticazione SASL

L'autenticazione più comune è quella di tipo **amqp1authSASLPlain**. Questo tipo di autenticazione può essere abilitato nel componente AMQP1, accedendo alla proprietà `AMQPOptions.Authentication`.

- **AuthType:** selezioni `amqp1authSASLPlain`
- **Username:** l'utente da utilizzare per l'autenticazione SASL.
- **Password:** il valore segreto da utilizzare per l'autenticazione SASL.

Il risultato dell'autenticazione SASL può essere ottenuto quando si genera l'evento **OnAMQPSASLAuthentication**.

```
public void OnAMQP1SASLAuthentication(object Sender, TsgcAMQP1SaslCode aCode,
    string aDescription, ref bool Handled)
{
    MessageBox.Show("#sas1-authentication: " + aDescription);
}
```

Connessione | Azure MessageBus

AMQP (Advanced Message Queuing Protocol) è un robusto sistema di messaggistica progettato per facilitare la comunicazione tra contenitori eterogenei su vari nodi. Standardizza sia il protocollo per la trasmissione dei messaggi sia il framework strutturale dei messaggi stessi, garantendo una comunicazione coerente e affidabile. Per approfondire i fondamenti di AMQP, consulti la nostra guida *Getting Started with AMQP*.

Il componente AMQP all'interno della libreria eSeGeCe consente un'integrazione fluida con i principali broker di messaggistica cloud, tra cui Amazon MQ e Azure Service Bus. Questa guida si concentra sull'utilizzo del componente AMQP per connettersi con Azure Service Bus, dimostrando come costruire un'applicazione multi-tenant in grado di inviare e ricevere messaggi in modo efficiente.

Il componente offre un'implementazione completa con supporto per funzionalità chiave come code, topic e sottoscrizioni, rendendolo una scelta ideale per le moderne applicazioni IoT ed enterprise.

Configurazione Azure

Per iniziare, creare una **risorsa Service Bus** nel portale Azure. Una volta stabilita la risorsa, assicurarsi di annotare il **nome di dominio della risorsa**, poiché sarà essenziale per l'integrazione e la configurazione.

Dopo che il **namespace** è stato creato con successo, è possibile gestirlo e monitorarlo direttamente dalla **panoramica del namespace** nel portale Azure. Questa interfaccia centralizzata fornisce accesso agli strumenti di gestione e alle impostazioni principali, consentendo un'amministrazione fluida della risorsa Service Bus.

Quando si utilizza l'**autenticazione SAS**, il **nome utente** è il **nome del criterio SAS** e la **password** è la **chiave primaria o secondaria**.

```
// Create TCP client
var oClient = new TsgcWebSocketClient
{
    Specifications = { RFC6455 = false },
    Host = "esegece.servicebus.windows.net",
    Port = 5671,
    TLS = true
};
// Create AMQP1 protocol client
var oAMQP1 = new TsgcWSClient_AMQP1
{
    Specifications = { RFC6455 = false },
    AMQPOptions =
    {
        Authentication = {
            AuthType = AMQP1AuthType.SASLPlain,
            Username = "RootManageSharedAccessKey",
            Password = "BhJ78+w8kMXhs/eE/nBy0cRzodx9tipbi+ASbAXIah8="
        }
    }
},
Client = oClient;
// Connect to the server
oClient.Active = true;
```

Autenticazione Azure CBS

Azure Service Bus implementa la sicurezza basata su attestazioni (CBS) su AMQP per autorizzare mittenti e destinatari dopo l'handshake SASL iniziale. Il client apre un collegamento di gestione al nodo **\$cbs** e invia una richiesta **put-token** contenente un token SAS (Shared Access Signature) o un JSON Web Token (JWT) emesso da Microsoft Entra ID. Una volta che il broker convalida il token, l'autorizzazione viene memorizzata nella cache per tutta la sua durata e l'applicazione può procedere a creare collegamenti mittente e destinatario verso code, argomenti o sottoscrizioni.

Il client AMQP1 automatizza questo flusso tramite due metodi helper:

- **CreateAzureCbsSasToken** stabilisce una coppia di link mittente/ricevente CBS, genera un token SAS per l'entità di destinazione e lo pubblica su **\$cbs**. Utilizzarlo per l'autenticazione con una policy di accesso condizionale.
- **CreateAzureCbsJWT** segue lo stesso scambio CBS ma ottiene un access token da Microsoft Entra ID (Azure AD) utilizzando il grant client-credentials prima di inviare il JWT a **\$cbs**.

Entrambi i metodi richiedono una connessione AMQP attiva e accettano i seguenti parametri:

- **aName**: Identificatore per la coppia di link CBS creata internamente.
- **aNameSpace** e **aEntityName**: Il namespace del Service Bus (senza il suffisso *.servicebus.windows.net*) e il percorso della coda, topic o subscription utilizzato per costruire l'audience del token.
- **aKeyName** / **aKeyValue**: Nome e chiave della policy di accesso condiviso per i token SAS. Il componente firma il token e lo invia usando il tipo di token *servicebus.windows.net:sastoken*.
- **aTenant**, **aApplicationId**, **aSecret**: ID directory Microsoft Entra (Azure AD), ID applicazione (client) e segreto client utilizzati per richiedere il JWT con il flusso di credenziali client.
- **aListeningPort** (JWT): Porta HTTP locale per il reindirizzamento OAuth 2.0 (il valore predefinito è 8080 se non specificato).
- **aExpiration** e **aTimeout**: Durata del token emesso (in secondi) e tempo massimo di attesa (in millisecondi) per la negoziazione CBS.
- **aRaiseIfError**: Quando impostato su *True*, il metodo genera un'eccezione se l'acquisizione del token o la risposta CBS fallisce.

I seguenti esempi illustrano come autenticarsi con CBS prima di inviare messaggi.

```
// Create TCP client
var oClient = new TsgcWebSocketClient
{
    Specifications = { RFC6455 = false },
    Host = "esegece.servicebus.windows.net",
    Port = 5671,
    TLS = true
};
// Create AMQP1 protocol client
var oAMQP1 = new TsgcWSClient_AMQP1
{
    Specifications = { RFC6455 = false },
    AMQPOptions =
    {
        Authentication = { AuthType = AMQP1AuthType.SASLAnonymous }
    },
    Client = oClient
};
// Connect and publish SAS token through CBS
oClient.Active = true;
oAMQP1.CreateAzureCbsSasToken("cbs", "esegece", "queue1",
    "RootManageSharedAccessKey",
    "BhJ78+w8kMXhS/eE/nBy0cRz0dx9tipbi+ASbAXIaH8=",
    3600, 10000, true);
```

Il prossimo esempio si concentra esclusivamente sull'autenticazione tramite Microsoft Entra ID (Azure AD) con JWT. Mostra come richiedere un token con il flusso delle credenziali client e pubblicarlo su **\$cbs** prima di creare link per inviare o ricevere messaggi.

```
// Create TCP client
var oClient = new TsgcWebSocketClient
{
    Specifications = { RFC6455 = false },
    Host = "esegece.servicebus.windows.net",
    Port = 5671,
    TLS = true
};
// Create AMQP1 protocol client
var oAMQP1 = new TsgcWSClient_AMQP1
{
    Specifications = { RFC6455 = false },
    AMQPOptions =
    {
        Authentication = { AuthType = AMQP1AuthType.SASLAnonymous }
    },
    Client = oClient
};
// Connect and publish JWT through CBS
```

```
oClient.Active = true;
oAMQP1.CreateAzureCbsJWT("cbs", "esegece", "queue1",
    "00000000-0000-0000-0000-000000000000", // Tenant ID
    "11111111-1111-1111-1111-111111111111", // Application ID
    "client-secret", 8080, 3600, 10000, true);
```

Comandi | AMQP1 Sessions

Nel contesto della specifica AMQP (Advanced Message Queuing Protocol) 1.0.0, una sessione rappresenta un contesto logico per la comunicazione tra un client e un message broker. Di seguito è riportata una descrizione di cosa comporta una sessione AMQP 1.0.0:

- **Contesto logico:** Una sessione stabilisce un contesto logico per le operazioni di messaggistica tra un client AMQP (produttore o consumatore) e un broker AMQP. Fornisce un modo per raggruppare le operazioni di messaggistica correlate all'interno di una singola connessione.
- **Canale di comunicazione:** Le sessioni fungono da canali di comunicazione attraverso cui i messaggi vengono inviati e ricevuti. Esse racchiudono lo scambio di messaggi, le conferme di ricezione e i meccanismi di controllo del flusso.
- **Confini transazionali:** le sessioni definiscono i confini transazionali per le operazioni sui messaggi. Consentono di raggruppare più invii o ricezioni di messaggi in un'unica unità atomica, garantendo che tutte le operazioni all'interno della sessione vengano elaborate con successo oppure nessuna venga elaborata.
- **Controllo del flusso:** Le sessioni supportano meccanismi di controllo del flusso per regolare la velocità con cui i messaggi vengono scambiati tra il client e il broker. Il controllo del flusso aiuta a prevenire il sovraccarico delle risorse di entrambe le parti, garantendo una consegna efficiente e affidabile dei messaggi.
- **Gestione del Ciclo di Vita:** Le sessioni hanno un ciclo di vita che inizia quando vengono create e termina quando vengono chiuse. I client possono stabilire più sessioni all'interno di una singola connessione per parallelizzare l'elaborazione dei messaggi o isolare i flussi di messaggi.
- **Allocazione delle risorse:** le sessioni possono essere associate a risorse specifiche come code, topic o sottoscrizioni all'interno del broker. I messaggi inviati o ricevuti nell'ambito di una sessione sono legati a queste risorse, consentendo un instradamento e una consegna mirati dei messaggi.

In sintesi, una sessione AMQP 1.0.0 fornisce un contesto logico per lo scambio di messaggi tra un client AMQP e un broker, facilitando l'integrità transazionale, il controllo del flusso e la gestione delle risorse all'interno del sistema di messaggistica. Definisce i confini entro cui vengono eseguite le operazioni di messaggistica e contribuisce a garantire uno scambio di messaggi efficiente e affidabile.

Apri sessione

Il metodo **CreateSession** crea una nuova sessione con il nome specificato (o, se vuoto, con un nome casuale); se la sessione esiste già, viene sollevata un'eccezione. Il client consente di creare più sessioni utilizzando la stessa connessione AMQP.

Una volta che la sessione è stata creata con successo, viene attivato l'evento **OnAMQPSessionOpen**.

```
oAMQP.OnAMQPSessionOpen += OnAMQPSessionOpenEvent;
oAMQP.CreateSession("MySession");
private void OnAMQP1AMQPSessionOpenEvent(object sender, TsgcAMQP1Session aSession, TsgcAMQP1FrameBegin aBegin)
{
    Console.WriteLine("#session-open: " + aSession.Id);
}
```

Il metodo **CreateSession** restituisce la classe **TsgcAMQP1Session** che contiene le informazioni sulla sessione.

Attendi apertura sessione

Per impostazione predefinita, il metodo **CreateSession** è **Asincrono**, quindi dopo aver chiamato il metodo, il codice continua. Se si desidera **attendere il completamento del metodo CreateSession** e la ricezione della conferma inviata dal server, impostare la proprietà **Await** a **True** nel parametro **Options**.

```
public static void OpenSession(string aSession)
{
```



```

TsgcAMQP1MethodOptions_SessionOpen oOptions = new TsgcAMQP1MethodOptions_SessionOpen();
try
{
    oOptions.Await = true;
    AMQP1.CreateSession(aSession, oOptions);
}
finally
{
    oOptions.Dispose();
}
}

```

Chiudi sessione

Per chiudere una sessione esistente, utilizzare il metodo **CloseSession** passando il nome della sessione da chiudere.

Una volta che la sessione viene chiusa con successo, viene generato l'evento **OnAMQPSessionClose**.

```

oAMQP.OnAMQPSessionClose += OnAMQPSessionCloseEvent;
oAMQP.CloseSession("MySession");
private void OnAMQPSessionCloseEvent(object Sender, TsgcAMQP1Session aSession, TsgcAMQP1FrameEnd aEnd)
{
    Console.WriteLine("#session-close: " + aSession.Id + " [" + aSession.Channel.ToString() + "] reason: " + aEnd
}

```

Attendi la chiusura della sessione

Per impostazione predefinita, il metodo **CloseSession** è **asincrono**, quindi dopo aver chiamato il metodo il codice prosegue. Se desidera **attendere il completamento del metodo CloseSession** e la ricezione della conferma inviata dal server, imposti la proprietà **Await** su **True** nel parametro Options.

```

public static void CloseSession(string aSession)
{
    TsgcAMQP1MethodOptions_SessionClose oOptions = new TsgcAMQP1MethodOptions_SessionClose();
    try
    {
        oOptions.Await = true;
        AMQP1.CloseSession(aSession, oOptions);
    }
    finally
    {
        oOptions.Dispose();
    }
}

```

Comandi | AMQP1 Link

Nella specifica AMQP (Advanced Message Queuing Protocol) 1.0.0, un link rappresenta un canale di comunicazione unidirezionale tra un client AMQP e un broker di messaggi. Approfondiamo cosa comportano i link AMQP 1.0.0:

- **Canale di comunicazione:** Un collegamento funge da percorso attraverso cui i messaggi fluiscono tra un mittente e un ricevitore AMQP. Consente la trasmissione di messaggi in una direzione, dal mittente al ricevitore o viceversa.
- **Flusso unidirezionale:** ogni collegamento è unidirezionale, il che significa che i messaggi possono viaggiare in una sola direzione lungo il collegamento. Se è necessaria una comunicazione bidirezionale, devono essere stabiliti due collegamenti — uno per ciascuna direzione.
- **Trasferimento Messaggi:** I messaggi vengono trasferiti tra i link secondo le regole del protocollo AMQP. Questi messaggi possono includere payload, proprietà del messaggio e metadati aggiuntivi necessari per la comunicazione.
- **Resource Binding:** I collegamenti sono associati a risorse specifiche all'interno del broker AMQP, come code, topic o exchange. I messaggi inviati o ricevuti tramite un collegamento sono indirizzati verso o originati da queste risorse.
- **Controllo del flusso:** i link supportano meccanismi di controllo del flusso per regolare la velocità con cui i messaggi vengono inviati o ricevuti. Il controllo del flusso garantisce che né il mittente né il destinatario siano sopraffatti dal volume dei messaggi scambiati.
- **Gestione del ciclo di vita:** I collegamenti hanno un ciclo di vita che inizia quando vengono stabiliti e termina quando vengono chiusi. Possono essere creati dinamicamente secondo necessità e chiusi quando non sono più necessari.
- **Addressing:** I link sono identificati da indirizzi univoci che specificano gli endpoint sorgente e destinazione della comunicazione. Questi indirizzi consentono a client e broker di identificare e stabilire connessioni agli endpoint appropriati.
- **Confini transazionali:** I link definiscono i confini transazionali per le operazioni sui messaggi. Consentono di raggruppare più invii o ricezioni di messaggi in un'unica unità atomica, garantendo coerenza e affidabilità nella consegna dei messaggi.

In sintesi, i link AMQP 1.0.0 forniscono un mezzo per la comunicazione unidirezionale tra client AMQP e broker, facilitando il trasferimento di messaggi supportando al contempo il controllo del flusso, il binding delle risorse, l'indirizzamento e l'integrità transazionale all'interno del sistema di messaggistica. Costituiscono i blocchi fondamentali dello scambio di messaggi nel protocollo AMQP.

Esistono 2 tipi di Link:

- **Sender Links:** questi link vengono utilizzati per inviare messaggi.
- **Receiver Links:** questi link vengono utilizzati per ricevere messaggi.

Ogni volta che viene creato o eliminato un nuovo link, vengono generati i seguenti eventi:

- **OnAMQPLinkOpen:** questo evento viene attivato quando viene creato un nuovo collegamento. Utilizzare la proprietà `aLink.Mode` per verificare se il collegamento è in modalità ricevitore o mittente.
- **OnAMQPLinkClose:** questo evento viene generato quando un link viene chiuso.

Comandi | Link Mittente AMQP1

Nel protocollo AMQP 1.0.0, un **Sender Link** è un **canale di comunicazione** stabilito tra un client AMQP e un server AMQP allo scopo di **inviare messaggi**. Opera nel contesto di una sessione AMQP, che rappresenta un canale logico per la comunicazione tra il client e il server.

Crea collegamento mittente

Per **creare un nuovo Sender Link**, chiamare il metodo **CreateSenderLink** che contiene i seguenti parametri:

- **Session:** il nome della sessione a cui verrà collegato il link mittente.
- **Name:** (facoltativo) il nome del link del mittente; se non è impostato, verrà assegnato automaticamente un nome casuale.
- **Target:** (facoltativo) è possibile specificare la destinazione in cui i messaggi devono essere ricevuti sull'host remoto impostando il parametro "target". Tuttavia, in certi scenari, la specifica del target potrebbe non essere necessaria. In questi casi, sarà sufficiente fornire una stringa vuota.
- **SndSettleMode:** (mixed per impostazione predefinita) AMQP offre la possibilità di negoziare le garanzie di consegna tramite il meccanismo di liquidazione dei messaggi. All'apertura di un collegamento, mittente e destinatario negoziano e concordano una modalità di liquidazione (una per ciascun ruolo). I mittenti operano in una delle seguenti modalità:
 - **amqp1ssmSettled:** Il messaggio viene considerato consegnato e riconosciuto con successo una volta inviato.
 - **amqp1ssmUnsettled:** Il messaggio non viene considerato liquidato finché non è esplicitamente accettato o rifiutato dal destinatario. Ciò consente un maggiore controllo sull'elaborazione e la gestione dei messaggi.
 - **amqp1ssmMixed:** In una singola sessione AMQP è possibile utilizzare una combinazione di modalità settled e unsettled. Utilizzare il parametro **MessageOptions** del metodo **SendMessage** per configurare se il messaggio è liquidato o meno.

Quando il Sender Link è stato creato con successo, verrà generato l'evento **OnAMQP1LinkOpen**.

```
oAMQP1.OnAMQP1LinkOpen += AMQP1AMQP1LinkOpen;
oAMQP1.CreateSenderLink("MySession", "MySenderLink");
private void AMQP1AMQP1LinkOpen(object sender, TsgcAMQP1Session aSession, TsgcAMQP1Link aLink, TsgcAMQP1FrameAttac
{
    Console.WriteLine("#link-open: " + aLink.Name);
}
```

Attendi la Creazione del Sender Link

Per impostazione predefinita, il metodo **CreateSenderLink** è **asincrono**; dopo la chiamata al metodo, il codice continua. Se si desidera **attendere il completamento del metodo CreateSenderLink** e ricevere la conferma dal server, impostare la proprietà **Await** su True nel parametro Options.

```
public static void CreateSenderLink(string aSession, string aSender)
{
    TsgcAMQP1MethodOptions_CreateSenderLink oOptions = new TsgcAMQP1MethodOptions_CreateSenderLink();
    try
    {
        oOptions.Await = true;
        AMQP1.CreateSenderLink(aSession, aSender, "", oOptions);
    }
    finally
    {
        oOptions.Dispose();
    }
}
```

Invio messaggi

Per inviare un nuovo messaggio, chiami il metodo `SendMessage` che contiene i seguenti parametri:

- **Session:** nome della sessione.
- **Link:** nome del link mittente.
- **Text:** il testo del messaggio stringa.

```
oAMQP1.SendMessage("MySession", "MySenderLink", "My first AMQP Message");
```

Invio di messaggi in modalità Mixed

Quando il Sender Link viene creato in modalità Mixed (l'impostazione predefinita), durante l'invio di un messaggio, l'utente può impostare se desidera che il messaggio sia **settled** o **no**. Utilizzi il parametro **MessageOptions** per definire se il messaggio è settled o meno.

```
TsgcAMQP1MessageOptions oMessageOptions = new TsgcAMQP1MessageOptions();
try
{
    oMessageOptions.Settled = true;
    oAMQP1.SendMessage("MySession", "MySenderLink", "MyMessage", "message-id", oMessageOptions);
}
finally
{
    oMessageOptions.Dispose();
}
```

Chiudi link mittente

Per chiudere un Sender Link esistente, chiamare il metodo **CloseLink** che contiene i seguenti parametri:

- Session: nome della sessione che contiene il link.
- Link: nome del link mittente.
- Error: (opzionale) qui può impostare il motivo per cui il link viene chiuso.

Quando il Sender Link è stato chiuso correttamente, verrà generato l'evento **OnAMQP1LinkClose**.

```
oAMQP.CloseLink("MySession", "MySenderLink");
public void OnAMQP1LinkCloseEvent(object sender, TsgcAMQP1Session aSession, TsgcAMQP1Link aLink, TsgcAMQP1FrameDet
{
    Console.WriteLine("#link-close: " + aLink.Name);
}
```

Await Close Sender Link

Per impostazione predefinita, il metodo **CloseLink** è **asincrono**, quindi dopo aver chiamato il metodo il codice continua. Se si desidera **attendere il completamento del metodo CloseLink** e la ricezione della conferma inviata dal server, impostare la proprietà **Await** su **True** nel parametro Options.

```
public static void CloseSenderLink(string aSession, string aSenderLink)
{
    TsgcAMQP1MethodOptions_CloseLink oOptions = new TsgcAMQP1MethodOptions_CloseLink();
    try
    {
        oOptions.Await = true;
        AMQP1.CloseLink(aSession, aSenderLink, oOptions);
    }
    finally
    {
    }
```

```
{  
    oOptions.Dispose();  
}
```

Comandi | AMQP1 Receiver Links

Nel protocollo AMQP 1.0.0, un **Receiver Link** è un **canale di comunicazione** stabilito tra un client AMQP e un server AMQP per la **ricezione di messaggi**. Opera nel contesto di una sessione AMQP, che rappresenta un canale logico per la comunicazione tra client e server.

Crea collegamento ricevitore

Per **creare un nuovo Receiver Link**, chiami il metodo **CreateReceiverLink** che contiene i seguenti parametri:

- **Session:** il nome della sessione a cui verrà collegato il link mittente.
- **Name:** (facoltativo) il nome del link del mittente; se non è impostato, verrà assegnato automaticamente un nome casuale.
- **Source:** (opzionale) il source può essere configurato per indicare la posizione del nodo sull'host remoto che dovrebbe fungere da mittente. In alcune situazioni, specificare questo indirizzo potrebbe non essere necessario. In tali casi, è sufficiente fornire una stringa vuota come valore per i parametri.
- **ReadMode:** (amqp1srmAuto per impostazione predefinita) I receiver link possono funzionare in una delle due modalità per la ricezione dei messaggi:
 - **amqp1srmAuto:** Modalità automatica, in questa modalità il receiver lavora attivamente per garantire che i messaggi vengano ricevuti prontamente non appena diventano disponibili. Ascolta e riceve automaticamente i messaggi senza alcuna istruzione esplicita ogni volta che arriva un nuovo messaggio.
 - **amqp1srmManual:** Modalità basata sul fetch, in questa modalità il receiver recupera o effettua il fetch di un nuovo messaggio solo quando gli viene specificamente richiesto. A differenza della modalità automatica, il receiver non ascolta attivamente i nuovi messaggi ma attende invece istruzioni manuali per effettuare il fetch del messaggio successivo.
- **RcvSettleMode:** (amqp1srmFirst per impostazione predefinita) I link del Receiver operano in una delle seguenti modalità:
 - **amqp1srmFirst:** Quando i messaggi arrivano, vengono elaborati e confermati immediatamente. Se il messaggio non è già stato confermato al momento dell'invio, il mittente viene informato che il messaggio è stato ricevuto.
 - **amqp1srmSecond:** I messaggi in arrivo vengono confermati solo dopo che il mittente li ha confermati per primo. Inoltre, il mittente riceve una notifica quando un messaggio è stato ricevuto, a condizione che il messaggio non fosse già stato confermato al momento dell'invio.

Quando il Receiver Link è stato creato con successo, verrà attivato l'evento **OnAMQPLinkOpen**.

```
oAMQP1.OnAMQPLinkOpen += AMQP1AMQPLinkOpen;
oAMQP1.CreateReceiverLink("MySession", "MyReceiverLink");
private void AMQP1AMQPLinkOpen(object sender, TsgcAMQP1Session aSession, TsgcAMQP1Link aLink, TsgcAMQP1FrameAttac
{
    Console.WriteLine("#link-open: " + aLink.Name);
}
```

Await Create Receiver Link

Per impostazione predefinita, il metodo **CreateReceiverLink** è **asincrono**, quindi dopo la chiamata al metodo, il codice continua. Se si desidera **attendere il completamento del metodo CreateReceiverLink** e la ricezione della conferma inviata dal server, impostare la proprietà **Await** su **True** nel parametro Options.

```
public static void CreateReceiverLink(string aSession, string aReceiver)
{
    TsgcAMQP1MethodOptions_CreateReceiverLink oOptions = new TsgcAMQP1MethodOptions_CreateReceiverLink();
    try
    {
        oOptions.Await = true;
        AMQP1.CreateReceiverLink(aSession, aReceiver, "", oOptions);
    }
}
```

```

    }
    finally
    {
        oOptions.Dispose();
    }
}

```

Messaggi Sincroni

Quando il Receiver Link opera in modalità ReadMode manuale, chiamare il metodo **GetMessage** per ricevere nuovi messaggi. Questo metodo è sincrono, il che significa che attende fino al superamento di un timeout (per impostazione predefinita 10 secondi). Quando il metodo viene chiamato, il componente incrementa il credito di un'unità e attende l'arrivo di un nuovo messaggio o il superamento del timeout. Se non arriva alcun messaggio, il credito viene azzerato di nuovo.

Il metodo **GetMessage** ha i seguenti parametri:

- **Session:** nome della sessione che contiene il link.
- **Link:** nome del collegamento del destinatario.
- **Timeout:** (predefinito 1000 = 10 secondi) il tempo massimo che la funzione attenderà per ricevere un nuovo messaggio.

Chiudi link ricevitore

Per chiudere un Receiver Link esistente, chiamare il metodo **CloseLink** che contiene i seguenti parametri:

- **Session:** nome della sessione che contiene il link.
- **Link:** nome del collegamento del destinatario.
- **Error:** (opzionale) qui è possibile impostare il motivo per cui il collegamento viene chiuso.

Quando il Receiver Link è stato chiuso correttamente, verrà generato l'evento **OnAMQPLinkClose**.

```

oAMQP1.CloseLink("MySession", "MyReceiverLink");
public void OnAMQPLinkCloseEvent(object sender, TsgcAMQP1Session aSession, TsgcAMQP1Link aLink, TsgcAMQP1FrameDet
{
    Console.WriteLine("#link-close: " + aLink.Name);
}

```

Await Close Receiver Link

Per impostazione predefinita, il metodo **CloseLink** è **asincrono**, quindi dopo aver chiamato il metodo il codice continua. Se si desidera **attendere il completamento del metodo CloseLink** e la ricezione della conferma inviata dal server, impostare la proprietà **Await** su True nel parametro Options.

```

public static void CloseReceiverLink(string aSession, string aReceiverLink)
{
    TsgcAMQP1MethodOptions_CloseLink oOptions = new TsgcAMQP1MethodOptions_CloseLink();
    try
    {
        oOptions.Await = true;
        AMQP1.CloseLink(aSession, aReceiverLink, oOptions);
    }
    finally
    {
        oOptions.Dispose();
    }
}

```

AMQP1 | Invia messaggio

Leggere prima [AMQP1 Sender Links](#) per sapere come creare un Sender Link.

Invia messaggio

Utilizzare il metodo **SendMessage** passando il nome della Session e del SenderLink per inviare un messaggio di testo al server AMQP1. Il metodo ha i seguenti parametri:

- **Session:** nome della sessione.
- **Link:** nome del link mittente.
- **Text:** testo del messaggio.
- **MessageId:** (opzionale) l'id del messaggio, può essere utilizzato in modalità unsettled per sapere se il server ha elaborato il messaggio.
- **Options:** (facoltativo) consente di personalizzare alcune opzioni durante l'invio del messaggio.
 - **Settled:** quando si utilizza un link mittente in modalità mista, la proprietà Settled può essere personalizzata durante l'invio di un messaggio.
 - **Await:** se il messaggio non è settled e il valore è true, il codice attenderà fino a quando il messaggio non sarà elaborato dal server o il timeout non sarà scaduto.
 - **Timeout:** valore in millisecondi se await è true (per impostazione predefinita 10000).
 - **RaiseTimeoutException:** se il timeout viene superato, viene generata un'eccezione (per impostazione predefinita true).

```
oAMQP1.SendMessage("MySession", "MySenderLink", "My first AMQP Message");
```

Invio messaggio in attesa

Per impostazione predefinita, il metodo **SendMessage** è asincrono quando si invia un messaggio non confermato; impostando la proprietà **Await** a true, il client attenderà fino a ricevere una conferma dal server che il messaggio è stato elaborato.

```
public static void SendMessageAwait(string aSession, string aSenderLink, string aText)
{
    TsgcAMQP1MethodOptions_SendMessageAck oOptions = new TsgcAMQP1MethodOptions_SendMessageAck();
    try
    {
        oOptions.Settled = false;
        oOptions.Await = true;
        AMQP1.SendMessage(aSession, aSenderLink, aText, "message-id", oOptions);
    }
    finally
    {
        oOptions.Dispose();
    }
}
```

Eventi

Quando si invia un messaggio, ci sono 2 eventi che possono essere utilizzati per sapere quando il messaggio è stato inviato e se il messaggio è stato elaborato dal server (quando si invia non confermato).

- **OnAMQPMessageSent:** questo evento viene chiamato dopo che il messaggio è stato inviato al server. Quando si chiama il metodo `SendMessage`, il messaggio viene memorizzato in una coda interna ed elaborato da un thread secondario, quindi dopo l'invio del messaggio questo evento viene chiamato.
- **OnAMQPMessageSentAck:** questo evento viene chiamato quando il client riceve la conferma che il messaggio è stato elaborato dal Server AMQP1.

```
public void OnAMQPMessageSentAck(object Sender, TsgcAMQP1Session aSession,
    TsgcAMQP1SenderLink aLink, string aMessageId, TsgcAMQP1FrameDeliveryStates aDeliveryState,
    TsgcAMQP1FrameDisposition aDisposition)
{
    string vMessageId = aMessageId;
    switch (aDeliveryState)
    {
        case TsgcAMQP1FrameDeliveryStates.amqp1fdtsAccepted:
            MessageBox.Show("#msg-accepted: " + vMessageId);
            break;
        case TsgcAMQP1FrameDeliveryStates.amqp1fdtsRejected:
            MessageBox.Show("#msg-rejected: " + vMessageId + " " +
                ((TsgcAMQP1FrameRejected)aDisposition.State).Error.Condition + " " +
                ((TsgcAMQP1FrameRejected)aDisposition.State).Error.Description);
            break;
        case TsgcAMQP1FrameDeliveryStates.amqp1fdtsReleased:
            MessageBox.Show("#msg-released: " + vMessageId);
            break;
        case TsgcAMQP1FrameDeliveryStates.amqp1fdtsModified:
            MessageBox.Show("#msg-modified: " + vMessageId + " " +
                ((TsgcAMQP1FrameModified)aDisposition.State).MessageAnnotations);
            break;
        case TsgcAMQP1FrameDeliveryStates.amqp1fdtsReceived:
            MessageBox.Show("#msg-received: " + vMessageId);
            break;
    }
}
```

AMQP1 | Lettura Messaggio

Ogni volta che viene ricevuto un nuovo messaggio, viene generato l'evento **OnAMQPMessage**.

L'istanza `TsgcAMQP1Message` contiene il messaggio ricevuto. È possibile accedere al messaggio di testo utilizzando la proprietà **aMessage.ApplicationData.AMQPValue.Value**.

Per specificare il risultato della consegna, utilizzare il parametro **DeliveryState**. Per impostazione predefinita, tutti i messaggi hanno lo stato accettato, ma è possibile impostare uno dei seguenti valori:

- **amqp1mdtsAccepted**: Il messaggio è stato elaborato con successo.
- **amqp1mdtsRejected**: Il messaggio non è stato elaborato correttamente. Impostare l'errore tramite la proprietà `DeliveryState.Rejected`.
- **amqp1mdtsReleased**: Il messaggio non è stato e non sarà elaborato.
- **amqp1mdtsModified**: Uguale ad `amqp1mdtsReleased`, ma è possibile aggiungere dati aggiuntivi utilizzando la proprietà `DeliveryState.Modified`.

```
public void OnAMQPMessage(object Sender, TsgcAMQP1Session aSession,
    TsgcAMQP1ReceiverLink aLink, TsgcAMQP1Message aMessage,
    ref TsgcAMQP1MessageDeliveryState DeliveryState)
{
    if (aMessage.ApplicationData.AMQPValue.Value == "xxx")
    {
        DeliveryState.State = TsgcAMQP1MessageDeliveryStates.amqp1mdtsRejected;
        DeliveryState.Rejected.Error.Condition = "amqp-error-processing";
        DeliveryState.Rejected.Error.Description = "Value received was not the expected.";
    }
    else
    {
        DeliveryState.State = TsgcAMQP1MessageDeliveryStates.amqp1mdtsAccepted;
    }
}
```

Protocollo STOMP

STOMP è il Simple (o Streaming) Text Orientated Messaging Protocol. STOMP fornisce un formato wire interoperabile in modo che i client STOMP possano comunicare con qualsiasi message broker STOMP per garantire un'interoperabilità di messaggistica semplice e diffusa tra molti linguaggi, piattaforme e broker.

I nostri componenti client STOMP supportano le seguenti versioni STOMP: 1.0, 1.1 e 1.2.

Componenti

TsgcWSPClient_STOMP: client generico del protocollo STOMP, consente di connettersi a qualsiasi server STOMP.

TsgcWSPClient_STOMP_RabbitMQ: Client STOMP per broker RabbitMQ.

TsgcWSPClient_STOMP_ActiveMQ: Client STOMP per il broker ActiveMQ.

TsgcWSPClient_STOMP

Questo è il componente Client Protocol STOMP.

Introduzione

Questo è il componente Client Protocol STOMP. È necessario inserire questo componente nel form e selezionare un componente [TsgcWebSocketClient](#) tramite la proprietà Client.

Riferimento

- Properties — 8 proprietà disponibili nel componente.
- Metodi — 11 metodi pubblici.
- Eventi — 6 eventi gestibili.

TsgcWSPClient_STOMP_RabbitMQ

Questo è il componente Client Protocol STOMP per il broker RabbitMQ.

Introduzione

Questo è il componente del protocollo client STOMP per il broker RabbitMQ. È necessario trascinare questo componente nel form e selezionare un componente [TsgcWebSocketClient](#) tramite la proprietà Client.

Destinazioni

La specifica STOMP non prescrive i tipi di destinazioni che un broker deve supportare; il valore dell'intestazione destination nei frame SEND e MESSAGE è specifico del broker. L'adattatore STOMP di RabbitMQ supporta diversi tipi di destinazione:

- **Topic:** SEND e SUBSCRIBE a topic transitori e durevoli.
- **Queue:** SEND e SUBSCRIBE alle code gestite dal gateway STOMP.
- **QueueOutside:** SEND e SUBSCRIBE alle code create al di fuori del gateway STOMP.
- **TemporaryQueue:** crea code temporanee (solo negli header reply-to).
- **Exchange:** SEND verso chiavi di routing arbitrarie e SUBSCRIBE a pattern di binding arbitrari.

Riferimento

- Proprietà — 8 proprietà disponibili sul componente.
- Metodi — 25 metodi pubblici.
- Events — 6 eventi gestibili.

TsgcWSPClient_STOMP_ActiveMQ

Questo è il componente Client Protocol STOMP per ActiveMQ Broker.

Introduzione

Questo è il componente del protocollo client STOMP per il broker ActiveMQ. È necessario trascinare questo componente nel form e selezionare un componente [TsgcWebSocketClient](#) utilizzando la proprietà Client.

Destinazioni

La specifica STOMP non prescrive quali tipi di destinazioni un broker deve supportare; il valore dell'intestazione destination nei frame SEND e MESSAGE è specifico del broker. L'adattatore Active STOMP supporta diversi tipi di destinazione:

- **Topic:** SEND e SUBSCRIBE a topic transitori e durevoli.
- **Queue:** SEND e SUBSCRIBE alle code gestite dal gateway STOMP.

Opzioni di pubblicazione

Si noti che STOMP è progettato per essere il più semplice possibile - in modo che qualsiasi linguaggio/piattaforma di scripting possa scambiare messaggi con qualsiasi altro con il minimo sforzo. STOMP consente header collegabili su ogni richiesta, come l'invio & e la ricezione di messaggi. ActiveMQ ha diverse estensioni del protocollo Stomp, in modo che la semantica JMS possa essere supportata dai client Stomp. Un produttore JMS OpenWire può inviare messaggi a un consumer Stomp, e un produttore Stomp può inviare messaggi a un consumer JMS OpenWire. E le configurazioni da Stomp a Stomp possono utilizzare il più ricco controllo dei messaggi JMS.

STOMP supporta le seguenti proprietà JMS standard nei messaggi SENT:

- **CorrelationId:** i consumer corretti aggiungono questa intestazione a tutte le risposte che inviano.
- **Expires:** Ora di scadenza del messaggio.
- **JMSXGroupID:** Specifica i gruppi di messaggi.
- **JMSXGroupSeq:** Intestazione opzionale che specifica il numero di sequenza nei Gruppi di Messaggi.
- **Persistent:** Indica se il messaggio è persistente o meno.
- **Priority:** Priorità del messaggio.
- **ReplyTo:** Destinazione a cui inviare le risposte.
- **MsgType:** Tipo del messaggio.

Riferimento

- Proprietà — 9 proprietà disponibili sul componente.
- Metodi — 16 metodi pubblici.
- Events — 6 eventi che è possibile gestire.

Protocol AppRTC

WebRTC (Web Real-Time Communication) è una definizione API in fase di elaborazione da parte del World Wide Web Consortium (W3C) per abilitare applicazioni browser-to-browser per chiamate vocali, videochiamate e condivisione di file P2P senza plugin. RTC in WebRTC sta per Real-Time Communications, una tecnologia che abilita lo streaming audio/video e la condivisione di dati tra client browser (peer). Come insieme di standard, WebRTC fornisce a qualsiasi browser la capacità di condividere dati applicativi ed effettuare videoconferenze peer to peer, senza la necessità di installare plug-in o software di terze parti.

I componenti WebRTC sono accessibili tramite le API JavaScript. Attualmente sono in fase di sviluppo la Network Stream API, che rappresenta uno stream di dati audio o video, e la PeerConnection API, che consente a due o più utenti di comunicare da browser a browser. È in sviluppo anche una DataChannel API che abilita la comunicazione di altri tipi di dati per giochi in tempo reale, chat testuale, trasferimento di file e altro ancora.

[appr.tc](#) è un'applicazione demo WebRTC sviluppata da Google e Mozilla; consente a due browser di "comunicare" tra di loro tramite le API WebRTC.

Componenti

[TsgcWSPServer_AppRTC](#): Componente VCL Server Protocol AppRTC.

TsgcWSPServer_AppRTC

Questo è il componente Server Protocol AppRTC.

Introduzione

Questo è il componente Server Protocol AppRTC. È necessario trascinare questo componente nel form e selezionare un componente [TsgcWebSocketServer](#) tramite la proprietà Server.

Parametri

- **IceServers:** qui è possibile configurare i server turn/stun per le connessioni WebRTC.
- **RoomLink:** URL base per accedere alla stanza. Esempio: <https://mydemo.com/r/>
- **WebSocketURL:** URL del server WebSocket. Esempio: <wss://mydemo.com>

Il protocollo WebRTC richiede un server STUN/TURN; le demo utilizzano server STUN/TURN pubblici a scopo di test. Per un sistema in produzione, è necessario un server STUN/TURN dedicato.

Gli utenti registrati possono scaricare i binari compilati del **server Coturn per Windows**. Per saperne di più su [COTURN STUN/TURN](#).

Configurazione IceServers

Se sta eseguendo il Suo server STUN/TURN sul seguente indirizzo IP: 51.122.4.88 e questo è in ascolto sulla porta 3478. L'utente con cui connettersi è "apprtc" e la credenziale è "secret". Configuri gli IceServers come segue:

```
{
  "lifetimeDuration": "86400s",
  "iceServers": [{
    "urls": "stun:51.122.4.88:3478",
    "username": "apprtc",
    "credential": "secret"
  }, {
    "urls": "turn:51.122.4.88:3478",
    "username": "apprtc",
    "credential": "secret"
  }],
  "blockStatus": "NOT_BLOCKED",
  "iceTransportPolicy": "all"
}
```

Riferimento

- Proprietà — 5 proprietà disponibili sul componente.
- Metodi — 0 metodi pubblici.
- Eventi — 11 eventi gestibili.

Protocollo WebRTC

WebRTC (Web Real-Time Communication) è una definizione API in fase di elaborazione da parte del World Wide Web Consortium (W3C) per abilitare applicazioni browser-to-browser per chiamate vocali, videochiamate e condivisione di file P2P senza plugin. RTC in WebRTC sta per Real-Time Communications, una tecnologia che abilita lo streaming audio/video e la condivisione di dati tra client browser (peer). Come insieme di standard, WebRTC fornisce a qualsiasi browser la capacità di condividere dati applicativi ed effettuare videoconferenze peer to peer, senza la necessità di installare plug-in o software di terze parti.

I componenti WebRTC sono accessibili tramite le API JavaScript. Attualmente sono in fase di sviluppo la Network Stream API, che rappresenta uno stream di dati audio o video, e la PeerConnection API, che consente a due o più utenti di comunicare da browser a browser. È in sviluppo anche una DataChannel API che abilita la comunicazione di altri tipi di dati per giochi in tempo reale, chat testuale, trasferimento di file e altro ancora.

Componenti

[TsgcWSPServer_WebRTC](#): Componente VCL Server Protocol WebRTC.

Parametri

- **IceServers**: qui è possibile configurare server turn/stun per le connessioni WebRTC. Per impostazione predefinita vengono utilizzati i seguenti server STUN pubblici

```
{"iceServers": [{"url": "stun:stun.l.google.com:19302"}]}
```

Browser Test

Se desidera testare questo protocollo con il suo browser web preferito, digiti questo url (deve definire il suo host e porta personalizzati)

```
http://host:port/webrtc.esegece.com.html
```

TsgcWSPServer_WebRTC

Questo è il componente Server Protocol WebRTC.

Introduzione

Questo è il componente protocollo WebRTC lato server. È necessario inserire questo componente nel form e selezionare un componente [TsgcWebSocketServer](#) tramite la proprietà Server.

Il protocollo WebRTC richiede un server STUN/TURN; le demo utilizzano server STUN/TURN pubblici a scopo di test. Per un sistema in produzione, è necessario un server STUN/TURN dedicato.

Gli utenti registrati possono scaricare i binari compilati del **server Coturn per Windows**. Per saperne di più su [COTURN STUN/TURN](#).

Riferimento

- Proprietà — 5 proprietà disponibili sul componente.
- Metodi — 2 metodi pubblici.
- Eventi — 11 eventi gestibili.

Protocollo WebRTC Javascript

Qui può trovare i metodi disponibili, deve sostituire le variabili {%host%} e {%port%} secondo necessità, esempio: se ha configurato il Suo server sgcWebSocket per ascoltare sulla porta 80 sul sito www.example.com deve configurare:

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
<script src="http://www.example.com:80/webrtc.esegece.com.js"></script>
```

Apri Connessione

Quando viene aperta una connessione WebSocket, il browser richiede l'accesso alla fotocamera e al microfono locali; è necessario consentire tale accesso.

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/webrtc.esegece.com.js"></script>
<script>
  var socket = new sgcws_webrtc('ws://{%host%}:{%port%}');
</script>
```

Apri Canale WebRTC

Quando un browser ha accesso alla telecamera e al microfono locali, viene attivato l'evento 'sgcmediastart' e poi è possibile tentare di connettersi a un altro client usando la procedura webrtc_connect

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/webrtc.esegece.com.js"></script>
<script>
  var socket = new sgcws_webrtc('ws://{%host%}:{%port%}');
  socket.on('sgcmediastart', function(event)
  {
    socket.webrtc_connect('custom channel');
  }
</script>
```

Chiudi il canale WebRTC

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/webrtc.esegece.com.js"></script>
<script>
  socket.webrtc_disconnect('custom channel');
</script>
```

Protocollo WAMP

WAMP è un sottoprotocollo WebSocket aperto che fornisce due modelli di messaggistica asincrona: RPC e Pub-Sub.

Tecnicamente, WAMP è un sottoprotocollo WebSocket ufficialmente registrato (eseguito su WebSocket) che utilizza JSON come formato di serializzazione dei messaggi.

Cos'è RPC?

Remote Procedure Call (RPC) è un pattern di messaggistica che coinvolge peer con due ruoli: client e server.

Un server fornisce metodi o procedure da chiamare sotto endpoint ben noti.

Un client chiama metodi o procedure remote fornendo l'endpoint del metodo o della procedura e gli eventuali argomenti per la chiamata.

Il server eseguirà il metodo o la procedura utilizzando gli argomenti forniti alla chiamata e restituirà il risultato della chiamata al client.

Cos'è PubSub?

Publish & Subscribe (PubSub) è un pattern di messaggistica che coinvolge peer con tre ruoli: publisher, subscriber e broker.

Un publisher invia (pubblica) un evento fornendo un argomento (detto anche canale) come indirizzo astratto, non un peer specifico.

Un subscriber riceve eventi fornendo prima i topic (detti anche canali) a cui è interessato. Successivamente riceverà tutti gli eventi pubblicati su quel topic.

Il broker si colloca tra i publisher e i subscriber e media i messaggi che i publisher pubblicano verso i subscriber.

Un broker mantiene elenchi di subscriber per ogni topic in modo da poter distribuire gli eventi appena pubblicati ai subscriber appropriati.

Un broker può anche inviare eventi di propria iniziativa, ad esempio quando il broker funge anche da server RPC e un metodo eseguito sul server deve attivare un evento PubSub.

In sintesi, PubSub disaccoppia i publisher e i ricevitori tramite un intermediario, il broker.

Componenti

[TsgcWSPServer_WAMP](#): Componente VCL Protocollo Server WAMP.

[TsgcWSPClient_WAMP](#): Componente VCL Client Protocol WAMP.

[Componente Javascript](#): Riferimento Javascript per il Client.

Utilizzi più comuni

- **RPC**
 - [RPC Semplice](#)
 - [RPC Risultati Progressivi](#)
- **PubSub**
 - [Subscribers](#)
 - [Publishers](#)

Browser Test

Per testare questo protocollo con il proprio browser Web preferito, digitare questo URL (è necessario definire il proprio host e porta personalizzati)

`http://host:port/wamp.esegece.com.html`

TsgcWSPServer_WAMP

Questo è il componente Server Protocol WAMP.

Introduzione

Questo è il componente server Protocol WAMP. È necessario trascinare questo componente nel form e selezionare un componente [TsgcWebSocketServer](#) utilizzando la proprietà Server.

Riferimento

- Proprietà — 4 proprietà disponibili sul componente.
- Methods — 4 metodi pubblici.
- Eventi — 14 eventi gestibili.

TsgcWSPClient_WAMP

Questo è il Componente Client Protocol WAMP.

Introduzione

Questo è il Componente Client Protocol WAMP. Occorre rilasciare questo componente nel form e selezionare un componente [TsgcWebSocketClient](#) tramite la proprietà Client.

Riferimento

- Proprietà — 3 proprietà disponibili sul componente.
- Methods — 7 metodi pubblici.
- Eventi — 13 eventi che è possibile gestire.

Protocollo WAMP Javascript

Qui può trovare i metodi disponibili, deve sostituire le variabili {%host%} e {%port%} secondo necessità, esempio: se ha configurato il Suo server sgcWebSocket per ascoltare sulla porta 80 sul sito www.example.com deve configurare:

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
<script src="http://www.example.com:80/wamp.esegece.com.js"></script>
```

Apri Connessione

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  var socket = new sgcws_wamp('ws://{%host%}:{%port%}');
</script>
```

Invia nuovo prefisso

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  var socket = new sgcws_wamp('ws://{%host%}:{%port%}');
  socket.prefix('sgc', 'http://www.esegece.com');
</script>
```

Richiesta RPC (Remote Procedure Call)

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  var socket = new sgcws_wamp('ws://{%host%}:{%port%}');
  socket.call('', 'sgc:CallTest', '20')
</script>
```

Sottoscrizione a un TopicURI

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  var socket = new sgcws_wamp('ws://{%host%}:{%port%}');
  socket.subscribe('sgc:test')
</script>
```


Annula l'iscrizione da un TopicURI

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  var socket = new sgcws_wamp('ws://{%host%}:{%port%}');
  socket.unsubscribe('sgc:test')
</script>
```

Pubblica messaggio

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  var socket = new sgcws_wamp('ws://{%host%}:{%port%}');
  socket.publish('sgc:channel', 'Test Message', [], []);
</script>
```

Mostra avviso con messaggio ricevuto

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  var socket = new sgcws('ws://{%host%}:{%port%}');
  socket.on('sgcmessage', function(event)
  {
    alert(event.message);
  }
</script>
```

Mostra avviso su OnCallResult o OnCallError

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  var socket = new sgcws_wamp('ws://{%host%}:{%port%}');
  socket.on('wampcallresult', function(event)
  {
    alert('call result: ' + event.CallId + ' - ' + event.CallResult);
  }
  socket.on('wampcallprogressresult', function(event)
  {
    alert('call progress result: ' + event.CallId + ' - ' + event.CallResult);
  }
  socket.on('wampcallerror', function(event)
  {
    alert('call error: ' + event.CallId + ' - ' + event.ErrorURI + ' - ' + event.ErrorDesc +
      ' - ' + event.ErrorDetails);
  }
</script>
```

Mostra avviso OnEvent

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
```

```
<script>
  var socket = new sgcws_wamp('ws://{%host%}:{%port%}');
  socket.on('wampevent', function(event)
  {
    alert('call result: ' + event.TopicURI + ' - ' + event.Event);
  }
</script>
```

Mostra avviso negli eventi OnConnect, OnDisconnect e OnError

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  var socket = new sgcws_wamp('ws://{%host%}:{%port%}');
  socket.on('open', function(event)
  {
    alert('sgcWebSocket Open!');
  });
  socket.on('close', function(event)
  {
    alert('sgcWebSocket Closed!');
  });
  socket.on('error', function(event)
  {
    alert('sgcWebSocket Error: ' + event.message);
  });
</script>
```

Chiudi connessione

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  socket.close();
</script>
```

Ottieni lo stato della connessione

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  socket.state();
</script>
```

WAMP | Subscribers

Un subscriber riceve eventi fornendo prima i topic (detti anche canali) a cui è interessato. Successivamente riceverà tutti gli eventi pubblicati su quel topic.

Per ricevere eventi da un topic, il subscriber deve prima sottoscrivere a quel topic.

WAMP Client

```
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");
```

WAMP Server

```
void OnSubscriptionEvent(TsgcWSConnection Connection, string Subscription)
{
    MessageBox.Show("Subscribed: " + Subscription);
}

oServer = new TsgcWebSocketServer();
oServer.Port = 80;
oServerWAMP = new TsgcWSPServer_WAMP();
oServerWAMP.OnSubscription = OnSubscriptionEvent;
oServerWAMP.Server = oServer;
oServerWAMP.Active = true;
```

WAMP | Publisher

Un publisher invia (pubblica) un evento fornendo un topic (detto anche canale) come indirizzo astratto, non un peer specifico. È sufficiente chiamare il metodo Publish e passare come argomenti il nome del topic e il messaggio che si desidera inviare. Il messaggio verrà recapitato a tutti i sottoscrittori di quel topic. Come nota, non è necessario iscriversi a un topic per pubblicare messaggi su quel topic.

Non è necessario configurare nulla lato server, poiché i messaggi vengono trasmessi automaticamente ai client quando viene ricevuto un messaggio di publish.

WAMP Client

```
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");
```

WAMP | RPC Semplice

L'uso più comune del componente WAMP è che un client richieda un metodo al server, e il server invii una risposta al client. Il client può inviare solo il nome del metodo e/o passare alcuni parametri richiesti dal server per calcolare il risultato. Il server elabora le richieste e, in caso di successo, invia una risposta al client con il risultato. In caso di errore, il server invia una risposta di errore al client.

Come può vedere, c'è solo Una richiesta e Una risposta (con esito positivo o negativo).

Esempio: il server ha un metodo chiamato **GetTime**, quindi ogni volta che un client richiede questo metodo, il server restituisce l'ora del server.

WAMP Server

```
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;
```

WAMP Client

```
void OnCallResultClient(TsgcWSConnection Connection, string CallId, string Result);
{
    MessageBox.Show(Result);
}
void OnCallErrorClient(TsgcWSConnection Connection, string Error)
{
    MessageBox.Show(Error);
}
oClient = new TsgcWebSocketClient();
oClient.Host = "127.0.0.1";
oClient.Port = 80;
oClientWAMP = new TsgcWSPClient_WAMP();
oClientWAMP.OnCallResult = OnCallResultClient;
oClientWAMP.OnCallError = OnCallErrorClient;
oClientWAMP.Client = oClient;
oClient.Active = true;
// After client has connected, request GetTime from server
oClientWAMP.Call("GetTime");
```

WAMP | Risultati di avanzamento RPC

A volte le chiamate di procedura remota richiedono più di un risultato per completare le richieste; per impostazione predefinita il protocollo WAMP 1.0 non consente risultati parziali in una chiamata, questa è una funzionalità esclusiva della libreria `sgcWebSockets`.

Il flusso è molto simile a un semplice RPC, ma qui ci sono 1 o più risultati parziali prima che `CallResult` venga chiamato per terminare il processo.

Fondamentalmente, un client richiede una procedura al server, e il server può inviare un risultato o un errore. Se invia un risultato, questo può essere il risultato finale oppure deve inviare ulteriori risultati in seguito. Se è il risultato finale, chiamerà il metodo **CallResult** e il processo sarà terminato. Se ci sono altri risultati da inviare, chiamerà il metodo **CallProgressResult**.

Esempio: il client richiede al server un metodo per ricevere ogni secondo l'orario del server e fermarsi dopo 20 messaggi.

WAMP Server

```
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", 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;
```

WAMP Client

```
void OnCallResultClient(TsgcWSConnection Connection, string CallId, string Result);
{
    MessageBox.Show(Result);
}

void OnCallProgressResultClient(TsgcWSConnection Connection, string CallId, string Result);
{
    MessageBox.Show(Result);
}

void OnCallErrorClient(TsgcWSConnection Connection, string Error)
{
    MessageBox.Show(Error);
}

oClient = new TsgcWebSocketClient();
oClient.Host = "127.0.0.1";
oClient.Port = 80;
```

```
oClientWAMP = new TsgcWSPClient_WAMP();  
oClientWAMP.OnCallResult = OnCallResultClient;  
oClientWAMP.OnCallProgressResult = OnCallProgressResultClient;  
oClientWAMP.OnCallError = OnCallErrorClient;  
oClientWAMP.Client = oClient;  
oClient.Active = true;  
// After client has connected, request GetTime from server  
oClientWAMP.Call("GetProgressTime");
```

Protocollo WAMP 2

WAMP fornisce l'Unified Application Routing in un protocollo WebSocket aperto che funziona con linguaggi differenti.

Utilizzando WAMP è possibile costruire sistemi distribuiti composti da componenti applicativi debolmente accoppiati che comunicano in tempo reale (soft).

Alla base, WAMP offre due pattern di comunicazione per i componenti dell'applicazione per comunicare tra loro:

- Pubblica & Sottoscrivi (PubSub)
- Remote Procedure Calls (RPC)

WAMP è facile da usare, semplice da implementare e basato su moderni standard Web: WebSocket, JSON e URI.

Componenti

[TsgcWSPClient_WAMP2](#): Componente VCL client del protocollo WAMP2.

TsgcWSPClient_WAMP2

Questo è il componente client del protocollo WAMP2.

Introduzione

Questo è il componente Client Protocol WAMP2. È necessario trascinare questo componente nel form e selezionare un componente [TsgcWebSocketClient](#) tramite la proprietà Client.

Metodi di sessione

- **ABORT:** Sia il Router che il Client possono interrompere l'apertura di una sessione WAMP inviando un messaggio ABORT.

Reason DEVE essere un URI.

Details DEVE essere un dizionario che consente di fornire informazioni di chiusura aggiuntive e opzionali (vedere di seguito).

Non è attesa alcuna risposta a un messaggio ABORT.

- **GOODBYE:** Una sessione WAMP inizia il suo ciclo di vita con il Router che invia un messaggio WELCOME al Client e termina quando il trasporto sottostante scompare o quando la sessione WAMP viene chiusa esplicitamente tramite un messaggio GOODBYE inviato da un Peer e un messaggio GOODBYE inviato dall'altro Peer in risposta.

Reason DEVE essere un URI.

Details DEVE essere un dizionario che consente di fornire informazioni di chiusura aggiuntive e opzionali.

Metodi Publish/Subscribe

- **PUBLISH:** Quando un Publisher richiede di pubblicare un evento su un argomento, invia un messaggio PUBLISH a un Broker:

Request è un ID casuale ed effimero scelto dal Publisher e utilizzato per correlare la risposta del Broker con la richiesta.

Options è un dizionario che consente di fornire ulteriori dettagli della richiesta di pubblicazione in modo estensibile. Questo viene descritto più avanti.

Topic è l'argomento su cui viene pubblicato.

Arguments è un elenco di elementi del payload dell'evento a livello applicativo. L'elenco può avere lunghezza zero.

ArgumentsKw è un dizionario opzionale contenente il payload dell'evento a livello applicativo, fornito come argomenti per parola chiave. Il dizionario può essere vuoto.

Se il Broker è in grado di soddisfare e consentire la pubblicazione, invierà l'evento a tutti i Sottoscrittori attuali dell'argomento dell'evento pubblicato.

Per impostazione predefinita, le pubblicazioni non sono riconosciute e il Broker non risponde, indipendentemente dal fatto che la pubblicazione abbia avuto successo o meno.

- **SUBSCRIBE:** Un Subscriber comunica il suo interesse per un topic a un Broker inviando un messaggio SUBSCRIBE:

Request DEVE essere un ID casuale ed effimero scelto dal Subscriber e utilizzato per correlare la risposta del Broker con la richiesta.

Options DEVE essere un dizionario che consente di fornire dettagli aggiuntivi sulla richiesta di sottoscrizione in modo estensibile.

Topic è il topic a cui il Subscriber desidera iscriversi e DEVE essere un URI.

- **UNSUBSCRIBE:** Quando un Subscriber non è più interessato a ricevere eventi per una sottoscrizione, invia un messaggio UNSUBSCRIBE

Request DEVE essere un ID casuale ed effimero scelto dal Subscriber e utilizzato per correlare la risposta del Broker con la richiesta.

SUBSCRIBED.Subscription DEVE essere l'ID dell'abbonamento da cui annullare l'iscrizione, originariamente assegnato dal Broker al Subscriber.

Metodi RPC

- **CALL:** Quando un Caller desidera chiamare una procedura remota, invia un messaggio CALL a un Dealer:

Request è un ID casuale ed effimero scelto dal Caller e utilizzato per correlare la risposta del Dealer con la richiesta.

Options è un dizionario che consente di fornire dettagli aggiuntivi della richiesta di chiamata in modo estensibile. Questo argomento è descritto ulteriormente di seguito.

Procedure è l'URI della procedura da chiamare.

Arguments è un elenco di argomenti di chiamata posizionali (ciascuno di tipo arbitrario). L'elenco può avere lunghezza zero.

ArgumentsKw è un dizionario di argomenti di chiamata con parole chiave (ciascuno di tipo arbitrario). Il dizionario può essere vuoto.

- **REGISTERCALL:** Un Callee annuncia la disponibilità di un endpoint che implementa una procedura con un Dealer inviando un messaggio REGISTER:

Request è un ID casuale ed effimero scelto dal Callee e utilizzato per correlare la risposta del Dealer con la richiesta.

Options è un dizionario che consente di fornire dettagli aggiuntivi della richiesta di registrazione in modo estensibile. Questo è descritto più in dettaglio di seguito.

Procedure è la procedura che il Callee vuole registrare

- **UNREGISTERCALL:** Quando un Callee non è più disposto a fornire un'implementazione della procedura registrata, invia un messaggio UNREGISTER al Dealer:

Request è un ID casuale ed effimero scelto dal Callee e utilizzato per correlare la risposta del Dealer con la richiesta.

REGISTERED.Registration è l'ID della registrazione da revocare, originariamente assegnato dal Dealer al Callee.

- **INVOCATION:** Se il Dealer è in grado di soddisfare (mediare) la chiamata e la consente, invia un messaggio INVOCATION al rispettivo Callee che implementa la procedura:

Request è un ID casuale ed effimero scelto dal Dealer e utilizzato per correlare la risposta del Callee con la richiesta.

REGISTERED.Registration è l'ID di registrazione con cui la procedura è stata registrata presso il Dealer.

Details è un dizionario che consente di fornire dettagli aggiuntivi sulla richiesta di invocazione in modo estensibile. Questo viene descritto più avanti.

CALL.Arguments è l'elenco originale degli argomenti posizionali della chiamata fornito dal Caller.

CALL.ArgumentsKw è il dizionario originale degli argomenti di chiamata con parola chiave forniti dal Caller.

- **YIELD:** Se il Callee è in grado di elaborare con successo e terminare l'esecuzione della chiamata, risponde inviando un messaggio YIELD al Dealer:

INVOCATION.Request è l'ID della richiesta di invocazione originale.

Options è un dizionario che consente di fornire opzioni aggiuntive.

Arguments è un elenco di elementi di risultato posizionali (ciascuno di tipo arbitrario). L'elenco può avere lunghezza zero.

ArgumentsKw è un dizionario di elementi risultato con chiave (ciascuno di tipo arbitrario). Il dizionario può essere vuoto.

Riferimento

- Proprietà — 3 proprietà disponibili nel componente.
- Metodi — 11 metodi pubblici.
- Eventi — 20 eventi gestibili.

Protocollo predefinito

Questo è il sotto-protocollo predefinito implementato utilizzando messaggi "JSONRPC 2.0", ogni volta che si invia un messaggio utilizzando questo protocollo viene creato un oggetto JSON con le seguenti proprietà:

jsonrpc: Una stringa che specifica la versione del protocollo JSON-RPC. DEVE essere esattamente "2.0".

method: Una stringa contenente il nome del metodo da invocare. I nomi dei metodi che iniziano con la parola rpc seguita da un punto (U+002E o ASCII 46) sono riservati ai metodi interni rpc e alle estensioni e NON DEVONO essere utilizzati per altro.

params: Un valore strutturato che contiene i valori dei parametri da utilizzare durante l'invocazione del metodo. Questo membro PUÒ essere omissivo.

id: Un identificatore stabilito dal Client che DEVE contenere un valore String, Number, o NULL se incluso. Se non è incluso, si assume che sia una notifica. Il valore NON DOVREBBE normalmente essere Null [1] e i Number NON DOVREBBERO contenere parti frazionarie [2]

Esempio di oggetto JSON:

```
{"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "id": 1}
```

Funzionalità

- **Publish/subscribe** modello di messaggistica per fornire distribuzione messaggi uno-a-molti e disaccoppiamento delle applicazioni. Supporta caratteri jolly, quindi è possibile iscriversi a una gerarchia di canali. Esempio: per iscriversi a tutti i canali che iniziano con 'news', chiamare `Subscribe('news*')`.
- Un trasporto di messaggistica che è **agnostico** rispetto al contenuto del payload
- **Conferma** dei messaggi inviati.
- Supporta **messaggi transazionali** tramite transazioni locali del server. Quando il client esegue il commit della transazione, il server elabora tutti i messaggi in coda. Se il client esegue il rollback della transazione, tutti i messaggi vengono eliminati.
- Implementa **QoS** (Quality of Service) per la consegna dei messaggi.

Componenti

[TsgcWSPClient_sgc](#): Componente VCL Server Protocol Default.

[TsgcWSPClient_sgc](#): Client Componente VCL predefinito del protocollo.

[Componente Javascript](#): Riferimento Client Javascript.

Browser Test

Per testare questo protocollo con il proprio browser Web preferito, digitare questo URL (è necessario definire il proprio host e porta personalizzati)

`http://host:port/esegece.com.html`

TsgcWSPServer_sgc

Questo è il componente predefinito del protocollo server.

Introduzione

Questo è il componente predefinito del protocollo server. È necessario trascinare questo componente nel form e selezionare un componente [TsgcWebSocketServer](#) tramite la proprietà Server.

Riferimento

- Proprietà — 6 proprietà disponibili sul componente.
- Metodi — 6 metodi pubblici.
- Eventi — 16 eventi che è possibile gestire.

TsgcWSPClient_sgc

Questo è il componente predefinito del protocollo client.

Introduzione

Questo è il componente predefinito del protocollo client. È necessario inserire questo componente nel form e selezionare un componente [TsgcWebSocketClient](#) tramite la proprietà Client.

Riferimento

- Proprietà — 5 proprietà disponibili sul componente.
- Metodi — 11 metodi pubblici.
- Eventi — 15 eventi gestibili.

TsgclWWSPClient_sgc

Questo è il Componente Predefinito del Protocollo Client Intraweb.

Introduzione

Questo è il componente predefinito del protocollo client IntraWeb. È necessario trascinare questo componente nel form e selezionare un componente [TsgclWWebSocketClient](#) usando la proprietà Client.

Riferimento

- Proprietà — 2 proprietà disponibili sul componente.
- Metodi — 3 metodi pubblici.
- Eventi — 6 eventi gestibili.

Protocol Default Javascript

Protocollo JavaScript predefinito: sgcWebSockets utilizza i file **sgcWebSocket.js** e **esegece.com.js**.

Qui può trovare i metodi disponibili, deve sostituire le variabili `{%host%}` e `{%port%}` secondo necessità, esempio: se ha configurato il Suo server sgcWebSocket per ascoltare sulla porta 80 sul sito `www.example.com` deve configurare:

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
<script src="http://www.example.com:80/esegece.com.js"></script>
```

Apri Connessione

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcws('ws://{%host%}:{%port%}');
</script>
```

Invia messaggio

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcws('ws://{%host%}:{%port%}');
  socket.send('Hello sgcWebSockets!');
</script>
```

Mostra avviso con messaggio ricevuto

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcws('ws://{%host%}:{%port%}');
  socket.on('sgcmessage', function(event)
  {
    alert(event.message);
  }
</script>
```

Pubblica messaggio nel canale di test

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcws('ws://{%host%}:{%port%}');
  socket.publish('Hello sgcWebSockets!', 'test');
</script>
```

Mostra avviso con messaggio evento ricevuto

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcs('ws://{%host%}:{%port%}');
  socket.on('sgcevent', function(event)
  {
    alert('channel:' + event.channel + '. message: ' + event.message);
  }
</script>
```

Chiama RPC

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcs('ws://{%host%}:{%port%}');
  var params = {param:10};
  socket.rpc(GUID(), 'test', JSON.stringify(params));
</script>
```

Gestisci risposta RPC

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcs('ws://{%host%}:{%port%}');
  socket.on('sgcrpcresult', function(event)
  {
    alert('result:' + event.result);
  }
  socket.on('sgcrpcerror', function(event)
  {
    alert('error:' + event.code + ' ' + event.message);
  }
</script>
```

Notifica Chiamata

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcs('ws://{%host%}:{%port%}');
  var params = {param:10};
  socket.notify('test', JSON.stringify(params));
</script>
```

Send Messages in a Transaction

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcs('ws://{%host%}:{%port%}');

  socket.starttransaction('sgc:test');
  socket.publish('Message1', 'sgc:test');
  socket.publish('Message2', 'sgc:test');
```

```
socket.publish('Message3', 'sgc:test');
socket.commit('sgc:test');
</script>
```

Mostra avviso su OnSubscribe o OnUnSubscribe a un canale

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcs('ws://{%host%}:{%port%}');
  socket.on('sgcsubscribe', function(event)
  {
    alert('subscribed: ' + event.channel);
  });
  socket.on('sgcunsubscribe', function(event)
  {
    alert('unsubscribed: ' + event.channel);
  });
</script>
```

Mostra avviso negli eventi OnConnect, OnDisconnect e OnError

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcs('ws://{%host%}:{%port%}');
  socket.on('open', function(event)
  {
    alert('sgcWebSocket Open!');
  });
  socket.on('close', function(event)
  {
    alert('sgcWebSocket Closed!');
  });
  socket.on('error', function(event)
  {
    alert('sgcWebSocket Error: ' + event.message);
  });
</script>
```

Get Session

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcs('ws://{%host%}:{%port%}');
  socket.on('sgcsession', function(event)
  {
    alert(event.guid);
  });
  socket.getSession();
</script>
```

Chiudi connessione

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  socket.close();
</script>
```

Ottieni lo stato della connessione

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcws('ws://{%host%}:{%port%}');

  socket.state();
</script>
```

Imposta QoS

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcws('ws://{%host%}:{%port%}');

  socket.qoslevel1();
  socket.publish('message', 'channel');
</script>
```

Imposta il livello della coda

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcws('ws://{%host%}:{%port%}');

  socket.queuelevel2();
  socket.publish('message1', 'channel1');
  socket.publish('message2', 'channel1');
</script>
```

TsgcWSPServer_Dataset

Questo è il componente Server Protocol Dataset.

Introduzione

Questo è il componente Server Protocol Dataset. È necessario inserire questo componente nel form e selezionare un componente [TsgcWebSocketServer](#) tramite la proprietà Server e selezionare un componente Dataset tramite la proprietà Dataset.

Questo componente eredita da [TsgcWSProtocol_Server_sgc](#) tutti i metodi e le proprietà.

Riferimento

- Proprietà — 13 proprietà disponibili sul componente.
- Metodi — 9 metodi pubblici.
- Eventi — 22 eventi gestibili.

TsgcWSPClient_Dataset

Questo è il componente Dataset del protocollo Client.

Introduzione

Questo è il componente Client Protocol Dataset. Deve posizionare questo componente nella form e selezionare un componente [TsgcWebSocketClient](#) utilizzando la proprietà Client e selezionare un componente Dataset utilizzando la proprietà Dataset.

Questo componente eredita da [TsgcWSProtocol_Client_sgc](#) tutti i metodi e le proprietà.

Riferimento

- Proprietà — 13 proprietà disponibili sul componente.
- Methods — 14 metodi pubblici.
- Eventi — 25 eventi gestibili.

TsgclWWSPClient_Dataset

Questo è il Componente Dataset del Protocollo Client Intraweb.

Introduzione

Questo è il componente Dataset del protocollo client Intraweb. È necessario trascinare questo componente nel form e selezionare un componente [TsgclWWWebSocketClient](#) tramite la proprietà Client e selezionare un componente Dataset tramite la proprietà Dataset.

Questo componente eredita da [TsgclWWSPClient_sgc](#) tutti i metodi e le proprietà.

Riferimento

- Proprietà — 3 proprietà disponibili nel componente.
- Metodi — 6 metodi pubblici.
- Eventi — 9 eventi che Lei può gestire.

File di protocollo

Questo protocollo consente l'invio di file tramite trasporto WebSocket binario. Può gestire file di grandi dimensioni con un basso utilizzo di memoria.

Funzionalità

- **Publish/subscribe** pattern di messaggi per la distribuzione di messaggi uno-a-molti e il disaccoppiamento delle applicazioni.
- **Conferma** dei messaggi inviati.
- Implementa **QoS** (Quality of Service) per la consegna dei file.
- Facoltativamente è possibile richiedere **Autorizzazione** per i file ricevuti.
- **Basso consumo** di memoria.

Componenti

[TsgcWSPServer_Files](#): Componente VCL Server Protocol Files.

[TsgcWSPClient_Files](#): Componente VCL Client Protocol Files.

Classi

[TsgcWSMessageFile](#): l'oggetto che incapsula le informazioni del pacchetto file.

Usi più comuni

- **Inviare file**
 - [Come inviare file al server](#)
 - [Come inviare file ai client](#)
- **File di Grandi Dimensioni**
 - [Come inviare file di grandi dimensioni](#)

TsgcWSPServer_Files

Questo è il componente Server Files Protocol.

Introduzione

Questo è il componente protocollo Files lato server. È necessario inserire questo componente nel form e selezionare un componente [TsgcWebSocketServer](#) tramite la proprietà Server.

Riferimento

- Proprietà — 5 proprietà disponibili nel componente.
- Metodi — 3 metodi pubblici.
- Eventi — 20 eventi che è possibile gestire.

TsgcWSPClient_Files

Questo è il componente Client Files Protocol.

Introduzione

Questo è il componente del protocollo Client Files. È necessario rilasciare questo componente nel form e selezionare un componente [TsgcWebSocketClient](#) tramite la proprietà Client.

Riferimento

- Properties — 5 proprietà disponibili sul componente.
- Metodi — 4 metodi pubblici.
- Eventi — 19 eventi gestibili.

TsgcWSMessageFile

Questo oggetto viene passato come parametro ogni volta che viene generato un evento del protocollo file.

Proprietà

- **BufferSize:** dimensione predefinita del pacchetto.
- **Channel:** se specificato, questo file verrà inviato solo ai client iscritti a un canale specifico.
- **Method:** metodo interno.
- **FileId:** identificatore di un file; è univoco per tutti i file ricevuti/inviati.
- **Data:** dati personalizzati dell'utente. Qui l'utente può impostare qualsiasi testo.
- **FileName:** nome del file.
- **FilePosition:** posizione del file in byte.
- **FileSize:** Dimensione totale del file in byte.
- **Id:** identificatore di un pacchetto; è univoco per ogni pacchetto.
- **QoS:** qualità del servizio del messaggio.
- **Streaming:** per uso interno.
- **Text:** per uso interno.

File di protocollo | Come inviare file al server

Per inviare un file al server, è sufficiente chiamare il metodo **SendFile** del protocollo Files e passare il **FileName** completo come argomento.

Il file ricevuto dal server verrà salvato per impostazione predefinita nella stessa directory dove si trova l'eseguibile del server oppure nel percorso impostato nella proprietà Files.SaveDirectory.

```
// ... Create Server
TsgcWebSocketServer oServer = new TsgcWebSocketServer();
TsgcWSPServer_Files oServer_Files = new TsgcWSPServer_Files();
oServer_Files.Server = oServer;
oServer.Host = "127.0.0.1";
oServer.Port = 8080;

// ... Create Client
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
oClient.URL = "ws://127.0.0.1:8080";

// ... Create Protocol
TsgcWSPClient_Files oClient_Files = new TsgcWSPClient_Files();
oClient_Files.Client = oClient;

// ... Start Server
oServer.Active = true;

// ... Connect client and Send File
if oClient.Connect() then
    oClient_Files.SendFile("c:\Documents\yourfile.txt");
```

File di Protocollo | Come Inviare File ai Client

Per inviare un file a un client, è sufficiente chiamare il metodo **SendFile** del protocollo Files e passare il **Guid** della connessione e il **FileName** completo come argomento. Il Guid della connessione client può essere acquisito nell'evento OnConnect del protocollo Server Files.

Il file ricevuto dal client verrà salvato per impostazione predefinita nella stessa directory dove si trova l'eseguibile del client oppure nel percorso impostato nella proprietà Files.SaveDirectory.

```
// ... capture the guid of the client connection to send later the file
void OnConnectEvent(TsgcWSConnection *Connection)
{
    FGuid = Connection.Guid;
}

// ... Create Server
TsgcWebSocketServer oServer = new TsgcWebSocketServer();
TsgcWSPServer_Files oServer_Files = new TsgcWSPServer_Files();
oServer_Files.Server = oServer;
oServer_Files.OnConnect += OnConnectEvent;
oServer.Host = "127.0.0.1";
oServer.Port = 8080;

// ... Create Client
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
oClient.URL = "ws://127.0.0.1:8080";

// ... Create Protocol
TsgcWSPClient_Files oClient_Files = new TsgcWSPClient_Files();
oClient_Files.Client = oClient;

// ... Start Server
oServer.Active = true;
oClient.Connect();

// ... Send File to the client connected
oServer_Files.SendFile(FGuid, "c:\\Documents\\yourfile.txt");
```

Protocollo Files | Come inviare file di grandi dimensioni

Quando si desidera inviare file di grandi dimensioni al server o al client, ad esempio un file di alcuni gigabyte, si possono riscontrare problemi di memoria nel tentativo di caricare il file intero. Il protocollo Files consente di inviare i file in pacchetti più piccoli che, una volta ricevuti dall'altro peer, vengono riassemblati in un unico file. Utilizzare il parametro **Size** del metodo **SendFile** per impostare la dimensione in byte di ogni singolo pacchetto.

```
// ... Create Server
TsgcWebSocketServer oServer = new TsgcWebSocketServer();
TsgcWSPServer_Files oServer_Files = new TsgcWSPServer_Files();
oServer_Files.Server = oServer;
oServer.Host = "127.0.0.1";
oServer.Port = 8080;

// ... Create Client
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
oClient.URL = "ws://127.0.0.1:8080";

// ... Create Protocol
TsgcWSPClient_Files oClient_Files = new TsgcWSPClient_Files();
oClient_Files.Client = oClient;

// ... Start Server
oServer.Active = true;

// ... Connect client and Send File in packets of 100000 bytes
if oClient.Connect() then
    oClient_Files.SendFile("c:\Documents\yourfile.txt", 100000, TwsQoS.qosLevel0, "");
```

Protocol Presence

Il protocollo Presence consente di sapere chi è iscritto a un canale; questo semplifica la creazione di applicazioni di chat e la conoscenza di chi è online, ad esempio: utenti di giochi, sale chat, utenti che visualizzano lo stesso documento...

Funzionalità

- Per impostazione predefinita l'utente è **identificato da un nome**, ma questo può essere **personalizzato** passando ulteriori dati: email, azienda, twitter...
- Eventi per **autorizzare** se un **canale** può essere creato, se un **membro** è ammesso...
- Ogni volta che un nuovo **membro si unisce** a un canale, tutti i membri vengono **notificati**.
- **Pubblica messaggi** a tutti i sottoscrittori del canale.
- **Basso consumo** di memoria.

Componenti

[TsgcWSPServer_Presence](#): Componente VCL Server Protocol Presence.

[TsgcWSPClient_Presence](#): Componente VCL Client Protocol Presence.

Classi

[TsgcWSPresenceMessage](#): l'oggetto che incapsula le informazioni del pacchetto presence.

TsgcWSPServer_Presence

Questo è il componente server del protocollo Presence.

Introduzione

Questo è il componente Server Presence Protocol. È necessario trascinare questo componente nel form e selezionare un componente [TsgcWebSocketServer](#) tramite la proprietà Server.

Riferimento

- Proprietà — 6 proprietà disponibili sul componente.
- Metodi — 1 metodo pubblico.
- Eventi — 17 eventi gestibili.

TsgcWSPresenceMessage

Questo oggetto incapsula tutti i messaggi interni scambiati dal protocollo di presenza tra server e client.

TsgcWSPresenceMember

ID: identificatore interno

Name: nome del membro, fornito dal client.

Info: informazioni aggiuntive del membro, fornite dal client.

Data: TObject che può essere utilizzato per scopi del server.

TsgcWSPresenceMemberList

Member[i]: membro di un elenco per indice

Count: numero di membri dell'elenco

TsgcWSPresenceChannel

Name: nome del canale, fornito dal client.

TsgcWSPresenceMsg

Text: messaggio di testo, fornito dal client quando chiama il metodo Publish

TsgcWSPresenceError

Code: valore intero che identifica l'errore

Text: descrizione dell'errore.

TsgcWSPClient_Presence

Questo è il componente Client Presence Protocol.

Introduzione

Questo è il componente del protocollo di presenza client. È necessario inserire questo componente nel form e selezionare un componente [TsgcWebSocketClient](#) tramite la proprietà Client.

Riferimento

- Proprietà — 7 proprietà disponibili nel componente.
- Metodi — 5 metodi pubblici.
- Eventi — 16 eventi che è possibile gestire.

Protocollo Presence Javascript

Presence Protocol Javascript sgcWebSockets utilizza i file **sgcWebSocket.js** e **presence.esegece.com.js**.

Qui è possibile trovare i metodi disponibili; è necessario sostituire le variabili `{%host%}` e `{%port%}` secondo necessità, ad esempio: se si è configurato il server sgcWebSocket per ascoltare la porta 80 sul sito web `www.example.com` è necessario configurare:

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
<script src="http://www.example.com:80/presence.esegece.com.js"></script>
```

Apri Connessione

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  var socket = new sgcws_presence('ws://{%host%}:{%port%}');
</script>
```

Nuovo membro dopo la connessione

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  var socket = new sgcws_presence('ws://{%host%}:{%port%}');
  socket.on('sgcsession', function(event)
  {
    socket.newmember(event.id, 'John', 'Additional Info');
  });
</script>
```

Sottoscrivi al canale Topic 1

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  var socket = new sgcws_presence('ws://{%host%}:{%port%}');
  socket.subscribe('Topic 1');
</script>
```

Annullare la sottoscrizione dal canale Topic 1

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  var socket = new sgcws_presence('ws://{%host%}:{%port%}');
  socket.unsubscribe('Topic 1');
</script>
```

Pubblica Messaggio sul Canale Topic 1

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  var socket = new sgcsws_presence('ws://{%host%}:{%port%}');
  socket.publish('Hello sgcWebSockets!', 'Topic 1');
</script>
```

Ricevi Messaggio

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  var socket = new sgcsws_presence('ws://{%host%}:{%port%}');
  socket.on('sgcpublishmsg', function(event)
  {
    console.log('#publishmsg: ' + event.message.text);
  });
</script>
```

Ottieni tutti i membri connessi

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  var socket = new sgcsws_presence('ws://{%host%}:{%port%}');
  socket.on('sgcgetmembers', function(event)
  {
    for (var i in event.members) {
      console.log(event.members[i].id + ' ' + event.members[i].name);
    }
  });
  socket.getmembers();
</script>
```

Mostra un avviso quando i membri si iscrivono/disiscrivono

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  var socket = new sgcsws_presence('ws://{%host%}:{%port%}');
  socket.on('sgcnewmember', function(event)
  {
    alert('#new member: ' + event.member.name);
  });
  socket.on('sgcremovemember', function(event)
  {
    alert('#removed member: ' + event.member.name);
  });
  socket.on('sgcnewchannelmember', function(event)
  {
    alert('#new member: ' + event.member.name + ' in channel: ' + event.channel.name);
  });
  socket.on('sgcremovechannelmember', function(event)
  {
    alert('#remove member: ' + event.member.name + ' from channel: ' + event.channel.name);
  });
</script>
```

Mostra avviso negli eventi OnConnect, OnDisconnect e OnError

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  var socket = new sgcws_presence('ws://{%host%}:{%port%}');
  socket.on('open', function(event)
  {
    alert('sgcWebSocket Open!');
  });
  socket.on('close', function(event)
  {
    alert('sgcWebSocket Closed!');
  });
  socket.on('sgcerrormemberchannel', function(event)
  {
    alert('#error member channel: ' + event.error.text);
  });
  socket.on('sgcerrorpublishmsg', function(event)
  {
    alert('#error publish: ' + event.error.text);
  });
  });
</script>
```

Chiudi connessione

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  socket.close();
</script>
```

Protocollo E2EE

La crittografia end-to-end (E2EE) significa che i messaggi vengono crittografati sul dispositivo del mittente e possono essere decrittografati solo sui dispositivi del destinatario. Il server instrada i pacchetti ma non può leggere il contenuto in chiaro.

Questo argomento spiega il flusso tecnico per:

- **Direct message (1:1)** tra due peer.
- **Messaggi di gruppo** utilizzando un modello sender-key con rotazione delle chiavi basata sull'appartenenza.

Componenti crittografici di base

- **Identità / accordo di chiave:** Elliptic Curve Diffie-Hellman (ECDH) P-256.
- **Cifratura simmetrica:** AES-256-GCM.
- **Derivazione della chiave:** HKDF-SHA-256 per derivare le chiavi di crittografia dai segreti condivisi.
- **Autenticazione del messaggio:** tag di autenticazione AEAD (fornito da AES-GCM).
- **Protezione dal replay:** nonce/IV per messaggio più contatori di sequenza del mittente.

Flusso tecnico dei messaggi diretti (1:1)

1. **Scoperta della chiave pubblica**
Ogni peer pubblica una chiave di identità pubblica. Il server può distribuire le chiavi pubbliche, ma le chiavi private non lasciano mai il dispositivo client.
2. **Stabilimento del segreto condiviso**
Il mittente e il destinatario eseguono ECDH (chiave privata + chiave pubblica del peer) e ottengono lo stesso segreto condiviso.
3. **Derivazione della chiave di sessione**
HKDF-SHA-256 deriva una o più chiavi simmetriche (chiave di cifratura, chiave di intestazione opzionale) dall'output ECDH.
4. **Crittografia dei messaggi**
Il testo in chiaro viene cifrato con AES-256-GCM utilizzando un nonce/IV univoco. L'output include il testo cifrato più il tag di autenticazione.
5. **Trasporto**
Il server inoltra payload cifrati e metadati (ad esempio: id mittente, id chiave, contatore, timestamp) senza accesso al testo in chiaro.
6. **Decifratura del destinatario**
Il destinatario deriva la stessa chiave di sessione, verifica il tag di autenticazione e decifra. Qualsiasi manomissione provoca un errore di autenticazione.

Flusso tecnico dei messaggi di gruppo (Sender Keys)

Per i gruppi, cifrare ogni messaggio separatamente per ogni membro è costoso. Un'ottimizzazione comune è la progettazione con **sender key**:

- Ogni mittente mantiene uno **Stato della chiave del mittente** per gruppo.
- Lo stato contiene una **chiave di catena** simmetrica e un contatore di messaggi.
- Per ogni messaggio di gruppo in uscita, il mittente deriva una chiave di messaggio monouso dalla chiave della catena e la avanza (hash ratchet).
- La chiave del messaggio cifra il payload con AES-256-GCM.

Come viene distribuita una chiave mittente

1. Quando un mittente si unisce a un gruppo, crea un nuovo stato di sender key.
2. Lo stato della chiave sender viene condiviso con ciascun membro corrente tramite sessioni cifrate 1:1 esistenti (E2EE a coppie).
3. Dopo la distribuzione, i payload normali del gruppo utilizzano il percorso veloce sender-key (singola cifratura per messaggio).

Modifiche all'appartenenza e rotazione della chiave del mittente

Per preservare la segretezza forward e backward, le chiavi del mittente di gruppo devono ruotare in occasione degli eventi di appartenenza:

- **Un utente si unisce a un gruppo:** ruota le sender key in modo che il nuovo membro non possa decifrare la cronologia precedente a meno che non venga condivisa esplicitamente.
- **L'utente lascia o viene rimosso:** ruota immediatamente le sender key in modo che il membro rimosso non possa decrittografare il nuovo traffico.
- **Rotazione periodica:** la rotazione facoltativa basata sul tempo o sul conteggio dei messaggi riduce l'impatto della compromissione delle chiavi.

Sequenza tipica di rotazione:

1. Crea un nuovo stato della chiave sender (nuovo id chiave, nuova chiave di catena).
2. Distribuirli solo ai membri attualmente autorizzati attraverso canali E2EE a coppie.
3. Inizia a crittografare i nuovi messaggi di gruppo con il nuovo id di chiave.
4. Accettare il vecchio id della chiave solo durante una breve finestra di transizione, poi ritirarlo.

Proprietà di sicurezza

- **Riservatezza:** solo i client con chiavi valide possono decifrare.
- **Integrità e autenticità:** la verifica AEAD rileva manomissioni e testi cifrati contraffatti.
- **Cecità del server:** i server relay non conservano chiavi in chiaro per il contenuto dei messaggi.
- **Protezione post-appartenenza:** le chiavi ruotate impediscono agli ex membri di accedere ai messaggi futuri del gruppo.

Componenti

[TsgcWSPServer_E2EE](#): componente del protocollo server E2EE. Inoltra i messaggi crittografati tra i client senza conoscerne il contenuto.

[TsgcWSPClient_E2EE](#): Componente del Protocollo Client E2EE. Gestisce lo scambio di chiavi, la cifratura e la decifratura sui dispositivi peer.

TsgcWSPServer_E2EE

Il componente del protocollo server E2EE instrada i messaggi diretti e di gruppo cifrati tra i client mantenendo opachi i payload.

Introduzione

Il componente del protocollo server E2EE instrada messaggi diretti e di gruppo cifrati tra i client mantenendo i payload opachi. Aggiungerlo a un form, assegnare un [TsgcWebSocketHTTPServer](#) (o altro componente server) alla proprietà **Server** e configurare le opzioni E2EE. L'implementazione si trova nell'unità **sgcWebSocket_Protocol_E2EE_Server**.

Configurazione

Per utilizzare il componente:

- Imposti **Server** su un'istanza **TsgcWebSocketHTTPServer** configurata.
- Configurare la gestione degli acknowledgment in **E2EE_Options.Ack**.
- Scelga come gestire le nuove chiavi pubbliche con **E2EE_Options.PublicKeys.ReceiveNewPublicKey**.
- Impostare **Server.Active := True** per iniziare l'ascolto.

Riferimento

- Properties — 5 proprietà disponibili sul componente.
- Methods — 0 metodi pubblici.
- Eventi — 8 eventi gestibili.

TsgcWSPClient_E2EE

Il componente di protocollo client E2EE aggiunge la crittografia end-to-end su una connessione WebSocket, inclusi messaggi diretti crittografati e messaggi di gruppo crittografati.

Introduzione

Il componente protocollo client E2EE aggiunge la crittografia end-to-end su una connessione WebSocket, inclusi messaggi diretti cifrati e messaggi di gruppo cifrati. Trascinare il componente su un form, assegnare un [TsgcWebSocketClient](#) alla proprietà **Client** e configurare le opzioni E2EE prima di connettersi. L'implementazione si trova nell'unità **sgcWebSocket_Protocol_E2EE_Client**.

Configurazione

Per utilizzare il componente:

- Imposti **Client** su un'istanza configurata di **TsgcWebSocketClient**.
- Assegnare un identificatore utente univoco tramite **E2EE_Options.UserId**.
- Abilitare gli acknowledgment dei messaggi con **E2EE_Options.Ack.RcvDirectMessage** e **E2EE_Options.Ack.RcvGroupMessage** quando si desiderano callback sullo stato di consegna.
- Impostare **Client.Active := True** per aprire la connessione WebSocket.

Riferimento

- Properties — 5 proprietà disponibili sul componente.
- Metodi — 9 metodi pubblici.
- Events — 19 eventi gestibili.

API WebSocket

Esistono diverse implementazioni basate su WebSocket: finanza, pubblicazione di messaggi, code... sgcWebSockets implementa le API più importanti basate sul protocollo WebSocket. Per usare un'API, basta collegare il componente API al client e tutti i messaggi saranno gestiti dal componente API (solo un componente API può essere collegato a un client).

API Client

API	Description
Binance	è un exchange di criptovalute internazionale multilingue.
Binance Futures	consente di connettersi agli stream di mercato WebSocket / REST di Binance Futures
Coinbase	Coinbase è un exchange di criptovalute con sede negli Stati Uniti. Consente di scambiare Bitcoin (BTC), Ethereum (ETH) e altro ancora in USD, EUR e GBP. Supporta WebSocket API e REST API.
SignalR	è una libreria per sviluppatori ASP.NET che semplifica lo sviluppo di funzionalità di comunicazione in tempo reale. WebSockets come trasporto.
SignalRCore	ASP.NET Core SignalR è una libreria open source che semplifica l'aggiunta di funzionalità di comunicazione in tempo reale alle applicazioni.
SocketIO	è una libreria JavaScript per applicazioni web in tempo reale. Consente la comunicazione in tempo reale tra client web e server.
Kraken	è un exchange di criptovalute con sede negli Stati Uniti.
Kraken Futures	consente di connettersi ai dati di mercato WebSocket / REST di Kraken Futures.
Pusher	Pusher è una piattaforma semplice e affidabile con messaggistica pub/sub flessibile e in tempo reale, autenticazione...
Bitfinex	Bitfinex è una delle piattaforme di trading di criptovalute più grandi e avanzate al mondo. Permette di scambiare Bitcoin, Ethereum, Ripple, EOS, Bitcoin Cash, Iota, NEO, Litecoin, Ethereum Classic.
Bitstamp	Bitstamp è uno degli exchange di criptovalute più longevi al mondo, a supporto di Bitcoin, Ethereum, Litecoin, Bitcoin Cash, Iota, NEO, Litecoin, Ethereum Classic dal 2011.
Huobi	è un exchange di criptovalute internazionale multilingue.
Cex	è un exchange di criptovalute e precedente fornitore di cloud mining di Bitcoin.
Cex Plus	CEX.IO Exchange Plus è la piattaforma di trading di criptovalute definitiva che offre la migliore esperienza di trading avanzati
Bitmex	è un exchange di criptovalute e piattaforma di trading su derivati.
3Commas	È un bot di trading di criptovalute.
Kucoin	è un exchange di criptovalute che consente di acquistare, vendere e conservare criptovalute come Bitcoin, KCS, SHIB, DOGE
Kucoin Futures	consente di connettersi ai server Kucoin Futures (WebSocket e REST)
OKX	precedentemente nota come OKEx, è uno dei più grandi exchange di trading di criptovalute
XTB	Trading FX e CFD, con accesso a oltre +2000 mercati finanziari.
Discord	è uno degli strumenti di comunicazione più diffusi per i videogiochi online e lo sviluppo di applicazioni.
Bybit	exchange di criptovalute e piattaforma di trading
OpenAI API	L'API OpenAI Realtime abilita interazioni multimodali a bassa latenza, incluse speech-to-text, text-to-speech e trascrizione in tempo reale.
MEXC	Piattaforma centralizzata di criptovalute e trading, questo componente implementa le API più importanti basate sul protocollo WebSocket.
MEXC Futures	Piattaforma di scambio e trading di criptovalute centralizzata; questo componente implementa le API più importanti basate sul protocollo WebSocket.
Bitget	Exchange e piattaforma di trading di criptovalute con supporto per i mercati Spot e Futures
Gate.io	Piattaforma di scambio criptovalute che supporta il trading Spot e Futures con API REST e WebSocket

Deribit	Exchange di derivati di criptovalute che offre il trading di futures e opzioni su Bit
Crypto.com	Exchange di criptovalute che supporta canali Market e User con API WebSocket
HTX	Borsa internazionale di criptovalute (ex Huobi) con API REST per dati di mercato

Le API WebSocket possono essere registrate in fase di **runtime**, è sufficiente chiamare il metodo **RegisterAPI** e passare il componente API come parametro.

Altre API client

API	Description
Telegram	è un servizio di messaggistica istantanea e VoIP basato su cloud. Gli utenti possono inviare foto, video, adesivi, audio e file di qualsiasi tipo.
Whatsapp	è un servizio di messaggistica istantanea centralizzato e voice-over-IP freeware, non a pagamento a livello internazionale
RCON	è un protocollo di comunicazione basato su TCP/IP che consente di impartire comandi
CryptoHopper	È un bot di trading di criptovalute e gestore di portafoglio.
CryptoRobotics	È un robot di trading di criptovalute.

Server API

API	Description
RTCMultiConnection	RTCMultiConnection è una libreria JavaScript WebRTC per applicazioni peer-to-peer: video schermo, conferenze audio/video, condivisione di file, streaming multimediale.
WebPush	Il protocollo Web Push consente alle applicazioni web di inviare notifiche push anche se l'applicazione non è aperta o attiva.
WebAuthn	API server FIDO2/WebAuthn per l'autenticazione senza password tramite passkey.

API Binance

Binance

Binance è un exchange di criptovalute internazionale e multilingue. Offre alcune API per accedere ai dati di Binance. Sono supportate le seguenti API:

1. **Stream WebSocket:** consente di sottoscrivere ad alcuni metodi e ricevere dati in tempo reale. Gli eventi vengono inviati ai client dal server agli iscritti. Utilizza WebSocket come protocollo.
2. **Stream UserData:** i client sottoscritti ricevono i dettagli dell'account. Richiede una chiave API per l'autenticazione e utilizza WebSocket come protocollo.
3. **REST API:** Richiede una chiave API e un segreto per l'autenticazione e utilizza HTTPS come protocollo.
 1. [Dati di mercato](#)
 2. [Dati account e trading](#)
 3. [Portafoglio](#)
4. **Futures:** I flussi di dati di mercato WebSocket Futures sono supportati tramite il [Binance Futures Client API](#).

Il client supporta anche **Binance.us**; sono supportate le seguenti API:

1. **Stream WebSocket:** consente di sottoscrivere ad alcuni metodi e ricevere dati in tempo reale. Gli eventi vengono inviati ai client dal server agli iscritti. Utilizza WebSocket come protocollo.
2. **Stream UserData:** i client sottoscritti ricevono i dettagli dell'account. Richiede una chiave API per l'autenticazione e utilizza WebSocket come protocollo.
3. **REST API:** i client possono richiedere al server dati di mercato e dell'account. Richiede una API Key e una Secret per l'autenticazione e utilizza HTTPs come protocollo.

Proprietà

L'API di Binance ha 2 tipi di metodi: pubblici e privati. I metodi pubblici sono accessibili senza autenticazione, ad esempio: ottenere i prezzi dei ticker. Alcuni sono privati e relativi ai dati utente; tali metodi richiedono l'uso delle chiavi API di Binance.

- **ApiKey:** è possibile richiedere una nuova chiave API nel proprio account Binance; copiare il valore in questa proprietà.
- **ApiSecret:** Il segreto API è richiesto solo per REST_API, l'api websocket richiede solo ApiKey per alcuni metodi.
- **TestNet:** se abilitato, si conatterà all'account Demo di Binance (per impostazione predefinita false).
 - **HTTPLogOptions:** memorizza in un file di testo un log delle richieste HTTP
 - **Enabled:** se abilitato, memorizzerà tutte le richieste HTTP dell'API WebSocket.
 - **FileName:** percorso completo del file in cui verranno memorizzati i log
 - **REST:** memorizza in un file di testo un log delle richieste REST API
 - **Enabled:** se abilitato, memorizzerà tutte le richieste HTTP dell'API REST.
 - **FileName:** percorso completo del file in cui verranno memorizzati i log.
- **UserStream:** se abilitato, il client riceverà notifiche su aggiornamenti di Account, Ordini o Saldo (per impostazione predefinita true).
- **BinanceUS:** se abilitato, si conatterà ai server Binance.us (invece dei server Binance.com che è il valore predefinito).
- **ListenKeyOnDisconnect:** questa proprietà specifica cosa fare quando il client si disconnette dai server Binance con un ListenKey attivo.
 - **blkodDeleteListenKey:** Elimina il ListenKey attivo eseguendo una richiesta HTTP ai server Binance (questa è l'impostazione predefinita).
 - **blkodClearListenKey:** Non elimina il ListenKey dai server Binance e si limita a cancellare il valore del campo.
 - **blkodDoNothing:** non esegue alcuna operazione; pertanto, alla successiva connessione a Binance verrà tentato l'utilizzo dello stesso ListenKey.
- **UseCombinedStreams:** se abilitato, combinerà gli stream come segue:

```
{"stream": "<streamName>", "data": "<rawPayload>"}
```

 (disabilitato per impostazione predefinita)

Usi più comuni

- **WebSockets API**
 - [Come connettersi alla WebSocket API](#)
 - [Come sottoscrivere un canale WebSocket](#)
- **REST API**
 - [Come ottenere i dati di mercato](#)
 - [Come utilizzare le REST API private](#)
 - [Come fare trading Spot](#)
 - [Tempo delle richieste private](#)
 - [Prelievo](#)

WebSocket Stream API

L'endpoint base è `wss://stream.binance.com:9443`; il client può sottoscrivere/annullare la sottoscrizione agli eventi dopo una connessione riuscita.

Sono supportati i seguenti metodi di sottoscrizione/cancellazione della sottoscrizione.

Metodo	Parametri	Descrizione
AggregateTrades	Symbol	invia informazioni di trading aggregate per un singolo ordine taker
Trades	Symbol	pubblica informazioni grezze sulle transazioni; ogni transazione ha un acquirente e un venditore univoci
KLine	Symbol, Interval	invia aggiornamenti alle klines/candlestick correnti ogni secondo, minuto, ora...
MiniTicker	Symbol	Statistiche mini-ticker su finestra mobile di 24 ore. NON si tratta delle statistiche del giorno UTC, ma di una finestra mobile di 24 ore per le 24 ore precedenti.
AllMiniTickers		Statistiche mini-ticker a finestra mobile di 24 ore per tutti i simboli che hanno subito variazioni, in un array. NON si tratta delle statistiche del giorno UTC, bensì di una finestra mobile di 24 ore sulle 24 ore precedenti. Si noti che nell'array saranno presenti solo i ticker che hanno subito variazioni.
Ticker	Symbol	Statistiche del ticker su finestra mobile di 24 ore per un singolo simbolo. Queste NON sono le statistiche del giorno UTC, ma una finestra mobile di 24 ore relativa alle 24 ore precedenti.
AllMarketTickers		Statistiche del ticker su finestra mobile di 24 ore per tutti i simboli che sono cambiati, in un array. Queste NON sono le statistiche del giorno UTC, ma una finestra mobile di 24 ore relativa alle 24 ore precedenti. Si noti che nell'array saranno presenti solo i ticker che sono cambiati.
BookTicker	Symbol	Invia in tempo reale qualsiasi aggiornamento al prezzo o alla quantità della migliore offerta o domanda per un simbolo specificato.
AllBookTickers		Trasmette in tempo reale qualsiasi aggiornamento al prezzo o alla quantità del miglior bid o ask per tutti i symbol.
PartialBookDepth	Symbol, Depth	Le migliori offerte e richieste per <levels> livelli, aggiornate ogni secondo. I valori validi per <levels> sono 5, 10 o 20.
DiffDepth	Symbol	Aggiornamenti della profondità di prezzo e quantità del libro degli ordini utilizzati per gestire localmente un libro degli ordini.

Dopo una sottoscrizione / annullamento della sottoscrizione con esito positivo, il client riceve un messaggio in merito, dove id è il risultato del metodo `Subscribed` / `Unsubscribed`.

```
{
```

```
"result": null,
"id": 1
}
```

User Data Stream API

Richiede un ApiKey valido ottenuto dal proprio account Binance; ApiKey deve essere impostato nella proprietà `Binance.ApiKey` del componente.

I seguenti dati vengono inviati al client ogni volta che si verifica una modifica. Non è necessario sottoscrivere alcun metodo; questa operazione viene eseguita automaticamente se si imposta un ApiKey valido.

Metodo	Descrizione
Aggiornamento account	Lo stato dell'account viene aggiornato con l'evento <code>outboundAccountInfo</code> .
Aggiornamento Saldo	L'aggiornamento del saldo si verifica nelle seguenti situazioni: • Depositi o prelievi dall'account • Trasferimento di fondi tra account (ad esempio da Spot a Margin)
Aggiornamento ordine	Gli ordini vengono aggiornati con l'evento <code>executionReport</code> .

REST API

L'endpoint di base è: <https://api.binance.com>. Tutti gli endpoint restituiscono un oggetto o un array JSON. I dati vengono restituiti in ordine crescente. Prima i più vecchi, ultimi i più recenti.

Accesso alle opzioni REST API tramite la proprietà `REST_API.BinanceOptions`.

EndPoint API Pubblici

Questi endpoint sono accessibili senza alcuna autorizzazione.

EndPoint generali

Metodo	Parametri	Descrizione
Ping		Verifica la connettività all'API Rest.
GetServerTime		Verifica la connettività alla Rest API e ottiene l'ora corrente del server.
GetExchangeInformation		Regole di trading correnti dell'exchange e informazioni sui simboli

EndPoint dei dati di mercato

Metodo	Parametri	Descrizione
GetOrderBook	Symbol	Recupera il libro degli ordini.
GetTrades	Symbol	Ottiene gli scambi recenti

GetHistorical-Trades	Symbol	Ottiene trade più vecchi.
GetAggregate-Trades	Symbol	Recupera trade aggregati e compressi. I trade eseguiti contemporaneamente, dallo stesso ordine e allo stesso prezzo avranno la quantità aggregata.
GetKLines	Symbol, Interval	Barre Kline/candlestick per un simbolo. Le Kline sono identificate in modo univoco dal loro orario di apertura.
GetAveragePrice	Symbol	Prezzo medio attuale per un simbolo.
Get24hrTicker	Symbol	Statistiche di variazione del prezzo nelle ultime 24 ore a finestra scorrevole. Fare attenzione quando si accede senza specificare un simbolo.
GetPriceTicker	Symbol	Prezzo più recente per un simbolo.
GetPriceTickers	Simboli	Prezzo più recente per un array di simboli. Esempio: ["BTCUSDT","BNBUSDT"]
GetBookTicker	Symbol	Miglior prezzo/quantità sull'order book per un simbolo o più simboli.
GetUIKLines	Symbol, Interval	Barre Kline/candlestick per un simbolo. La risposta è simile a GetKLines, ottimizzata per la presentazione di grafici a candele.
GetRollingWindowTicker	Symbol, Symbols, WindowSize	Statistiche di variazione del prezzo su finestra temporale mobile. Nota: il valore predefinito di WindowSize è 1d se non specificato.
GetTradingDay-Ticker	Symbol, Symbols, Type	Statistiche di variazione del prezzo per una giornata di trading.

EndPoint API privati

Richiede un APIKey e un APISecret per essere autorizzato dal server.

EndPoint dati account

Metodo	Parametri	Descrizione
NewOrder	Symbol, Side, Type	Invia un nuovo ordine.
PlaceMarketOrder	Side, Symbol, Quantity	Inserisce un nuovo ordine di mercato
PlaceMarketQuoteOrder	Side, Symbol, QuoteOrderQty	Inserisce un nuovo Market Quote Order
PlaceLimitOrder	Side, Symbol, Quantity, LimitPrice	Inserisce un nuovo ordine limite
PlaceStopOrder	Side, Symbol, Quantity, StopPrice, LimitPrice	Inserisce un nuovo ordine Stop
PlaceStopTrailingOrder	Side, Symbol, Quantity, TrailingDelta, LimitPrice	Inserisce un Nuovo Ordine Stop Trailing
PlaceTakeProfitOrder	Side, Symbol, Quantity, StopPrice, LimitPrice	Inserisce un nuovo ordine Take Profit
PlaceTakeProfitTrailingOrder	Side, Symbol, Quantity, TrailingDelta, LimitPrice	Inserisce un nuovo ordine Take Profit Trailing

PlaceLimitMakerOrder	Side, Symbol, Quantity	Inserisce un nuovo ordine limite di mercato
TestNewOrder	Symbol, Side, Type	Testa la creazione di un nuovo ordine e la firma/recvWindow lungo. Crea e valida un nuovo ordine ma non lo invia al motore di corrispondenza.
QueryOrder	Symbol	Verifica lo stato di un ordine.
CancelOrder	Symbol	Annulla un ordine attivo. Annulla un ordine attivo. Deve essere inviato OrderId oppure OrigClientOrderId.
CancelAllOpenOrders	Symbol (opzionale)	
GetOpenOrders		Recupera tutti gli ordini aperti su un simbolo. Prestare attenzione quando si accede senza specificare un simbolo.
GetAllOrders	Symbol	Ottieni tutti gli ordini dell'account: attivi, annullati o eseguiti.
NewOCO	Symbol, Side, Quantity, Price, StopPrice	Invia un nuovo OCO
CancelOCO	Symbol	Annulla un intero Elenco Ordini
QueryOCO	Symbol	Recupera un OCO specifico in base ai parametri opzionali forniti
GetAllOCO		Recupera tutti gli OCO in base ai parametri opzionali forniti
GetOpenOCO		Recupera tutti gli OCO aperti.
GetAccountInformation		Ottieni le informazioni sull'account corrente.
GetAccountTradeList	Symbol	Ottieni le operazioni per un account e un simbolo specifici.
CancelReplaceOrder	Symbol, Side, Type, CancelReplaceMode	Annulla un ordine esistente e inserisce un nuovo ordine sullo stesso simbolo.
NewOrderListOCO	Symbol, Side, Quantity, AboveType, BelowType	Inserire un nuovo ordine OCO.
NewOrderListOTO	Symbol, WorkingType, WorkingSide, WorkingQuantity, WorkingPrice, PendingType, PendingSide, PendingQuantity	Inserisce un nuovo elenco di ordini OTO (One-Triggers-the-Other).
NewOrderListOTOCO	Symbol, WorkingType, WorkingSide, WorkingQuantity, WorkingPrice, PendingSide, PendingAboveType, PendingBelowType, PendingQuantity	Inserire un nuovo ordine lista OTOCO (One-Triggers-a-One-Cancels-the-Other).
NewSOROrder	Symbol, Side, Type, Quantity	Inserisce un ordine tramite Smart Order Routing (SOR).
TestSOROrder	Symbol, Side, Type, Quantity	Testa un nuovo ordine utilizzando lo Smart Order Routing (SOR). Crea e convalida un nuovo ordine ma non lo invia al matching engine.
GetOrderRateLimitUsage		Visualizza il conteggio corrente degli ordini dell'utente per tutti gli intervalli.
GetPreventedMatches	Symbol	Visualizza l'elenco degli ordini scaduti a causa di STP (Self Trade Prevention).
GetAllocations	Symbol	Recupera le allocazioni risultanti dal posizionamento degli ordini SOR.
GetAccountCommission	Symbol	Ottiene gli attuali tassi di commissione dell'account.

Converti EndPoints

Metodo	Parametri	Descrizione
GetAllConvert-Pairs	FromAsset, ToAsset	Interroga tutte le coppie di token convertibili e i rispettivi limiti superiori/inferiori dei token
GetConvertAssetInfo		Interrogazione per informazioni sulla precisione degli asset supportati
SendConvertQuoteRequest	FromAsset, ToAsset	Richiedi una quotazione per le coppie di token richieste
AcceptConvertQuote	Quoteld	Accetta la quotazione offerta tramite l'ID della quotazione.
GetConvertOrderStatus	OrderId o Quoteld	Interroga lo stato dell'ordine tramite l'ID ordine.
PlaceConvertLimitOrder	BaseAsset, QuoteAsset, Side, LimitPrice	Consentire agli utenti di inserire un ordine limite. baseAsset o quoteAsset possono essere determinati tramite l'endpoint exchangeInfo. Il prezzo limite è definito da baseAsset a quoteAsset. Viene utilizzato baseAmount o quoteAmount.
CancelConvertLimitOrder	OrderId	Consente agli utenti di annullare un ordine limite
GetConvertLimitOpenOrders		Consentire agli utenti di consultare tutti gli ordini limite esistenti
GetConvertTradeHistory	StartTime, EndTime	L'intervallo massimo tra startTime e endTime è di 30 giorni.

Endpoint Wallet

(*gli endpoint wallet funzionano solo con il server di produzione, non con quello demo)

Metodo	Descrizione
GetWalletSystemStatus	Recuperare lo stato del sistema.
GetWalletAllCoinsInformation	Ottieni informazioni sulle monete (disponibili per deposito e prelievo) per l'utente.
GetWalletDailyAccountSnapshot	Tipo: "SPOT", "MARGIN", "FUTURES" - Il periodo di query deve essere inferiore a 30 giorni - Supporta query solo nell'ultimo mese - Se startTime e endTime non vengono inviati, vengono restituiti per impostazione predefinita i record degli ultimi 7 giorni
SetWalletDisableFastWithdrawSwitch	Questa richiesta disabiliterà l'opzione fastwithdraw nel Suo account. È necessario abilitare l'opzione "trade" per la chiave API che richiede questo endpoint.
SetWalletEnableFastWithdrawSwitch	Questa richiesta abiliterà il prelievo rapido nel proprio account. È necessario abilitare l'opzione "trade" per la chiave API che richiede questo endpoint. Quando il prelievo rapido è attivato, il trasferimento di fondi verso un account Binance avviene istantaneamente. Non vi è alcuna transazione on-chain, nessun ID transazione e nessun costo di prelievo.
WalletWithdraw	Invia una richiesta di prelievo.
GetWalletDepositHistory	Recupera la cronologia dei depositi.

GetWalletWithdrawHistory	Recupera la cronologia dei prelievi.
GetWalletDepositAddress	Recupera l'indirizzo di deposito con la rete.
GetWalletAccountStatus	Recupera i dettagli dello stato dell'account.
GetWalletAccountAPI-TradingStatus	Recupera i dettagli sullo stato di trading dell'API dell'account.
GetWalletDustLog	Restituisce solo gli ultimi 100 record Restituisce solo i record successivi al 01/12/2020
GetWalletAssetsConvertedBNB	
WalletDustTransfer	Converti asset dust in BNB. È necessario abilitare l'autorizzazione Spot & Margin Trading per la chiave API che effettua richieste a questo endpoint.
GetWalletAssetDividendRecord	Interroga il registro dei dividendi degli asset.
GetWalletAssetDetail	Recupera i dettagli degli asset supportati su Binance.
GetWalletTradeFee	Recupera la commissione di trading
WalletUserUniversal-Transfer	È necessario abilitare l'opzione Permits Universal Transfer per la chiave API che richiede questo endpoint. MAIN_UMFUTURE Trasferimento dal conto Spot al conto Futures USDⓈ-M ENUM del Tipo: - MAIN_CMFUTURE Trasferimento dal conto Spot al conto COIN-M Futures - MAIN_MARGIN Trasferimento dal conto Spot al conto Margin (cross) - UMFUTURE_MAIN Trasferimento dal conto USDⓈ-M Futures al conto Spot - UMFUTURE_MARGIN Trasferimento dal conto USDⓈ-M Futures al conto Margin (cross) - CMFUTURE_MAIN Trasferimento dal conto COIN-M Futures al conto Spot - CMFUTURE_MARGIN Trasferimento dal conto COIN-M Futures al conto Margin(cross) - MARGIN_MAIN Trasferimento dal conto Margin (cross) al conto Spot - MARGIN_UMFUTURE Trasferimento dal conto Margin (cross) a USDⓈ-M Futures - MARGIN_CMFUTURE Trasferimento dal conto Margin (cross) a COIN-M Futures - ISOLATEDMARGIN_MARGIN Trasferimento dal conto Isolated margin al conto Margin(cross) - MARGIN_ISOLATEDMARGIN Trasferimento dal conto Margin(cross) al conto Isolated margin - ISOLATEDMARGIN_ISOLATEDMARGIN Trasferimento dal conto Isolated margin al conto Isolated margin - MAIN_FUNDING Trasferimento dal conto Spot al conto Funding - FUNDING_MAIN Trasferimento dal conto Funding al conto Spot - FUNDING_UMFUTURE Trasferimento dal conto Funding al conto UMFUTURE - UMFUTURE_FUNDING Trasferimento dal conto UMFUTURE al conto Funding - MARGIN_FUNDING Trasferimento dal conto MARGIN al conto Funding - FUNDING_MARGIN Trasferimento dal conto Funding al conto Margin - FUNDING_CMFUTURE Trasferimento dal conto Funding al conto CMFUTURE - CMFUTURE_FUNDING Trasferimento dal conto CMFUTURE al conto Funding
GetWalletQueryUserUniversalTransferHistory	- fromSymbol deve essere inviato quando il tipo è ISOLATEDMARGIN_MARGIN e ISOLATEDMARGIN_ISOLATEDMARGIN - toSymbol deve essere inviato quando il tipo è MARGIN_ISOLATEDMARGIN e ISOLATEDMARGIN_ISOLATEDMARGIN - Supporta solo le query degli ultimi 6 mesi - Se startTime e endTime non vengono inviati, vengono restituiti per impostazione predefinita i record degli ultimi 7 giorni
GetWalletFundingWallet	Attualmente supporta la query dei seguenti asset aziendali : Binance Pay, Binance Card, Binance Gift Card, Stock Token
GetWalletUserAsset	Recupera gli asset dell'utente, solo per i dati positivi.

GetWalletApiKeyPer- mission	

Eventi

I messaggi Binance vengono ricevuti nel componente TsgcWebSocketClient; è possibile utilizzare i seguenti eventi:

OnConnect

Dopo una connessione riuscita al server Binance.

OnDisconnect

Dopo una disconnessione dal server Binance

OnMessage

I messaggi inviati dal server al client vengono gestiti in questo evento.

OnError

In caso di qualsiasi errore nel protocollo, verrà chiamato questo evento.

OnException

Se si verifica un'eccezione non gestita, verrà chiamato questo evento.

Inoltre, esiste un evento specifico nel componente API Binance, chiamato **OnBinanceHTTPException**, che viene generato ogni volta che si verifica un errore nella chiamata di una richiesta HTTP (API REST o WebSocket User Stream).

(*) A causa di modifiche ai server Binance, le versioni Indy precedenti a Rad Studio 10.1 non saranno in grado di connettersi ai server di test. Questo problema non riguarda l'edizione Enterprise né i casi in cui la versione Indy è stata aggiornata all'ultima versione.

Binance | Connetti WebSocket API

Per connettersi all'API WebSocket di Binance, è sufficiente creare un nuovo client API Binance e collegarlo a Tsgc-WebSocketClient.

Vedere di seguito un esempio:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();  
TsgcWSAPI_Binance oBinance = new TsgcWSAPI_Binance();  
oBinance.Client = oClient;  
oClient.Active = true;
```

Binance | Subscribe WebSocket Channel

Binance offre una varietà di canali a cui è possibile sottoscrivere per ricevere aggiornamenti in tempo reale di dati di mercato, ordini... Di seguito è riportato un esempio di come sottoscrivere a un Ticker:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Binance oBinance = new TsgcWSAPI_Binance();
oBinance.Client = oClient;
oBinance.SubscribeTicker("bnbbtc");

void OnMessage(TsgcWSConnection Connection, const string aText)
{
    // here you will receive the ticker updates
}
```

Binance | Ottieni Dati di Mercato

Binance offre dati di mercato pubblici tramite endpoint REST. Quando si chiama uno di questi endpoint, si ottiene uno snapshot dei dati di mercato richiesti.

Gli endpoint di dati di mercato non richiedono autenticazione e sono quindi liberamente accessibili a tutti gli utenti.

Esempio: per ottenere uno snapshot del ticker BNBBTC, effettui la seguente chiamata:

```
TsgcWSAPI_Binance oBinance = new TsgcWSAPI_Binance();  
MessageBox.Show(oBinance.REST_API.GetPriceTicker("BNBBTC"));
```

Binance | Private REST API

Le REST API di Binance offrono endpoint pubblici e privati. Gli endpoint privati richiedono che i messaggi siano firmati per aumentare la sicurezza delle transazioni.

Prima di tutto è necessario accedere al proprio account Binance e creare una nuova API; si otterranno i seguenti valori:

- ApiKey
- ApiSecret

Questi campi devono essere configurati nella proprietà Binance del componente client API Binance.

Una volta configurato, è possibile iniziare a effettuare richieste private all'API REST Binance Pro

*Le richieste private richiedono che l'orario del computer locale sia sincronizzato; in caso contrario, le richieste verranno rifiutate dal server Binance. Consultare il seguente articolo a riguardo, [Binance Private Requests Time](#).

```
TsgcWSAPI_Binance oBinance = new TsgcWSAPI_Binance();
oBinance.Binance.ApiKey = "<your api key>";
oBinance.Binance.ApiSecret = "<your api secret>";
MessageBox.Show(oBinance.REST_API.GetAccountInformation());
```

Binance | Trade Spot

Binance consente di fare trading spot tramite la sua API REST.

Configurazione

Prima è necessario creare una **API Key** nel proprio account Binance e aggiungere i privilegi per il trading con Spot. Una volta completato, è possibile iniziare il trading spot.

Per prima cosa, **imposta la Sua ApiKey e il Suo ApiSecret** nel componente Binance Client, questi verranno utilizzati per firmare le richieste inviate al server Binance.

Inserisci un Ordine

Per inserire un nuovo ordine, è sufficiente chiamare il metodo **REST_API.NewOrder** del componente client Binance.

A seconda del tipo di ordine (di mercato, limitato...) l'API richiede un numero maggiore o minore di campi.

Campi obbligatori

- **Symbol:** l'ID del prodotto symbol, esempio: BNBBTC
- **Side:** BUY o SELL
- **type:** il tipo di ordine
 - LIMIT
 - MARKET
 - STOP_LOSS
 - STOP_LOSS_LIMIT
 - TAKE_PROFIT
 - TAKE_PROFIT_LIMIT
 - LIMIT_MAKER

Campi Obbligatori Aggiuntivi basati sul Tipo

- **LIMIT:** timeInForce, quantity, price
- **MARKET:** quantity o quoteOrderQty
- **STOP_LOSS / TAKE_PROFIT:** quantity, stopPrice
- **STOP_LOSS_LIMIT / TAKE_PROFIT_LIMIT:** timeInForce, quantity, price, stopPrice
- **LIMIT_MAKER:** quantity, price

Quando si invia un ordine, ci sono 2 possibilità:

1. **Operazione riuscita:** la funzione NewOrder restituisce il messaggio inviato dal server di Binance.
2. **Errore:** l'eccezione viene restituita nell'evento OnBinanceHTTPException.

Inserisci Ordine Market 1 BNBBTC

```
TsgcWSAPI_Binance oBinance = new TsgcWSAPI_Binance();
oBinance.Binance.ApiKey = "<api key>";
oBinance.Binance.ApiSecret = "<api secret>";
MessageBox.Show(oBinance.REST_API.NewOrder("BNBBTC", "BUY", "MARKET", "", 1));
```

Inserisci Ordine Limit 1 BNBBTC a 0.009260

```
TsgcWSAPI_Binance oBinance = new TsgcWSAPI_Binance();
oBinance.Binance.ApiKey = "<api key>";
```



```
oBinance.Binance.ApiSecret = "<api secret>";  
MessageBox.Show(oBinance.REST_API.NewOrder("BNBBTC", "BUY", "LIMIT", "GTC", 1, 0, 0.009260));
```

-

Binance | Private Requests Time

Quando si esegue una richiesta privata a Binance, il messaggio viene firmato per aumentare la sicurezza delle richieste. Il messaggio include l'ora locale e la invia all'interno del messaggio firmato; se l'ora locale differisce di più di 5 secondi dall'ora dei server Binance, la richiesta verrà rifiutata. È quindi importante verificare che l'ora locale sia sincronizzata; ciò è possibile utilizzando il metodo di sincronizzazione dell'ora del proprio sistema operativo.

La logica è la seguente:

```
if (timestamp < (serverTime + 1000) && (serverTime - timestamp) <= recvWindow) {  
  // process request  
} else {  
  // reject request  
}
```

Si consiglia di utilizzare un `recvWindow` ridotto di 5000 o inferiore! Il massimo non può superare 60000 millisecondi.

È possibile verificare l'ora del server Binance chiamando il metodo **GetServerTime**, che restituirà l'ora del server Binance

Il **RecvWindow** è impostato per impostazione predefinita a **5000**; questo valore può essere aumentato usando la proprietà **REST_API.BinanceOptions.RecvWindow**.

Binance | Prelievo

Binance consente di utilizzare l'API Wallet per inviare una richiesta di prelievo; i seguenti parametri sono obbligatori:

- Coin
- Indirizzo
- Importo

```
TsgcWSAPI_Binance oBinance = new TsgcWSAPI_Binance();
oBinance.Binance.ApiKey = "<your api key>";
oBinance.Binance.ApiSecret = "<your api secret>";
MessageBox.Show(oBinance.REST_API.WalletWithdraw("BTC", "7213fea8e94b4a5593d507237e5a555b", 0.25));
```

API Binance Futures

Binance

Binance è un exchange di criptovalute internazionale e multilingue. Offre alcune API per accedere ai dati di Binance. Questo componente consente di ottenere i Binance Futures WebSocket Market Streams.

<https://binance-docs.github.io/apidocs/futures/en>
<https://binance-docs.github.io/apidocs/delivery/en>

Contratti futures

L'API Binance ha 2 tipi di metodi: pubblici e privati. I metodi pubblici sono accessibili senza autenticazione, ad esempio: ottenere i prezzi del ticker. Alcuni sono privati e relativi ai dati dell'utente; questi metodi richiedono l'uso delle chiavi API Binance.

- **ApiKey:** è possibile richiedere una nuova chiave API nel proprio account Binance; copiare il valore in questa proprietà.
- **ApiSecret:** Il segreto API è richiesto solo per REST_API, l'api websocket richiede solo ApiKey per alcuni metodi.
- **TestNet:** se abilitato, si conatterà all'account Demo di Binance (per impostazione predefinita false).
 - **HTTPLogOptions:** memorizza in un file di testo un log delle richieste HTTP
 - **Enabled:** se abilitato, memorizzerà tutte le richieste HTTP dell'API WebSocket.
 - **FileName:** percorso completo del file in cui verranno memorizzati i log
 - **REST:** memorizza in un file di testo un log delle richieste REST API
 - **Enabled:** se abilitato, memorizzerà tutte le richieste HTTP dell'API REST.
 - **FileName:** percorso completo del file in cui verranno memorizzati i log.
- **UserStream:** se abilitato, il client riceverà notifiche su aggiornamenti di Account, Ordini o Saldo (per impostazione predefinita true).
- **ListenKeyOnDisconnect:** questa proprietà specifica cosa fare quando il client si disconnette dai server Binance con un ListenKey attivo.
 - **blkodDeleteListenKey:** Elimina il ListenKey attivo eseguendo una richiesta HTTP ai server Binance (questa è l'impostazione predefinita).
 - **blkodClearListenKey:** Non elimina il ListenKey dai server Binance e cancella semplicemente il valore del campo.
 - **blkodDoNothing:** non esegue alcuna azione, quindi alla successiva connessione a Binance verrà tentato l'utilizzo dello stesso ListenKey.
- **UseCombinedStreams:** se abilitato, combinerà gli stream come segue:
 {"stream": "<streamName>", "data": "<rawPayload>"} (disabilitato per impostazione predefinita)

Il client può connettersi a **USDT** o **COIN** Binance Futures; imposti il contratto che desidera negoziare utilizzando la proprietà **FuturesContracts**:

- **bfcUSDT:** si connette all'API USD-M Futures.
- **bfcCOIN:** si connette all'API COIN-M Futures.

Il client può connettersi agli account Binance di Produzione o Demo. Se la proprietà **TestNet** è abilitata, si collegherà all'account Demo, altrimenti si collegherà ai server Binance di produzione.

WebSocket Stream API

Il client può sottoscrivere o annullare la sottoscrizione agli eventi dopo una connessione riuscita. Sono supportati i seguenti metodi di sottoscrizione/cancellazione della sottoscrizione.

Metodo	Parametri	Descrizione

AggregateTrades	Symbol	Gli Aggregate Trade Streams inviano informazioni sulle operazioni aggregate per un singolo ordine taker ogni 100 millisecondi.
MarkPrice	Symbol, UpdateSpeed	Prezzo mark e tasso di finanziamento per un singolo simbolo trasmessi ogni 3 secondi o ogni secondo.
AllMarkPrice	Update-Speed	Mark price e funding rate per tutti i simboli inviati ogni 3 secondi o ogni secondo.
KLine	Symbol, Interval	Il flusso Kline/Candlestick invia aggiornamenti alle kline/candlestick correnti ogni 250 millisecondi (se esistenti).
MiniTicker	Symbol	Statistiche mini-ticker su finestra mobile di 24 ore per un singolo simbolo. NON sono le statistiche del giorno UTC, ma una finestra mobile di 24 ore da requestTime alle 24 ore precedenti.
AllMiniTicker		Statistiche mini-ticker della finestra mobile di 24 ore per tutti i simboli. Queste NON sono le statistiche del giorno UTC, ma una finestra mobile di 24 ore da requestTime a 24 ore prima. Si noti che nell'array saranno presenti solo i ticker che sono cambiati.
Ticker	Symbol	Statistiche del ticker su una finestra mobile di 24 ore per un singolo simbolo. NON sono le statistiche del giorno UTC, bensì una finestra mobile di 24 ore dal momento della richiesta alle 24 ore precedenti.
AllMarketTickers		Statistiche ticker su finestra mobile di 24 ore per tutti i simboli. NON si tratta delle statistiche del giorno UTC, ma di una finestra mobile di 24 ore dal momento della richiesta alle 24 ore precedenti. Si noti che nell'array saranno presenti solo i ticker che hanno subito variazioni.
BookTicker	Symbol	Invia in tempo reale qualsiasi aggiornamento al prezzo o alla quantità della migliore offerta o domanda per un simbolo specificato.
AllBookTickers		Trasmette in tempo reale qualsiasi aggiornamento al prezzo o alla quantità del miglior bid o ask per tutti i symbol.
LiquidationOrders	Symbol	Gli stream degli ordini di liquidazione forzata trasmettono le informazioni sugli ordini di liquidazione forzata per un simbolo specifico
AllLiquidationOrders		Gli stream All Liquidation Order inviano informazioni sugli ordini di liquidazione forzata per tutti i simboli nel mercato.
PartialBookDepth	Symbol, Depth	Migliori offerte e richieste, valori validi: 5, 10 o 20.
DiffDepth	Symbol	Offerte e richieste, inviate ogni 250 millisecondi, 500 millisecondi, 100 millisecondi o in tempo reale (se disponibili)

Dopo una sottoscrizione/annullamento sottoscrizione riuscito, il client riceve un messaggio a riguardo, dove l'id è il risultato del metodo Subscribed/Unsubscribed.

```
{
  "result": null,
  "id": 1
}
```

User Data Stream API

Richiede un ApiKey valido ottenuto dal proprio account Binance; ApiKey deve essere impostato nella proprietà Binance.ApiKey del componente.

I seguenti dati vengono inviati al client ogni volta che si verifica una modifica. Non è necessario sottoscrivere alcun metodo; questa operazione viene eseguita automaticamente se si imposta un ApiKey valido.

Metodo	Descrizione

Margin Call	Quando il rapporto di rischio della posizione dell'utente è troppo elevato, questo stream verrà inviato. Questo messaggio è utilizzato solo come informazione di orientamento sul rischio e non è raccomandato per le strategie di investimento. In caso di mercato altamente volatile, può esistere la possibilità che la posizione dell'utente sia stata liquidata nello stesso momento in cui questo stream viene inviato.
Aggiornamento saldo e posizione	L'aggiornamento del saldo si verifica nelle seguenti situazioni: • Quando il saldo o la posizione vengono aggiornati, questo evento verrà inviato. • Quando "FUNDING FEE" cambia nel saldo dell'utente.
Aggiornamento ordine	Quando viene creato un nuovo ordine, verrà inviato un evento di cambio di stato dell'ordine.

REST API

Tutti gli endpoint restituiscono un oggetto o un array JSON. I dati vengono restituiti in ordine crescente. Prima i più vecchi, poi i più recenti.

EndPoint API Pubblici

Questi endpoint sono accessibili senza alcuna autorizzazione.

EndPoint generali

Metodo	Parametri	Descrizione
Ping		Verifica la connettività all'API Rest.
GetServerTime		Verifica la connettività alla Rest API e ottiene l'ora corrente del server.
GetExchangeInformation		Regole di trading correnti dell'exchange e informazioni sui simboli

EndPoint dei dati di mercato

Metodo	Parametri	Descrizione
GetOrderBook	Symbol	Recupera il libro degli ordini.
GetTrades	Symbol	Ottiene gli scambi recenti
GetHistorical-Trades	Symbol	Ottiene trade più vecchi.
GetAggregate-Trades	Symbol	Recupera trade aggregati e compressi. I trade eseguiti contemporaneamente, dallo stesso ordine e allo stesso prezzo avranno la quantità aggregata.
GetKLines	Symbol, Interval	Barre Kline/candlestick per un simbolo. Le Kline sono identificate in modo univoco dal loro orario di apertura.
Get24hrTicker	Symbol	Statistiche di variazione del prezzo nelle ultime 24 ore a finestra scorrevole. Fare attenzione quando si accede senza specificare un simbolo.
GetPriceTicker	Symbol	Prezzo più recente per un simbolo o più simboli.

GetBookTicker	Symbol	Miglior prezzo/quantità sull'order book per un simbolo o più simboli.
GetMarkPrice	Symbol	Prezzo mark e tasso di finanziamento
GetFundingRateHistory	Symbol	
GetOpenInterest	Symbol	Ottiene l'open interest attuale di un simbolo specifico.
GetOpenInterestStatistics	Symbol, Period	
GetTopTraderAccountRatio	Symbol, Period	
GetTopTraderPositionRatio	Symbol, Period	
GetGlobalAccountRatio	Symbol, Period	
GetTakerVolume	Symbol, Period	
GetContinuousKLines	Pair, ContractType, Interval	Barre kline/candlestick per un tipo di contratto specifico.
GetIndexPriceKLines	Pair, Interval	Candele/barre Kline per il prezzo indice di una coppia.
GetMarkPriceKLines	Symbol, Interval	Barre Kline/candlestick per il prezzo mark di un simbolo.
GetPremiumIndexKLines	Symbol, Interval	Barre kline dell'indice premium di un simbolo.
GetFundingInfo		Ottiene le informazioni sul funding rate per tutti i simboli.
GetPriceTickerV2	Symbol	Prezzo più recente per un simbolo o simboli (V2).
GetIndexInfo	Symbol	Ottenere le informazioni sull'indice.
GetAssetIndex	Symbol	Ottenere l'indice degli asset per la modalità multi-asset.
GetConstituents	Symbol	Ottiene i componenti di un indice.
GetDeliveryPrice	Coppia	Ottieni il prezzo di consegna.
GetBasis	Pair, ContractType, Period	Ottenere i dati di base.

EndPoint API Privati

Richiede un APIKey e un APISecret per essere autorizzato dal server.

Endpoint account e transazioni

Metodo	Parametri	Descrizione
ChangePositionMode	DualPosition	Modifica la modalità di posizione dell'utente (Hedge Mode o One-way Mode) su OGNI simbolo
GetCurrentPositionMode		Ottiene la modalità di posizione dell'utente (Hedge Mode o One-way Mode) su OGNI simbolo
NewOrder	Symbol, Side, PositionSide, Type	Invia un nuovo ordine.

PlaceMarketOrder	Side, Symbol, Quantity	
PlaceLimitOrder	Side, Symbol, Quantity, Limit-Price	
PlaceStopOrder	Side, Symbol, Quantity, Stop-Price, LimitPrice	
PlaceTrailing-StopOrder	Side, Symbol, Quantity, aActivationPrice, aCallbackRate	
QueryOrder	Symbol	Verifica lo stato di un ordine.
CancelOrder	Symbol	Annulla un ordine attivo. È necessario inviare OrderId o OrigClientId.
CancelAllOpenOrders	Symbol	
AutoCancelAllOpenOrders	Symbol, CountdownTimer	Annulla tutti gli ordini aperti del simbolo specificato alla fine del conto alla rovescia indicato.
QueryCurrentOpenOrder	Symbol	
GetOpenOrders	Symbol	Recupera tutti gli ordini aperti su un simbolo. Prestare attenzione quando si accede senza specificare un simbolo.
GetAllOrders	Symbol	Ottieni tutti gli ordini dell'account: attivi, annullati o eseguiti.
GetAccountBalance		
GetAccountInformation		Ottieni le informazioni sull'account corrente.
ChangeInitialLeverage	Symbol, Leverage	Modifica la leva iniziale dell'utente per uno specifico mercato di simboli.
ChangeMarginType	Symbol, MarginType	
ModifyIsolated-PositionMargin	Simbolo, Quantità, Tipo	
GetPosition-MarginChange-History	Symbol	
GetPositionInformation	Symbol	
GetAccountTradeList	Symbol	
GetIncomeHistory	Symbol	
GetNotionalLeverageBracket	Symbol	
TestNewOrder	Symbol, Side, PositionSide, Type	Testa la creazione di un nuovo ordine e la firma/recvWindow lungo. Crea e valida un nuovo ordine ma non lo invia al motore di corrispondenza.
ModifyOrder	Symbol	Modifica un ordine esistente.
NewBatchOrders	BatchOrders	Inserisce più ordini.
ModifyBatchOrders	BatchOrders	Modifica più ordini.
CancelBatchOrders	Symbol	Annulla più ordini.

GetOrderAmendment	Symbol	Ottieni la cronologia delle modifiche all'ordine.
CountdownCancelAll	Symbol, CountdownTime	Annula tutti gli ordini aperti del simbolo specificato alla fine del conto alla rovescia indicato.
GetForceOrders	Symbol	Ottieni gli ordini di liquidazione forzata dell'utente.
GetADLQuantile	Symbol	Ottieni la stima del quantile ADL per le posizioni.
GetAccountBalanceV3		Ottieni il saldo dell'account futures (V3).
GetAccountInformationV3		Recupera le informazioni sull'account corrente (V3).
GetPositionInformationV3	Symbol	Recupera le informazioni sulla posizione corrente (V3).
GetCommissionRate	Symbol	Ottieni il tasso di commissione dell'utente.
GetAccountConfig		Ottieni la configurazione dell'account corrente.
GetSymbolConfig	Symbol	Ottieni la configurazione del simbolo.
GetOrderRateLimit		Ottieni il limite di frequenza degli ordini dell'utente.
GetApiTradingStatus	Symbol	Ottieni indicatori delle regole quantitative di trading API.
ChangeMultiAssetsMode	MultiAssetsMargin	Modifica la modalità multi-asset dell'utente. Modalità Multi-Asset: true; Modalità Single-Asset: false.
GetMultiAssetsMode		Recupera la modalità multi-asset corrente dell'utente.
SetFeeBurn	FeeBurn	Modifica lo stato di burn della commissione BNB dell'utente.
GetFeeBurn		Ottieni lo stato di riduzione delle commissioni BNB dell'utente.
CreateListenKey		Avvia un nuovo stream di dati utente. Lo stream verrà chiuso dopo 60 minuti a meno che non venga inviato un keepalive.
KeepAliveListenKey		Mantieni attivo un flusso di dati utente per evitare un timeout.
CloseListenKey		Chiude un user data stream.

Eventi

I messaggi Binance Futures vengono ricevuti nel componente TsgcWebSocketClient; è possibile utilizzare i seguenti eventi:

OnConnect

Dopo una connessione riuscita al server Binance.

OnDisconnect

Dopo una disconnessione dal server Binance

OnMessage

I messaggi inviati dal server al client vengono gestiti in questo evento.

OnError

In caso di qualsiasi errore nel protocollo, verrà chiamato questo evento.

OnException

Se si verifica un'eccezione non gestita, verrà chiamato questo evento.

Inoltre, esiste un evento specifico nel componente API Binance, chiamato **OnBinanceHTTPException**, che viene generato ogni volta che si verifica un errore nella chiamata di una richiesta HTTP (API REST o WebSocket User Stream).

(*) A causa di modifiche nei server Binance, le versioni di Indy precedenti a Rad Studio 10.1 non saranno in grado di connettersi ai Test Server. Questo problema non interessa la Enterprise Edition o se la versione di Indy è stata aggiornata all'ultima disponibile.

API Binance Futures | Trade

Binance Le consente di operare con i futures utilizzando la sua REST API.

Configurazione

Per prima cosa deve creare una **API Key** nel Suo account binance e aggiungere i privilegi per il trading con i Futures.

Una volta eseguita questa operazione, è possibile iniziare a fare trading con i futures.

Prima è necessario selezionare se si desidera fare trading con **USDT** o futures **COIN**; esiste una proprietà chiamata FuturesContracts dove è possibile impostare il tipo di contratto futures con cui si desidera fare trading

Quindi, **impostare ApiKey e ApiSecret** nel componente client Binance Futures; questi verranno utilizzati per firmare le richieste inviate al server Binance.

Inserisci un Ordine

Per inserire un nuovo ordine, basta chiamare il metodo **REST_API.NewOrder** del componente Binance Futures Client.

A seconda del tipo di ordine (market, limit...) l'API richiede più o meno campi.

Campi obbligatori

- **Symbol:** il simbolo dell'id del prodotto, esempio: BTCUSD_210326
- **Side:** BUY o SELL
- **type:** il tipo di ordine
 - LIMIT
 - MARKET
 - STOP
 - TAKE_PROFIT
 - STOP_MARKET
 - TAKE_PROFIT_MARKET
 - TRAILING_STOP_MARKET

Campi Obbligatori Aggiuntivi basati sul Tipo

- **LIMIT:** timeInForce, quantity, price
- **MARKET:** quantità
- **STOP/TAKE_PROFIT:** quantity, price, stopPrice
- **STOP_MARKET/TAKE_PROFIT_MARKET:** stopPrice
- **TRAILING_STOP_MARKET:** callbackRate

Quando si invia un ordine, ci sono 2 possibilità:

1. **Operazione riuscita:** la funzione NewOrder restituisce il messaggio inviato dal server di Binance.
2. **Errore:** l'eccezione viene restituita nell'evento OnBinanceHTTPException.

API SocketIO

SocketIO

Socket.IO è una libreria JavaScript per applicazioni web in tempo reale. Consente una comunicazione bidirezionale in tempo reale tra client web e server. È composta da due parti: una libreria lato client che viene eseguita nel browser e una libreria lato server per Node.js. Entrambi i componenti hanno un'API quasi identica. Come Node.js, è guidata dagli eventi.

Tipi di Messaggi

0: aperto (Inviato dal server quando viene aperto un nuovo trasporto (ricontrollo))

1: close (Richiede la chiusura di questo trasporto ma non termina la connessione stessa.)

2: ping (Inviato dal client. Il server deve rispondere con un pacchetto pong contenente gli stessi dati)
esempio

il client invia: 2probe

server sends: 3probe

3: pong (Inviato dal server in risposta ai pacchetti ping.)

4: messaggio stringa (messaggio effettivo; client e server devono richiamare i propri callback con i dati.)
esempio:

42/chat,["join",{"room:1}"]

4 è il tipo di pacchetto messaggio nel protocollo engine.io

2 è il tipo EVENT nel protocollo socket.io

/chat è il dato elaborato da socket.io

socket.io genererà l'evento "join"

passerà i dati "room: 1". È possibile omettere il namespace solo quando esso è /.

5: upgrade (Prima che engine.io cambi un transport, verifica se server e client possono comunicare su questo transport. Se questo test ha esito positivo, il client invia un pacchetto di upgrade che richiede al server di svuotare la cache sul vecchio transport e passare al nuovo transport.)

6: noop (Un pacchetto noop. Utilizzato principalmente per forzare un ciclo di polling quando viene ricevuta una connessione WebSocket in ingresso.)

Proprietà

API: specifica la versione di SocketIO:

ioAPI0: supporta server socket.io 0.* (selezionato per impostazione predefinita)

ioAPI1: supporta i server socket.io 1.*

ioAPI2: supporta server socket.io 2.*

ioAPI3: supporta i server socket.io 3.*

ioAPI4: supporta i server socket.io 4.*

Base64: se abilitato, i messaggi binari vengono ricevuti come base64.

HandShakeCustomURL: consente di personalizzare l'URL per ottenere la sessione socket.io.

HandShakeTimestamp: abilitare solo se si desidera inviare il timestamp come parametro quando viene richiesta una nuova sessione (abilitare questa proprietà se si tenta di accedere a un server gevent-socketio Python).

HandShakeAuthToken: se il server richiede un token per l'autenticazione, imposti qui il token di autenticazione.

Namespace: consente di impostare un namespace quando ci si connette al server.

Polling: disabilitando questa proprietà, il client si conatterà direttamente al server utilizzando websocket come trasporto.

Parametri: consente di impostare i parametri di connessione.

EncodeParameters: se abilitato, i parametri vengono codificati.

Metodi

Utilizzare il metodo WriteData per inviare messaggi al server socket.io (seguendo la sezione Tipi di Messaggio).

1. Chiami il metodo "add user" con un parametro utilizzando John come nome utente.

```
WriteData("42[\"add user\", \"John\"]");
```

Eventi

OnHTTPRequest

Prima che venga stabilita una nuova connessione websocket, il server socket.io richiede al client di aprire una nuova connessione HTTP per ottenere un nuovo id di sessione. In alcuni casi, il server socket.io richiede l'autenticazione tramite header HTTP. Può utilizzare questo evento per aggiungere header HTTP personalizzati, come l'autorizzazione Basic o l'autenticazione con token Bearer.

OnAfterConnect

Questo evento viene chiamato dopo che la connessione socket.io ha avuto esito positivo e il client può inviare messaggi al server. Qui è possibile, ad esempio, sottoscrivere i namespace.

API Coinbase

Coinbase

API supportate

- **API WebSockets:** si connette a un server websocket pubblico e fornisce aggiornamenti in tempo reale dei dati di mercato.
- **REST API:** Le REST API dispongono di endpoint per la gestione di account e ordini, nonché di dati di mercato pubblici.

Usi più comuni

- **WebSockets API**
 - [Come connettersi all'API WebSocket](#)
 - [Come sottoscrivere un canale WebSocket](#)
- **REST API**
 - [Come Ottenere Dati di Mercato](#)
 - [Come Utilizzare la REST API Privata](#)
 - [Come Piazzare Ordini](#)
 - [Come Utilizzare un Account SandBox](#)
 - [Tempo per le Richieste Private](#)

WebSockets API

Il feed WebSocket è disponibile pubblicamente e fornisce aggiornamenti di mercato in tempo reale per ordini e transazioni. In produzione sono supportati due endpoint:

- **Market Data** è il feed che fornisce aggiornamenti sia per gli ordini che per le transazioni. La maggior parte dei canali è ora disponibile senza autenticazione.
- **User Order Data** fornisce aggiornamenti sugli ordini dell'utente.

È possibile sottoscrivere ai seguenti canali:

Method	Argomenti	Description
SubscribeHeart-Beat		Ping periodici del server per mantenere attive tutte le connessioni
SubscribeStatus	aProductId: id del prodotto	Invia tutti i prodotti e le valute a un intervallo prestabilito
SubscribeCandles	aProductId: id del prodotto	Aggiornamenti in tempo reale sui candlestick dei prodotti
SubscribeTicker	aProductId: id del prodotto	Aggiornamenti del prezzo in tempo reale ogni volta che avviene una corrispondenza
SubscribeTicker-Batch	aProductId: id del prodotto	Aggiornamenti del prezzo in tempo reale ogni 5000 millisecondi
SubscribeLevel2	aProductId: id del prodotto	Tutti gli aggiornamenti e il modo più semplice per mantenere lo snapshot del libro degli ordini
SubscribeMarket-Trades	aProductId: id del prodotto	Aggiornamenti in tempo reale ogni volta che avviene una transazione di mercato

SubscribeUser	aProductId: id del prodotto	Invia solo i messaggi che includono l'utente autenticato
SubscribeFuture-sBalanceSummary		Aggiornamenti in tempo reale ogni volta che il saldo futures di un utente cambia

I canali User e FuturesBalanceSummary richiedono l'autenticazione; pertanto, richiedere prima le chiavi API nel proprio account Coinbase e quindi impostare i valori nella proprietà Coinbase del componente:

- ApiKey
- ApiSecret

L'autenticazione comporta una serie di vantaggi:

1. I messaggi in cui si è uno dei partecipanti sono espansi e contengono campi più utili
2. Si riceveranno messaggi privati, come informazioni sul ciclo di vita degli ordini stop inseriti

REST API

Endpoint privati

Gli endpoint privati sono disponibili per la gestione degli ordini e degli account.

Prima di poter firmare qualsiasi richiesta, deve creare una API key tramite il sito web di Coinbase Pro. La API key sarà associata a uno specifico profilo. Alla creazione di una chiave avrà 3 informazioni che deve ricordare:

- Chiave
- Segreto
- Passphrase

La chiave e il segreto saranno generati casualmente e forniti da Coinbase Pro; la passphrase sarà fornita dall'utente per proteggere ulteriormente l'accesso all'API. Coinbase Pro memorizza l'hash con salt della passphrase per la verifica, ma non può recuperarla in caso di smarrimento.

È possibile limitare la funzionalità delle chiavi API. Prima di creare la chiave, deve scegliere quali permessi desidera che la chiave abbia. I permessi sono:

- View - Concede a una chiave i permessi di lettura. Include tutti gli endpoint GET.
- Transfer - Consente a una chiave di trasferire valuta per conto di un account, inclusi depositi e prelievi. Abilitare con cautela - i trasferimenti tramite chiave API BYPASSERANNO l'autenticazione a due fattori.
- Trade - Consente a una chiave di inserire ordini, nonché di recuperare i dati di trading. Ciò include POST / orders e diversi endpoint GET.

Account

Method	Argomenti	Descrizione
ListAccounts		Ottieni un elenco di account di trading dal profilo della chiave API.
GetAccount	aAccountId: id dell'account	Informazioni per un singolo account. Utilizzi questo endpoint quando conosce l'account_id. La chiave API deve appartenere allo stesso profilo dell'account.

Ordini

Method	Argomenti	Descrizione

PlaceNewOrder	aOrder: classe che contiene tutti i campi possibili di un ordine	Inserisce un nuovo ordine. Da utilizzare solo se è necessario accedere alle opzioni avanzate degli ordini.
PlaceMarketOrder	aSide: buy o sell aProductId: id del prodotto aQuoteSize: La quantità del secondo asset nella coppia di trading. aBaseSize: La quantità del primo asset nella coppia di trading aClient_oid: ID ordine selezionato dall'utente per identificare il proprio ordine	Inserisce un nuovo ordine Market.
PlaceLimitOrder	aSide: buy o sell aProductId: id del prodotto aQuoteSize: La quantità del secondo Asset nella coppia di trading. aBaseSize: La quantità del primo Asset nella coppia di trading aLimitPrice: limite di prezzo Client_oid: ID ordine scelto dall'utente per identificare il proprio ordine	Inserisce un nuovo ordine Limit.
PlaceStopOrder	aSide: buy o sell ProductId: ID del prodotto aBaseSize: La quantità del primo Asset nella Trading Pair StopPrice: prezzo dello stop aLimitPrice: prezzo limite aStopDirection: perdita o ingresso Client_oid: ID dell'ordine scelto dall'utente per identificare il proprio ordine	Inserisce un nuovo ordine Stop
CancelOrder	aOrderId: id dell'ordine	Annula un ordine inserito in precedenza. L'ordine deve appartenere al profilo a cui appartiene la chiave API.
EditOrder	aOrderId: id dell'ordine aPrice: prezzo aSize: Importo	Modifica un ordine con una nuova dimensione o un nuovo prezzo specificato
EditOrderPreview	aOrderId: id dell'ordine aPrice: prezzo aSize: Importo	Anteprima di una richiesta di modifica dell'ordine con una nuova dimensione o un nuovo prezzo specificati.
ListOrders		Ottieni un elenco di ordini filtrati per parametri di query opzionali (product_id, order_status, ecc.).
GetOrder	aOrderId: id dell'ordine	Ottiene un singolo ordine tramite l'ID dell'ordine.
PreviewOrder		Anteprima di un ordine.
ClosePosition	aOrderId: id dell'ordine aProductId: ID del prodotto aSize: quantità	Inserisce un ordine per chiudere qualsiasi posizione aperta per un product_id specificato.

Dati di mercato

Method	Argomenti	Descrizione
GetPublicProducts		Ottiene un elenco delle coppie di valute disponibili per il trading.

GetPublicProduct	aProductId: ID del prodotto	Ottieni informazioni su un singolo prodotto tramite ID prodotto.
GetPublicProduct-Book	aProductId: id del prodotto	Ottieni un elenco di offerte/richieste per un singolo prodotto. Il livello di dettaglio visualizzato può essere personalizzato con il parametro limit.
GetPublicProduct-Candles	aProductId: id del prodotto aStart: inizio dell'intervallo di tempo aEnd: fine dell'intervallo di tempo aGranularity: Il periodo temporale rappresentato da ogni candela.	Ottieni le tariffe per un singolo prodotto tramite ID prodotto, raggruppate in bucket.
GetTrades	aProductId: id del prodotto	Ottieni informazioni di snapshot per ID prodotto sugli ultimi trade (tick) e il miglior bid/ask.
GetTime		Ottieni l'ora corrente dall'API Coinbase Advanced.

Fills

Method	Argomenti	Descrizione
GetFillsByOrderId		Ottiene un elenco di fill filtrati per id dell'ordine
GetFillsByProductId		Ottieni un elenco di eseguiti filtrati per product id
GetFillsByTradeId		Ottieni un elenco di esecuzioni filtrate per ID transazione

Converti

Method	Argomenti	Descrizione
CreateConvertQuote		Crea un preventivo di conversione tra valute.
CommitConvertTrade		Conferma una transazione di conversione.
GetConvertTrade		Ottieni i dettagli della transazione di conversione.

Commissioni

Method	Argomenti	Descrizione
GetTransactionSummary		Ottieni il riepilogo delle commissioni di transazione.

Prodotti (autenticati)

Method	Argomenti	Descrizione
ListProducts		Elenca i prodotti disponibili.
GetProduct	aProductId: id del prodotto	Ottenere un prodotto specifico.
GetProductBook	aProductId: id del prodotto	Ottieni il libro degli ordini del prodotto.
GetProductCandles	aProductId: id del prodotto	Ottiene le candele OHLCV del prodotto.
GetMarketTrades	aProductId: id del prodotto	Ottieni le ultime operazioni di mercato.
GetBestBidAsk		Ottieni i migliori prezzi bid/ask.

Portfolios

Method	Argomenti	Descrizione
ListPortfolios		Elenca tutti i portafogli.
CreatePortfolio		Crea un nuovo portfolio.

DeletePortfolio		Elimina un portafoglio.
GetPortfolioBreakdown		Recupera i dettagli della ripartizione del portafoglio.
MovePortfolioFunds		Sposta fondi tra portafogli.

Perpetui

Method	Argomenti	Descrizione
GetPerpetualsPortfolioSummary		Recupera il riepilogo del portafoglio perpetuals.
ListPerpetualsPositions		Elenca le posizioni perpetue.
GetPerpetualsPosition		Ottieni una posizione perpetua specifica.

Coinbase | Connetti API WebSocket

Per connettersi all'API WebSocket di Coinbase, è sufficiente creare un nuovo client API Coinbase e collegarlo a TsgcWebSocketClient. Vedere di seguito un esempio:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase();
oCoinbase.Client = oClient;
oClient.Active = true;
```

Coinbase | Subscribe WebSocket Channel

Coinbase offre una varietà di canali a cui è possibile sottoscrivere per ricevere aggiornamenti in tempo reale su dati di mercato, ordini... Di seguito un esempio di come sottoscrivere a un Ticker:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase();
oCoinbase.Client = oClient;
oCoinbase.SubscribeTicker("ETH-USD");

void OnCoinbaseMessage(TObject Sender, string aType, string aRawMessage)
{
    // here you will receive the ticker updates
}
```

Coinbase Pro | Get Market Data

Coinbase offre Dati di Mercato pubblici tramite Endpoint REST; quando si chiama uno di questi endpoint, si ottiene uno snapshot dei dati di mercato richiesti.

Gli endpoint di dati di mercato non richiedono autenticazione e sono quindi liberamente accessibili a tutti gli utenti.

Esempio: per ottenere un'istantanea del libro BTC-USD, effettuare la seguente chiamata

```
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase();  
MessageBox.Show(oCoinbase.REST_API.GetPublicProductBook("BTC-USD"));
```

Coinbase Pro | REST API privata

La Coinbase REST API offre endpoint pubblici e privati. Gli endpoint privati richiedono che i messaggi siano firmati per aumentare la sicurezza delle transazioni.

Per prima cosa è necessario accedere al proprio account Coinbase e creare una nuova API; si otterranno i seguenti valori:

- ApiKey
- ApiSecret

Questi campi devono essere configurati nella proprietà Coinbase del componente client API Coinbase. Una volta configurato, è possibile iniziare a effettuare richieste private all'API REST di Coinbase

```
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase();
oCoinbase.Coinbase.ApiKey = "<your api key>";
oCoinbase.Coinbase.ApiSecret = "<your api secret>";
MessageBox.Show(oCoinbase.REST_API.ListAccounts);
```

Coinbase Pro | Orari delle richieste private

Quando si effettua una richiesta privata a Coinbase, il messaggio viene firmato per aumentare la sicurezza delle richieste. Il messaggio utilizza l'ora locale e la invia all'interno del messaggio firmato; se l'ora locale differisce di più di 30 secondi dai server Coinbase, la richiesta verrà rifiutata. È quindi importante verificare che l'ora locale sia sincronizzata, operazione che può essere eseguita utilizzando il metodo di sincronizzazione dell'ora del proprio sistema operativo.

È possibile verificare l'ora del server Coinbase Pro chiamando il metodo **GetTime**, che restituirà l'ora del server Coinbase Pro

Coinbase Pro | Inserimento ordini

Per piazzare nuovi ordini in Coinbase, è necessario prima disporre delle proprie chiavi API per accedere ai dati privati. Consultare il seguente articolo: [Come utilizzare la REST API privata](#).

Una volta configurate le chiavi API, è possibile iniziare a piazzare ordini

Ordine di Mercato

Inserisce un nuovo Market Order, acquisto di 0,002 contratti di BTC-USD

```
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase();
oCoinbase.Coinbase.ApiKey = "your api key";
oCoinbase.Coinbase.ApiSecret = "your api secret";
oCoinbase.Coinbase.ApiPassphrase = "your passphrase";
MessageBox.Show(oCoinbase.REST_API.PlaceMarketOrder(coisBuy, "BTC-USD", 0.002
, 0
));
```

Ordine limite

Inserisce un nuovo Limit Order, acquista 0,002 contratti di BTC-USD a un limite di prezzo di 10000

```
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase();
oCoinbase.Coinbase.ApiKey = "your api key";
oCoinbase.Coinbase.ApiSecret = "your api secret";
oCoinbase.Coinbase.ApiPassphrase = "your passphrase";
MessageBox.Show(oCoinbase.REST_API.PlaceLimitOrder(coisBuy, "BTC-USD", 0.002,
0,
10000));
```


Coinbase Pro SandBox Account

Coinbase consente di utilizzare un account SandBox con cui è possibile fare trading senza fondi reali. Questo account richiede la creazione di chiavi API diverse rispetto all'account di produzione.

Per utilizzare il conto SandBox, imposti semplicemente la proprietà **Coinbase.SandBox** su **true**, prima di effettuare qualsiasi richiesta alla REST API.

```
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase();
oCoinbase.Coinbase.ApiKey = "your api key";
oCoinbase.Coinbase.ApiSecret = "your api secret";
oCoinbase.Coinbase.SandBox = true;
MessageBox.Show(oCoinbase.REST_API.ListAccounts);
```

API SignalRCore

SignalRCore

ASP.NET Core SignalR è una libreria open source che semplifica l'aggiunta di funzionalità web in tempo reale alle app. La funzionalità web in tempo reale consente al codice lato server di inviare contenuti ai client istantaneamente.

Buoni candidati per SignalR:

- Applicazioni che richiedono aggiornamenti ad alta frequenza dal server. Esempi: giochi, social network, votazioni, aste, mappe e app GPS.
- Dashboard e applicazioni di monitoraggio. Gli esempi includono dashboard aziendali, aggiornamenti immediati delle vendite o avvisi di viaggio.
- App collaborative. Le app di lavagna condivisa e i software per riunioni di team sono esempi di app collaborative.
- App che richiedono notifiche. Social network, email, chat, giochi, avvisi di viaggio e molte altre app utilizzano le notifiche.

Il componente SignalRCore di sgcWebSockets utilizza WebSocket come trasporto per connettersi a un server SignalRCore; se questo trasporto non è supportato, verrà generato un errore.

Hub

SignalRCore utilizza hub per la comunicazione tra client e server. SignalRCore fornisce 2 protocolli hub: il protocollo testuale basato su JSON e il protocollo binario basato su MessagePack. Il componente sgcWebSockets implementa solo il protocollo testuale JSON per comunicare con i server SignalRCore.

Per configurare quale Hub il client utilizzerà, impostare nella proprietà **SignalRCore/Hub** il nome dell'Hub prima che il client si connetta al server.

Connessione

Quando un client apre una nuova connessione al server, invia un messaggio di richiesta contenente il protocollo e la versione del formato. sgcWebSockets invia sempre il protocollo di formato come JSON. Il server risponderà con un errore se il protocollo non è supportato; questo errore può essere gestito tramite l'evento **OnSignalRCoreError**, e se la connessione ha successo, verrà chiamato l'evento **OnSignalRCoreConnect**.

Quando un client si connette a un server SignalRCore, può inviare un **ConnectionId** che identifica il client tra le sessioni, in modo che in caso di disconnessione il client possa riconnettersi al server passando lo stesso **ConnectionId** precedente. Per ottenere un nuovo **ConnectionId**, basta connettersi normalmente al server e si può conoscere il **ConnectionId** tramite **OnBeforeConnectEvent**. Se si desidera riconnettersi al server e passare un **ConnectionId** precedente, utilizzare il metodo **ReConnect** e passare **ConnectionId** come parametro.

Protocollo SignalRCore

Il protocollo SignalR è un protocollo per RPC bidirezionale su qualsiasi trasporto basato su messaggi. Entrambe le parti nella connessione possono invocare procedure sull'altra parte, e le procedure possono restituire zero o più risultati oppure un errore. Esempio: il client può richiedere un metodo al server e il server può richiedere un metodo al client. I seguenti messaggi vengono scambiati tra server e client:

- **HandshakeRequest**: il client invia al server per concordare il formato dei messaggi.
- **HandshakeResponse**: il server risponde al client con una conferma del messaggio **HandshakeRequest** precedente. Contiene un errore se l'handshake non è riuscito.
- **Close**: chiamato dal client o dal server quando una connessione viene chiusa. Contiene un errore se la connessione è stata chiusa a causa di un errore.

- **Invocazione:** il client o il server invia un messaggio a un altro peer per invocare un metodo con o senza argomenti.
- **StreamInvocation:** il client o il server invia un messaggio a un altro peer per invocare un metodo in streaming con o senza argomenti. La risposta verrà suddivisa in elementi distinti.
- **StreamItem:** è una risposta a una precedente StreamInvocation.
- **Completion:** indica che una precedente invocazione o StreamInvocation è stata completata. Può contenere un risultato se il processo è andato a buon fine o un errore in caso di problemi.
- **CancellInvocation:** annulla una precedente richiesta StreamInvocation.
- **Ping:** è un messaggio per verificare se la connessione è ancora attiva.

SignalRCore Encoding

SignalRCore Le consente di utilizzare le seguenti codifiche:

- **JSON:** attualmente l'unica codifica supportata.
- **MessagePack**

Attualmente è supportato solo JSON, sebbene sia possibile utilizzare MessagePack codificando i messaggi inviati tramite una libreria messagepack esterna. Veda la sezione MessagePack di seguito per maggiori informazioni.

La configurazione del protocollo di codifica è definita nella proprietà SignalRCore.Protocol. Per impostazione predefinita il valore è **srcpJSON**.

Authorization

L'autenticazione può essere abilitata per associare un utente a ogni connessione e filtrare quali utenti possono accedere alle risorse. L'autenticazione viene implementata tramite Bearer Token: il client fornisce un token di accesso e il server convalida questo token e lo utilizza per identificare l'utente.

Nelle API Web standard, i bearer token vengono inviati in un Header HTTP, ma quando si utilizzano i websocket, il token viene trasmesso come parametro della query string.

Sono supportati i seguenti metodi:

srcpRequestToken

Se l'autenticazione è abilitata, il flusso è:

1. Prima tenta di ottenere un token valido dal server. Apre una connessione HTTP verso Authentication.RequestToken.URL ed esegue un POST utilizzando i dati di utente e password.
2. Se l'operazione precedente ha avuto successo, viene restituito un token. In caso contrario, viene restituito un errore.
3. Se viene restituito un token, allora apre una nuova connessione HTTP per negoziare una nuova connessione. Qui, il token viene passato come header HTTP.
4. Se l'operazione precedente ha esito positivo, apre una connessione WebSocket e passa il token come parametro della query string.

- **Authentication.Enabled:** se attivo, l'autorizzazione verrà utilizzata prima che venga stabilita una connessione websocket.
- **Authentication.Username:** il nome utente fornito al server per l'autenticazione.
- **Authentication.Password:** la parola segreta fornita al server per l'autenticazione.
- **Authentication.RequestToken.PostFieldUsername:** nome del campo per trasmettere il nome utente (dipende dalla configurazione, verificare la pagina JavaScript HTTP per vedere quale nome viene utilizzato).
- **Authentication.RequestToken.PostFieldPassword:** nome del campo per trasmettere la password (dipende dalla configurazione, verificare la pagina JavaScript HTTP per vedere quale nome viene utilizzato).
- **Authentication.RequestToken.URL:** url in cui viene richiesto il token.
- **Authentication.RequestToken.QueryFieldToken:** nome del parametro della stringa di query utilizzato nella connessione WebSocket.

srcpSetToken

Qui, si passa il token direttamente al server SignalRCore (perché il token è stato ottenuto da un altro server).

- **Authentication.Enabled:** se attivo, verrà utilizzata l'autorizzazione prima che venga stabilita una connessione websocket.
- **Authentication.SetToken.Token:** valore del token ottenuto.

L'Access token può essere inviato come parametro di query (questa è l'opzione predefinita) oppure inviato come HTTP Header come Bearer Token. Utilizzi la proprietà `Authentication.TokenParam` per configurare questo comportamento.

- **srctQuery:** l'`access_token` viene passato nell'URL della query della connessione WebSocket.
- **srctHeader:** l'`access_token` viene passato come intestazione http come Bearer Token.

srcaBasic

Questa opzione utilizza l'**autenticazione di base**; questo metodo di autenticazione richiede la configurazione del componente `SignalRCore` e del [TsgcWebSocketClient](#).

Esempio: se il server richiede l'autenticazione di base e il nome utente è "user" e la password è "secret", configurare i componenti come mostrato di seguito.

```
// websocket client
TsgcWebSocketClient WSClient = new TsgcWebSocketClient();
WSClient.Authentication.Enabled = true;
WSClient.Authentication.Basic.Enabled = true;
WSClient.Authentication.URL.Enabled = false;
WSClient.Authentication.Session.Enabled = false;
WSClient.Authentication.Token.Enabled = false;
WSClient.Authentication.User = "user";
WSClient.Authentication.Password = "secret";
// signalrcore
TsgcWSAPI_SignalRCore Signal = new TsgcWSAPI_SignalRCore();
Signal.SignalRCore.Authentication.Enabled = true;
Signal.SignalRCore.Authentication.Authentication = srcaBasic;
Signal.SignalRCore.Authentication.Username = "user";
Signal.SignalRCore.Authentication.Password = "secret";
Signal.Client = WSClient;
```

Comunicazione tra Client e Server

Esistono tre tipi di interazioni tra server e client:

Invocazioni

Il Caller invia un messaggio al Callee e si aspetta un messaggio che indichi che l'invocazione è stata completata e, facoltativamente, un risultato dell'invocazione

Esempio: il client invoca il metodo `SendMessage` e passa come parametri il nome utente e il messaggio di testo. Invia un ID invocazione per ottenere un messaggio di risultato dal server.

```
SignalRCore.Invoke("SendMessage", new object[] { "John", "Hello All.", "id-000001" });
void OnSignalRCoreCompletion(object Sender, TSignalRCore_Completion Completion)
{
    if (Completion.Error != "")
    {
        MessageBox.Show("Something goes wrong.");
    }
    else
    {
        MessageBox.Show("Invocation Successful!");
    }
}
```

Invocazioni non bloccanti

Il Caller invia un messaggio al Callee e non si aspetta ulteriori messaggi per questa invocazione. Le invocazioni possono essere inviate senza un valore di Invocation ID, il che indica che l'invocazione è "non bloccante".

Esempio: il client invoca il metodo `SendMessage` e passa come parametri il nome utente e il messaggio di testo. Il client non si aspetta alcuna risposta dal server riguardo al risultato dell'invocazione.

```
SignalRCore.Invoke("SendMessage", new object[] { "John", "Hello All." });
```

Invocazioni in streaming

Il Caller invia un messaggio al Callee e si aspetta uno o più risultati restituiti dal Callee seguiti da un messaggio che indica la fine dell'invocazione.

Esempio: il client invoca il metodo `Counter` e richiede 10 numeri con un intervallo di 500 millisecondi.

```
SignalRCore.InvokeStream("Counter", new object[] { 10, 500 }, "id-000002");

void OnSignalRCoreStreamItem(object Sender, TSignalRCore_StreamItem StreamItem, ref bool Cancel)
{
    DoLog("#stream item: " + StreamItem.Item);
}

void OnSignalRCoreCompletion(object Sender, TSignalRCore_Completion Completion)
{
    if (Completion.Error != "")
    {
        MessageBox.Show("Something goes wrong.");
    }
    else
    {
        MessageBox.Show("Invocation Successful!");
    }
}
```

Invocazioni

Per eseguire una singola invocazione, il Caller segue il seguente flusso di base:

```
void Invoke(string aTarget, object[] aArguments, string aInvocationId = "");
void InvokeStream(string aTarget, object[] aArguments, string aInvocationId);
```

Allocare un valore univoco di ID invocazione (stringa arbitraria, scelta dal chiamante) per rappresentare l'invocazione. Chiamare il metodo `Invoke` o `InvokeStream` specificando il Target invocato, gli Arguments e l'InvocationId (se non si invia l'InvocationId, non si riceverà il risultato di completamento).

Se l'invocazione è contrassegnata come non bloccante (vedere "Invocazioni Non Bloccanti" di seguito), ci si fermi qui e si restituisca immediatamente il controllo all'applicazione. Gestire il messaggio `StreamItem` o `Completion` con un ID Invocazione corrispondente.

```
SignalRCore.InvokeStream("Counter", new object[] { 10, 500 }, "id-000002");

void OnSignalRCoreStreamItem(object Sender, TSignalRCore_StreamItem StreamItem, ref bool Cancel)
{
    if (StreamItem.InvocationId == "id-000002")
    {
        DoLog("#stream item: " + StreamItem.Item);
    }
}

void OnSignalRCoreCompletion(object Sender, TSignalRCore_Completion Completion)
{
    if (Completion.InvocationId == "id-000002")
    {
        if (Completion.Error != "")
        {

```

```

        MessageBox.Show("Something goes wrong.");
    }
    else
    {
        MessageBox.Show("Invocation Successful!");
    }
}
}

```

È possibile effettuare una singola chiamata e attendere il completamento.

```

bool InvokeAndWait(string aTarget, object[] aArguments, string aInvocationId, out TSignalRCore_Completion Completion)
bool InvokeStreamAndWait(string aTarget, object[] aArguments, string aInvocationId, out TSignalRCore_Completion Completion)

```

Assegna un valore Invocation ID univoco (stringa arbitraria, scelta dal Caller) per rappresentare l'invocazione. Chiama il metodo `InvokeAndWait` o `InvokeStreamAndWait` contenente il Target invocato, gli Arguments e l'InvocationId. Il programma attenderà finché non viene chiamato l'evento di completamento oppure è stato superato il Time out.

```

TSignalRCore_Completion oCompletion;
if (SignalRCore.InvokeStreamAndWait("Counter", new object[] {10, 500}, "id-000002", out oCompletion))
{
    DoLog("#invoke stream ok: " + oCompletion.Result);
}
else
{
    DoLog("#invocke stream error: " + oCompletion.Error);
}

void OnSignalRCoreStreamItem(object Sender, TSignalRCore_StreamItem StreamItem, ref bool Cancel)
{
    if (StreamItem.InvocationId == "id-000002")
    {
        DoLog("#stream item: " + StreamItem.Item);
    }
}

```

Annullare l'invocazione

Se il client desidera smettere di ricevere messaggi `StreamItem` prima che il server invii un messaggio `Completion`, il client può inviare un messaggio `CancelInvocation` con lo stesso `InvocationId` utilizzato per il messaggio `StreamInvocation` che ha avviato lo stream.

```

void OnSignalRCoreStreamItem(object Sender, TSignalRCore_StreamItem StreamItem, ref bool Cancel)
{
    if (StreamItem.InvocationId == "id-000002")
    {
        Cancel = true;
    }
}

```

Risultati client

Un'invocazione è considerata completata solo quando viene ricevuto il messaggio di completamento. Se il client riceve un'invocazione dal server, verrà chiamato l'evento `OnSignalRCoreInvocation`.

```

void OnSignalRCoreInvocation(object Sender, TSignalRCore_Invocation Invocation)
{
    if (Invocation.Target == "SendMessage")
    {
        // ... your code here ...
    }
}

// Once invocation is completed, call Completion method to inform server invocation is finished.
// If result is successful, then call CompletionResult method:
SignalRCore.CompletionResult("id-000002", "ok");

// If not, then call CompletionError method:

```

```
SignalRCore.CompletionError("id-000002", "Error processing invocation.");
```

Chiudi connessione

Inviato dal client quando una connessione viene chiusa. Contiene un motivo di errore se la connessione è stata chiusa a causa di un errore.

```
SignalRCore.Close("Unexpected message");

// If the server close connection by any reason, OnSignalRCoreClose event will be called.
void OnSignalRCoreClose(object Sender, TSignalRCore_Close Close)
{
    DoLog("#closed: " + Close.Error);
}
```

Ping

Il protocollo SignalR Hub supporta messaggi "Keep Alive" utilizzati per garantire che la connessione di trasporto sottostante rimanga attiva. Questi messaggi aiutano a garantire:

I proxy non chiudono la connessione sottostante durante i periodi di inattività (quando vengono inviati pochi messaggi). Se la connessione sottostante viene interrotta senza essere terminata correttamente, l'applicazione viene informata il prima possibile.

Il comportamento keep alive viene ottenuto chiamando il metodo Ping o abilitando HeartBeat sul client WebSocket. Se il server invia un ping al client, il client invierà automaticamente una risposta e l'evento OnSignalRCoreKeepAlive verrà chiamato.

```
void OnSignalRCoreKeepAlive(object Sender)
{
    DoLog("#keepalive");
}
```

MessagePack

Nella codifica MsgPack del protocollo SignalR, ogni messaggio è rappresentato come un singolo array MsgPack contenente elementi che corrispondono alle proprietà del messaggio del protocollo hub specificato. Gli elementi dell'array possono essere valori primitivi, array (ad es. argomenti del metodo) o oggetti (ad es. valore dell'argomento). Il primo elemento dell'array è il tipo di messaggio.

Faccia riferimento alla [documentazione di MessagePack](#) per vedere come codificare i messaggi inviati.

Ogni volta che viene ricevuto un nuovo messaggio, questo viene inviato nell'evento OnSignalRCoreMessagePack. È possibile accedere al messaggio leggendo il parametro Data Stream. Il parametro JSON è vuoto per impostazione predefinita; se si converte il messaggio MessagePack in JSON, il componente elaborerà il messaggio JSON come se la codifica fosse JSON (quindi verranno generati gli eventi OnSignalRCoreCompletion, OnSignalRCoreInvocation...).

API SignalR

SignalR

Il componente SignalR utilizza WebSocket come trasporto per connettersi a un server SignalR; se questo trasporto non è supportato, verrà generato un errore.

Il componente client SignalR ha una proprietà chiamata `SignalR` dove è possibile impostare i seguenti dati:

- **Hubs:** contiene un elenco degli hub a cui il client si sta abbonando.
- **ProtocolVersion:** la versione del protocollo utilizzata dal client, supporta le versioni del protocollo dalla 1.2 alla 1.5
- **UserAgent:** user agent utilizzato per connettersi al server SignalR.

Il client supporta l'invio di dati di tipo Testo o Binario.

Messaggi degli Hub

L'API Hubs consente di invocare metodi del server dal client e metodi del client dal server. Il protocollo utilizzato per la connessione persistente non è sufficientemente ricco da consentire la semantica RPC (chiamata di procedura remota). Ciò non significa tuttavia che il protocollo utilizzato per le connessioni hub sia completamente diverso da quello usato per le connessioni persistenti. Piuttosto, il protocollo per le connessioni hub è principalmente un'estensione di quello per le connessioni persistenti.

Quando un client invoca un metodo del server, non invia più una stringa a flusso libero come accadeva per le connessioni persistenti. Invece, invia una stringa JSON contenente tutte le informazioni necessarie per invocare il metodo. Ecco un messaggio di esempio che un client invierebbe per invocare un metodo del server:

```
WriteData("{\"H\":\"chathub\",\"M\":\"Send\",\"A\":[\"CSharp Client\",\"Test message\"],\"I\":0});
```

Il payload ha le seguenti proprietà:

I – identificatore di invocazione – consente di abbinare le risposte alle richieste

H – il nome dell'hub

M – il nome del metodo

A – argomenti (un array, può essere vuoto se il metodo non ha parametri)

Se l'argomento stringa contiene **doppie virgolette** sostituire " con \"

Esempio: se l'argomento è {"test":1}, inviare l'argomento come {"test\\":1}

```
WriteData("{\"H\":\"chathub\",\"M\":\"Send\",\"A\":[\"test\\\":1\"],\"I\":0});
```

Authorization

L'autenticazione può essere abilitata per associare un utente a ogni connessione e filtrare quali utenti possono accedere alle risorse. L'autenticazione viene implementata tramite Bearer Token: il client fornisce un token di accesso e il server convalida questo token e lo utilizza per identificare l'utente.

Attualmente sono supportati solo i Bearer Token:

Qui si passa il token direttamente al server Signal (poiché il token è stato ottenuto da un altro server).

- **Authentication.Enabled:** se attivo, verrà utilizzata l'autorizzazione prima che venga stabilita una connessione websocket.
- **Authentication.Authentication:** Sono supportati 2 tipi di autenticazione: bearer token o cookie. Entrambi richiedono un modo esterno per ottenere i valori richiesti.
 - **BearerToken:** valore del token ottenuto.
 - **Cookie:** impostare il valore del cookie richiesto.

```
TsgcWSAPI_Signal oSignalR = new TsgcWSAPI_Signal();
oSignalR.SignalR.Enabled = true;
oSignalR.SignalR.Authentication = srcBearerToken;
oSignalR.SignalR.BearerToken.Token = "token here";
```

Il componente ha i seguenti eventi:

OnSignalRConnect

Questo evento viene sollevato quando il client si connette con successo al server.

OnSignalRDisconnect

Questo evento viene generato quando il client si disconnette dal server.

OnSignalRError

Questo evento viene chiamato quando si verifica un errore nella connessione WebSocket.

OnSignalRMessage

Il protocollo utilizzato per la connessione persistente è abbastanza semplice. I messaggi inviati al server sono semplici stringhe grezze, senza un formato specifico. I messaggi inviati al client sono più strutturati. Le proprietà presenti nel messaggio sono le seguenti:

C – id del messaggio, presente per tutti i messaggi non KeepAlive
M – un array contenente i dati effettivi.

```
{"C": "d-9B7A6976-B, 2 | C, 2", "M": ["Welcome!"]}
```

OnSignalRBinary

Questo evento viene chiamato quando vengono ricevuti dati binari dal server.

OnSignalRResult

Quando viene richiamato un metodo server, il server restituisce una conferma che l'invocazione è completata inviando l'ID di invocazione al client e – se il metodo ha restituito un valore – il valore restituito, oppure – se l'invocazione del metodo è fallita – l'errore.

Di seguito sono riportati esempi di risultati di una chiamata a un metodo del server:

```
{"I": "0"}
```

Un metodo server void il cui identificatore di invocazione era "0" è stato completato con successo.

```
{"I": "0", "R": 42}
```

Un metodo server che restituisce un numero il cui identificatore di invocazione era "0" è stato completato correttamente e ha restituito il valore 42.

```
{"I": "0", "E": "Error occurred"}
```

OnSignalRKeepAlive

Questo evento viene sollevato quando viene ricevuto un messaggio KeepAlive dal server.

API Kraken

Kraken

Panoramica

WebSockets API offre aggiornamenti dei dati di mercato in tempo reale. WebSockets è un protocollo bidirezionale che offre i dati in tempo reale più veloci, consentendo di creare applicazioni in tempo reale. I tipi di messaggi pubblici presentati di seguito non richiedono autenticazione. I messaggi di dati privati possono essere sottoscritti su un endpoint autenticato separato.

Kraken offre anche una REST API con dati di mercato pubblici e dati utente privati (che richiedono autenticazione).

Configurazione

L'API privata richiede la creazione di una chiave API dal proprio account Kraken. Kraken consente un ambiente di test sul protocollo WebSocket; abilitare la proprietà Beta da Kraken Property per utilizzare questa funzionalità beta.

API supportate

- [API WebSockets pubblica](#): si connette a un server WebSocket pubblico.
- [WebSockets Private API](#): si connette a un server WebSocket privato e richiede una API Key e un API Secret per autenticarsi presso il server.
- [REST Public API](#): si connette a un server REST pubblico.
- [REST Private API](#): si connette a un server REST pubblico e richiede una Chiave API e un Segreto API per autenticarsi presso il server.

Esempi Kraken

Come Connettersi a un Server WebSocket Pubblico

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Kraken oKraken = new TsgcWSAPI_Kraken();
oKraken.Client = oClient;
oClient.Active = true;
```

Come connettersi a un server WebSocket privato

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Kraken oKraken = new TsgcWSAPI_Kraken();
oKraken.Kraken.ApiKey = "your api key";
oKraken.Kraken.ApiSecret = "your api secret";
oKraken.Client = oClient;
oClient.Active = true;
```

Come ottenere il Ticker dall'API REST

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Kraken oKraken = new TsgcWSAPI_Kraken();
oKraken.Client = oClient;
MessageBox(oKraken.GetTicker("XBTUSD"));
```

Metodi REST API

Endpoint Pubblici

Method	Argomenti	Description
GetSystemStatus		Ottieni lo stato corrente del sistema.

Endpoint privati

Method	Argomenti	Description
GetExtendedBalance		Ottieni informazioni estese sul saldo.
AmendOrder		Modifica un ordine esistente.
CancelAllOrders		Annulla tutti gli ordini aperti.
CancelAllOrdersAfter		Dead man's switch - annulla tutti gli ordini dopo il timeout.
EditOrder		Modifica un ordine esistente.
AddOrderBatch		Aggiunta in batch di più ordini.
CancelOrderBatch		Annullamento batch di più ordini.
GetWithdrawalMethods		Ottieni i metodi di prelievo disponibili.
GetWithdrawalAddresses		Ottenere gli indirizzi di prelievo.

Come ottenere il saldo dell'account dalla REST API

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Kraken oKraken = new TsgcWSAPI_Kraken();
oKraken.Kraken.ApiKey = "your api key";
oKraken.Kraken.ApiSecret = "your api secret";
oKraken.Client = oClient;
MessageBox.Show(oKraken.GetAccountBalance());
```

API Kraken | WebSockets Public API

Connessione

URL: `wss://ws.kraken.com (v1)` o `wss://ws.kraken.com/v2 (v2, consigliato)`

Il componente ora supporta WebSocket API v2 tramite la proprietà **Version** (predefinito: 2). Impostare `Kraken.Version := 1` per utilizzare l'endpoint v1 legacy.

Una volta aperto il socket è possibile sottoscrivere a un canale pubblico inviando un messaggio di richiesta di sottoscrizione.

Considerazioni Generali

- Tutti i messaggi inviati e ricevuti tramite WebSockets sono codificati in formato JSON
- Tutti i campi in virgola mobile (inclusi i timestamp) sono quotati per preservare la precisione.
- Il formato di ogni coppia negoziabile è A/B, dove A e B sono ISO 4217-A3 per gli asset standardizzati e il simbolo unico popolare se non standardizzato.
- I timestamp non devono essere considerati univoci né considerati come alias degli id di transazione. Inoltre, la granularità dei timestamp non è rappresentativa dei tassi di transazione.

Coppie supportate

ADA/CAD, ADA/ETH, ADA/EUR, ADA/USD, ADA/XBT, ATOM/CAD, ATOM/ETH, ATOM/EUR, ATOM/USD, ATOM/XBT, BCH/EUR, BCH/USD, BCH/XBT, DASH/EUR, DASH/USD, DASH/XBT, EOS/ETH, EOS/EUR, EOS/USD, EOS/XBT, GNO/ETH, GNO/EUR, GNO/USD, GNO/XBT, QTUM/CAD, QTUM/ETH, QTUM/EUR, QTUM/USD, QTUM/XBT, USDT/USD, ETC/ETH, ETC/XBT, ETC/EUR, ETC/USD, ETH/XBT, ETH/CAD, ETH/EUR, ETH/GBP, ETH/JPY, ETH/USD, LTC/XBT, LTC/EUR, LTC/USD, MLN/ETH, MLN/XBT, REP/ETH, REP/XBT, REP/EUR, REP/USD, STR/EUR, STR/USD, XBT/CAD, XBT/EUR, XBT/GBP, XBT/JPY, XBT/USD, BTC/CAD, BTC/EUR, BTC/GBP, BTC/JPY, BTC/USD, XDG/XBT, XLM/XBT, DOGE/XBT, STR/XBT, XLM/EUR, XLM/USD, XMR/XBT, XMR/EUR, XMR/USD, XRP/XBT, XRP/CAD, XRP/EUR, XRP/JPY, XRP/USD, ZEC/XBT, ZEC/EUR, ZEC/JPY, ZEC/USD, XTZ/CAD, XTZ/ETH, XTZ/EUR, XTZ/USD, XTZ/XBT

Metodi

Ping

Il client può inviare un ping al server per verificare se la connessione è attiva; il server risponde con un pong.

Si tratta di un ping a livello applicativo, al contrario del ping predefinito dello standard WebSockets che è avviato dal server

Ticker

Le informazioni del ticker includono i prezzi di best ask e best bid, il volume nelle 24 ore, l'ultimo prezzo di trade, il prezzo medio ponderato per volume, ecc. per una determinata coppia di valute. Un messaggio ticker viene pubblicato ogni volta che avviene un trade o un gruppo di trade.

Sottoscrivere un ticker chiamando il metodo `SubscribeTicker`:

Se la sottoscrizione ha esito positivo, verrà chiamato l'evento **OnKrakenSubscribed**:

Annulla l'abbonamento chiamando il metodo `UnSubscribeTicker`:

Se l'annullamento della sottoscrizione ha esito positivo, verrà chiamato l'evento OnKrakenUnSubscribed:

Se si verifica un errore durante il tentativo di sottoscrizione / annullamento della sottoscrizione, verrà chiamato l'evento OnKrakenSubscriptionError.

Gli aggiornamenti del ticker verranno notificati nell'evento OnKrakenData.

```
[
  0,
  {
    "a": [
      "5525.40000",
      1,
      "1.000"
    ],
    "b": [
      "5525.10000",
      1,
      "1.000"
    ],
    "c": [
      "5525.10000",
      "0.00398963"
    ],
    "v": [
      "2634.11501494",
      "3591.17907851"
    ],
    "p": [
      "5631.44067",
      "5653.78939"
    ],
    "t": [
      11493,
      16267
    ],
    "l": [
      "5505.00000",
      "5505.00000"
    ],
    "h": [
      "5783.00000",
      "5783.00000"
    ],
    "o": [
      "5760.70000",
      "5763.40000"
    ]
  },
  "ticker",
  "XBT/USD"
]
```

OHLC

Quando si sottoscrive OHLC, viene inviato uno snapshot dell'ultima candela valida (indipendentemente dall'orario di fine), seguito dagli aggiornamenti alla candela in corso. Ad esempio, se si sottoscrive una candela da 1 minuto e non ci sono state transazioni per 5 minuti, verrà pubblicato uno snapshot dell'ultima candela da 1 minuto di 5 minuti fa. L'orario di fine può essere utilizzato per determinare che si tratta di una candela precedente.

Sottoscrivere un OHLC chiamando il metodo SubscribeOHLC; è necessario passare la coppia e l'intervallo.

Se la sottoscrizione ha successo, verrà chiamato l'evento OnKrakenSubscribed:

Disiscrizione tramite il metodo UnSubscribeOHLC:

Se l'annullamento della sottoscrizione ha esito positivo, verrà chiamato l'evento OnKrakenUnSubscribed:

Se si verifica un errore durante il tentativo di sottoscrizione / annullamento della sottoscrizione, verrà chiamato l'evento OnKrakenSubscriptionError.

Gli aggiornamenti OHLC verranno notificati nell'evento OnKrakenData.

```
[
  42,
  [
    "1542057314.748456",
    "1542057360.435743",
    "3586.70000",
    "3586.70000",
    "3586.60000",
    "3586.60000",
    "3586.68894",
    "0.03373000",
  ],
  2,
  "ohlcv-5",
  "XBT/USD"
]
```

Trade

Feed di trading per una coppia di valute.

Sottoscrivere il feed Trade chiamando il metodo SubscribeTrade.

Se la sottoscrizione ha successo, verrà chiamato l'evento OnKrakenSubscribed:

Annullare la sottoscrizione chiamando il metodo UnSubscribeTrade:

Se l'annullamento della sottoscrizione ha esito positivo, verrà chiamato l'evento OnKrakenUnSubscribed:

Se si verifica un errore durante il tentativo di sottoscrizione / annullamento della sottoscrizione, verrà chiamato l'evento OnKrakenSubscriptionError.

Gli aggiornamenti delle operazioni verranno notificati nell'evento OnKrakenData.

```
[
  0,
  [
    [
      "5541.20000",
      "0.15850568",
      "1534614057.321597",
      "s",
      "1",
      ""
    ],
    [
      "6060.00000",
      "0.02455000",
      "1534614057.324998",
      "b",
      "1",
      ""
    ]
  ]
],
```

```

    "trade",
    "XBT/USD"
  ]

```

Libro

Livelli dell'order book. Alla sottoscrizione verrà pubblicato uno snapshot alla profondità specificata; dopo lo snapshot verranno pubblicati gli aggiornamenti dei livelli.

Iscriversi a un Book chiamando il metodo `SubscribeBook`; è necessario passare la coppia e la profondità.

Se la sottoscrizione ha successo, verrà chiamato l'evento `OnKrakenSubscribed`:

Annullare l'iscrizione chiamando il metodo `UnSubscribeBook`:

Se l'annullamento della sottoscrizione ha esito positivo, verrà chiamato l'evento `OnKrakenUnSubscribed`:

Se si verifica un errore durante il tentativo di sottoscrizione / annullamento della sottoscrizione, verrà chiamato l'evento `OnKrakenSubscriptionError`.

Gli aggiornamenti del book verranno notificati nell'evento `OnKrakenData`.

```

[
  0,
  {
    "as": [
      [
        "5541.30000",
        "2.50700000",
        "1534614248.123678"
      ],
      [
        "5541.80000",
        "0.33000000",
        "1534614098.345543"
      ],
      [
        "5542.70000",
        "0.64700000",
        "1534614244.654432"
      ]
    ],
    "bs": [
      [
        "5541.20000",
        "1.52900000",
        "1534614248.765567"
      ],
      [
        "5539.90000",
        "0.30000000",
        "1534614241.769870"
      ],
      [
        "5539.50000",
        "5.00000000",
        "1534613831.243486"
      ]
    ]
  },
  "book-100",
  "XBT/USD"
]

```


Spread

Feed spread per mostrare il miglior prezzo bid e ask per la coppia di asset sottoscritta. Il volume bid e il volume ask fanno parte del messaggio.

Iscriversi al feed Spread chiamando il metodo `SubscribeSpread`.

Se la sottoscrizione ha successo, verrà chiamato l'evento `OnKrakenSubscribed`:

Annullare la sottoscrizione chiamando il metodo `UnSubscribeSpread`:

Se l'annullamento della sottoscrizione ha esito positivo, verrà chiamato l'evento `OnKrakenUnSubscribed`:

Se si verifica un errore durante il tentativo di sottoscrizione / annullamento della sottoscrizione, verrà chiamato l'evento `OnKrakenSubscriptionError`.

Gli aggiornamenti dello spread verranno notificati nell'evento `OnKrakenData`.

```
[
  0,
  [
    "5698.40000",
    "5700.00000",
    "1542057299.545897",
    "1.01234567",
    "0.98765432"
  ],
  "spread",
  "XBT/USD"
]
```

Altri Metodi

È possibile sottoscrivere/annullare la sottoscrizione di tutti i canali con un unico metodo:

Il valore dell'intervallo OHLC è 1 se tutti i canali sono sottoscritti.

Eventi

OnConnect: quando il client WebSocket è connesso al server.

OnKrakenConnect: chiamato dopo la connessione websocket riuscita e quando il server invia lo stato del sistema.

OnKrakenSystemStatus: chiamato quando lo stato del sistema cambia.

OnKrakenSubscribed: chiamato dopo una sottoscrizione riuscita a un canale.

OnKrakenUnSubscribed: chiamato dopo una disiscrizione riuscita da un canale.

OnKrakenSubscriptionError: chiamato se si verifica un errore durante il tentativo di sottoscrizione / annullamento della sottoscrizione.

OnKrakenData: chiamato ogni volta che un'iscrizione a un canale ha un aggiornamento.

API Kraken | WebSockets Private API

Connessione

URL: `wss://ws-auth.kraken.com (v1)` o `wss://ws-auth.kraken.com/v2 (v2, raccomandato)`
 Il componente ora supporta WebSocket API v2 tramite la proprietà **Version** (predefinito: 2).

Una volta aperto il socket, è possibile iscriversi ai canali di dati privati inviando un messaggio di richiesta di iscrizione autenticato.

Autenticazione

Il client API deve richiedere un "token" di autenticazione tramite il seguente endpoint REST API "GetWebSocketsToken" per connettersi agli endpoint privati WebSockets. Il token deve essere utilizzato entro 15 minuti dalla creazione. Il token non scade finché viene mantenuta una connessione a un messaggio privato dell'API WebSockets (openOrders o ownTrades).

Per ottenere un Token WebSockets, è necessario impostare una chiave API e un segreto API nel componente Kraken Options; la chiave API è fornita da Kraken nel proprio account

```
Kraken.ApiKey = "api key";
Kraken.ApiSecret = "api secret";
```

Metodi

OwnTrades

Ottenere un elenco delle proprie transazioni; alla prima sottoscrizione si ottiene un elenco delle ultime 50 transazioni

```
SubscribeOwnTrades();
```

In seguito, è possibile annullare la sottoscrizione da OwnTrades chiamando il metodo UnSubscribeOwnTrades

```
UnSubscribeOwnTrades();
```

Esempio di risposta dal server

```
[
  [
    {
      "TDLH43-DVQXD-2KHVYY": {
        "cost": "1000000.00000",
        "fee": "600.00000",
        "margin": "0.00000",
        "ordertxid": "TDLH43-DVQXD-2KHVYY",
        "ordertype": "limit",
        "pair": "XBT/EUR",
        "postxid": "OGTT3Y-C6I3P-XRI6HX",
        "price": "100000.00000",
        "time": "1560520332.914664",
        "type": "buy",
        "vol": "1000000000.00000000"
      }
    }
  ],
  "ownTrades"
]
```

Ordini aperti

Feed per visualizzare tutti gli ordini aperti appartenenti all'utente autenticato tramite chiave API. Lo snapshot iniziale fornirà l'elenco di tutti gli ordini aperti e successivamente verranno inviati gli aggiornamenti all'elenco degli ordini aperti. Per gli aggiornamenti di cambio di stato, come 'closed', i campi orderid e status saranno presenti nel payload

```
SubscribeOpenOrders();
```

Successivamente è possibile annullare la sottoscrizione da OpenOrders chiamando il metodo UnSubscribeOpenOrders

```
UnSubscribeOpenOrders();
```

Esempio di risposta dal server

```
[
  [
    {
      "OGTT3Y-C6I3P-XRI6HX": {
        "cost": "0.00000",
        "descr": {
          "close": "",
          "leverage": "0:1",
          "order": "sell 0.00001000 XBT/EUR @ limit 9.00000 with 0:1 leverage",
          "ordertype": "limit",
          "pair": "XBT/EUR",
          "price": "9.00000",
          "price2": "0.00000",
          "type": "sell"
        },
        "expiretm": "0.000000",
        "fee": "0.00000",
        "limitprice": "9.00000",
        "misc": "",
        "oflags": "fcib",
        "opentm": "0.000000",
        "price": "9.00000",
        "refid": "OKIVMP-5GVZN-Z2D2UA",
        "starttm": "0.000000",
        "status": "open",
        "stopprice": "0.000000",
        "userref": 0,
        "vol": "0.00001000",
        "vol_exec": "0.00000000"
      }
    }
  ],
  "openOrders"
]
```

Aggiungi Ordine

Invia un nuovo Ordine a Kraken

```
TsgcWSKrakenOrder oKrakenOrder = new TsgcWSKrakenOrder();
oKrakenOrder.Pair = "XBT/USD";
oKrakenOrder._Type = kosBuy;
oKrakenOrder.OrderType = kotMarket;
oKrakenOrder.Volume = 1;
AddOrder(oKrakenOrder);
```

Elenco dei parametri dell'Order

```
pair = asset pair
type = type of order (buy/sell)
ordertype = order type:
  market
  limit (price = limit price)
  stop-loss (price = stop loss price)
  take-profit (price = take profit price)
```

```

stop-loss-profit (price = stop loss price, price2 = take profit price)
stop-loss-profit-limit (price = stop loss price, price2 = take profit price)
stop-loss-limit (price = stop loss trigger price, price2 = triggered limit price)
take-profit-limit (price = take profit trigger price, price2 = triggered limit price)
trailing-stop (price = trailing stop offset)
trailing-stop-limit (price = trailing stop offset, price2 = triggered limit offset)
stop-loss-and-limit (price = stop loss price, price2 = limit price)
settle-position
price = price (optional. dependent upon ordertype)
price2 = secondary price (optional. dependent upon ordertype)
volume = order volume in lots
leverage = amount of leverage desired (optional. default = none)
oflags = comma delimited list of order flags (optional):
    viqc = volume in quote currency (not available for leveraged orders)
    fcib = prefer fee in base currency
    fciq = prefer fee in quote currency
    nompp = no market price protection
    post = post only order (available when ordertype = limit)
starttm = scheduled start time (optional):
    0 = now (default)
    +<n> = schedule start time <n> seconds from now
    <n> = unix timestamp of start time
expiretm = expiration time (optional):
    0 = no expiration (default)
    +<n> = expire <n> seconds from now
    <n> = unix timestamp of expiration time
userref = user reference id. 32-bit signed number. (optional)
validate = validate inputs only. do not submit order (optional)
optional closing order to add to system when order gets filled:
    close[ordertype] = order type
    close[price] = price
    close[price2] = secondary price

```

Esempio di risposta dal server

```

{
  "descr": "buy 0.01770000 XBTUSD @ limit 4000",
  "event": "addOrderStatus",
  "status": "ok",
  "txid": "ONPNXH-KMKMU-F4MR5V"
}

```

Annulla Ordine

Annullamento ordine

```
CancelOrder("Order Id");
```

Esempio di risposta dal server

```

{
  "event": "cancelOrderStatus",
  "status": "ok"
}

```

API Kraken | REST Public API

Connessione

URL: <https://api.kraken.com>

La Public API di Kraken non richiede alcuna autenticazione.

Configurazione

L'unica configurazione riguarda se abilitare un log per le richieste HTTP REST. Abilitare HTTPLogOptions se si desidera salvare in un file di testo log tutte le richieste/risposte HTTP

Eventi

OnKrakenHTTPException: questo evento viene chiamato se si verifica un'eccezione durante una richiesta HTTP dall'API REST.

Metodi

GetServerTime

Questo metodo aiuta ad approssimare il differenziale di orario tra server e client. Restituisce l'ora in formato Unix.

```
{"error": [], "result": {"unixtime": 1586705546, "rfc1123": "Sun, 12 Apr 20 15:32:26 +0000"}}
```

GetAssets

Restituisce informazioni sugli asset

```
{"error": [], "result": {"ADA": {"aclass": "currency", "altname": "ADA", "decimals": 8, "display_decimals": 6}}}}
```

GetAssetPairs

Restituisce informazioni su una coppia di asset

```
Kraken.REST_API.GetAssetPairs("XBTUSD");
```

GetTicker

Restituisce informazioni sul ticker

```
Kraken.REST_API.GetTicker("XBTUSD");
```

GetOHLC

Restituisce i dati Open-High-Low-Close.

```
Kraken.REST_API.GetOHLC("XBTUSD");
```

GetOrderBook

Restituisce la coppia di array nome e profondità di mercato.

```
Kraken.REST_API.GetOrderBook("XBTUSD");
```

GetTrades

Restituisce i dati sulle transazioni recenti di una coppia.

```
Kraken.REST_API.GetTrades("XBTUSD");
```

GetSpread

Restituisce i dati recenti di spread di una coppia.

```
Kraken.REST_API.GetSpread("XBTUSD");
```

API Kraken | REST Private API

Connessione

URL: <https://api.kraken.com>

Autenticazione

La REST Private API richiede una API Key e un API Secret, questi valori sono forniti da Kraken nel suo account.

```
Kraken.ApiKey = "api key";  
Kraken.ApiSecret = "api secret";
```

Metodi

GetAccountBalance

Restituisce il saldo del tuo account.

```
Kraken.REST_API.GetAccountBalance();
```

GetTradeBalance

Restituisce informazioni sulle proprie operazioni.

```
Kraken.REST_API.GetTradeBalance();
```

GetOpenOrders

Restituisce un elenco degli ordini aperti.

```
Kraken.REST_API.GetOpenOrders();
```

GetClosedOrders

Restituisce un elenco degli ordini chiusi.

```
Kraken.REST_API.GetClosedOrders();
```

QueryOrders

Interroga le informazioni su un ordine.

```
Kraken.REST_API.QueryOrders("1234");
```

GetTradesHistory

Restituisce un array di informazioni sulle transazioni.

```
Kraken.REST_API.GetTradesHistory();
```

QueryTrades

Interrogare informazioni su una transazione.

```
Kraken.REST_API.QueryTrades("1234");
```

GetOpenPositions

Restituisce le informazioni sulla posizione.

```
Kraken.REST_API.GetOpenPositions("1234");
```

GetLedgers

Restituisce un array associativo di informazioni sui registri contabili.

```
Kraken.REST_API.GetLedgers();
```

QueryLedgers

Restituisce un array associativo di informazioni sui registri contabili.

```
Kraken.REST_API.QueryLedgers("1234");
```

GetTradeVolume

Restituisce informazioni sul volume degli scambi.

```
Kraken.REST_API.GetTradeVolume();
```

AddExport

Aggiunge una nuova esportazione di report.

```
Kraken.REST_API.AddExport("Report All Trades");
```

ExportStatus

Ottieni lo Stato dei report

```
Kraken.REST_API.ExportStatus();
```


RetrieveExport

Ottieni report tramite ID report.

```
Kraken.REST_API.RetrieveExport("GOCO");
```

RemoveExport

Rimuovi report tramite ID report.

```
Kraken.REST_API.RemoveExport("GOCO");
```

Aggiungi Ordine

Aggiunge un nuovo ordine

```
pair = asset pair
type = type of order (buy/sell)
ordertype = order type:
    market
    limit (price = limit price)
    stop-loss (price = stop loss price)
    take-profit (price = take profit price)
    stop-loss-profit (price = stop loss price, price2 = take profit price)
    stop-loss-profit-limit (price = stop loss price, price2 = take profit price)
    stop-loss-limit (price = stop loss trigger price, price2 = triggered limit price)
    take-profit-limit (price = take profit trigger price, price2 = triggered limit price)
    trailing-stop (price = trailing stop offset)
    trailing-stop-limit (price = trailing stop offset, price2 = triggered limit offset)
    stop-loss-and-limit (price = stop loss price, price2 = limit price)
    settle-position
price = price (optional. dependent upon ordertype)
price2 = secondary price (optional. dependent upon ordertype)
volume = order volume in lots
leverage = amount of leverage desired (optional. default = none)
oflags = comma delimited list of order flags (optional):
    viqc = volume in quote currency (not available for leveraged orders)
    fcib = prefer fee in base currency
    fciq = prefer fee in quote currency
    nompp = no market price protection
    post = post only order (available when ordertype = limit)
starttm = scheduled start time (optional):
    0 = now (default)
    +n = schedule start time n seconds from now
    n = unix timestamp of start time
expiretm = expiration time (optional):
    0 = no expiration (default)
    +n = expire n seconds from now
    n = unix timestamp of expiration time
userref = user reference id. 32-bit signed number. (optional)
validate = validate inputs only. do not submit order (optional)
optional closing order to add to system when order gets filled:
    close[ordertype] = order type
    close[price] = price
    close[price2] = secondary price
```

```
TsgcHTTPKrakenOrder oKrakenOrder = new TsgcHTTPKrakenOrder();
oKrakenOrder.Pair = "XBT/USD";
oKrakenOrder._Type = koshBuy;
oKrakenOrder.OrderType = kothMarket;
oKrakenOrder.Volume = 1;
Kraken.REST_API.AddOrder(oKrakenOrder);
```

CancelOrder

Annulla un ordine aperto tramite id

```
Kraken.REST_API.CancelOrder("1234");
```

API Kraken Futures

Kraken Futures

Panoramica

La **REST API** consente di accedere in modo sicuro ai metodi del proprio account Kraken Futures. Esempi di metodi REST API:

- richiedi informazioni sul prezzo corrente o storico
- verificare il saldo del proprio account e il PnL
- i parametri del margine e le soglie di liquidazione stimate
- piazzare o annullare ordini (individualmente o in batch)
- visualizza i Suoi ordini aperti
- posizioni aperte o cronologia delle operazioni
- richiedere il prelievo di un asset digitale

Questi metodi sono chiamati "endpoint" e sono spiegati nella sezione REST API.

La **Websocket API** Le consente di stabilire in modo sicuro un canale di comunicazione con la piattaforma Kraken Futures per ricevere informazioni in tempo reale. Questo consente di ascoltare gli aggiornamenti invece di inviare continuamente richieste. Questi canali sono chiamati sottoscrizioni.

Alcuni degli endpoint consentono di eseguire attività sensibili, come l'avvio di un prelievo di asset digitali. Per accedere a questi endpoint in modo sicuro, l'API utilizza tecniche di cifratura sviluppate dalla National Security Agency.

Configurazione

Per utilizzare l'API, è necessario generare una coppia di **API keys** univoche (se si desidera accedere alle API private):

1. Accedere al proprio **account Kraken Futures**.
2. Fare clic sul proprio nome nell'angolo in alto a destra.
3. Selezionare "Impostazioni" dal menu a discesa.
4. Selezionare la scheda "Create Key" nel pannello API.
5. Premere il pulsante "Create Key".
6. Visualizzare le chiavi pubbliche e private e annotarle in un posto sicuro.

Copiare le chiavi pubblica e privata nella proprietà **KrakenOptions** del componente.

```
KrakenOptions.ApiKey  
KrakenOptions.ApiSecret
```

API supportate

- **API WebSocket pubblica**: si connette a un server WebSocket pubblico.
- **API WebSockets Privata**: si connette a un server WebSocket privato e richiede una chiave API e un segreto API per autenticarsi sul server.
- **REST Public API**: si connette a un server REST pubblico.
- **REST Private API**: si connette a un server REST pubblico e richiede una chiave API e un segreto API per l'autenticazione con il server.

Metodi REST API

Endpoint privati

Method	Argomen- ti	Description
BatchOrder		Invia ordini in batch (inserimento/ annullamento/modifica).
GetOrderStatus		Recupera lo stato di ordini specifici.
GetPNL-CurrencyPreferences		Recupera le preferenze di valuta per il PNL.
SetPNL-CurrencyPreference		Imposta la preferenza di valuta PNL per un simbolo.
GetLeverageSettings		Ottenere le preferenze di leva finanziaria.
SetLeverageSettings		Imposta la leva finanziaria per un simbolo.

API Kraken Futures | WebSockets Public API

Connessione

URL: `wss://futures.kraken.com/ws/v1`

Una volta aperto il socket è possibile sottoscrivere a un canale pubblico inviando un messaggio di richiesta di sottoscrizione.

Metodi

Ticker

Questo endpoint restituisce i dati di mercato correnti per tutti i contratti Futures e gli indici attualmente quotati. Non è richiesta autenticazione.

Sottoscrivere un ticker chiamando il metodo **SubscribeTicker**:

Se la sottoscrizione ha esito positivo, verrà chiamato l'evento **OnKrakenFuturesSubscribed**:

Annullare la sottoscrizione chiamando il metodo **UnSubscribeTicker**:

Se l'annullamento dell'iscrizione ha esito positivo, verrà chiamato l'evento **OnKrakenFuturesUnSubscribed**:

Se si verifica un errore durante la sottoscrizione o la cancellazione della sottoscrizione, verrà chiamato l'evento **OnKrakenFuturesError**.

Gli aggiornamenti del ticker verranno notificati nell'evento **OnKrakenData**.

```
{  "result": "success",
```

```
  "tickers": [
```

```
  {
```

```
    "tag": "perpetual",
```

```
    "pair": "XBT:USD",
```

```
    "symbol": "pi_xbtusd",
```

```
    "markPrice": 9520.2,
```

```
    "bid": 9520,
```

```
    "bidSize": 30950,
```

```
    "ask": 9520.5,
```

```

"askSize": 3779,

"vol24h": 68238712,

"openInterest": 29308193,

"open24h": 10137,

"last": 9521,

"lastTime": "2020-06-03T08:14:26.624Z",

"lastSize": 1,

"suspended": false,

"fundingRate": 4.943012455e-9,

"fundingRatePrediction": 4.414499215e-9

}

{

"tag": "quarter",

"pair": "XBT:USD",

"symbol": "fi_xbtusd_200925",

"markPrice": 9659.8,

"bid": 9659.5,

"bidSize": 6480,

"ask": 9660,

"askSize": 17100,

"vol24h": 4562580,

"openInterest": 3573325,

"open24h": 10370.5,

"last": 9660,

"lastTime": "2020-06-03T08:10:37.800Z",

"lastSize": 5000,

"suspended": false

},

{

"symbol": "in_xbtusd",

"last": 9519,

"lastTime": "2020-06-03T08:14:49.000Z"

}

```

```

],
"serverTime": "2020-06-03T08:14:49.865Z"
}

```

Trade

Il feed delle operazioni restituisce informazioni sulle transazioni eseguite. Sottoscrivere il feed Trade chiamando il metodo **SubscribeTrade**.

Se la sottoscrizione ha esito positivo, verrà chiamato l'evento **OnKrakenFuturesSubscribed**:

Annulli la sottoscrizione richiamando il metodo **UnSubscribeTrade**:

Se l'annullamento dell'iscrizione ha esito positivo, verrà chiamato l'evento **OnKrakenFuturesUnSubscribed**:

Se si verifica un errore durante la sottoscrizione o la cancellazione della sottoscrizione, verrà chiamato l'evento **OnKrakenFuturesError**.

Gli aggiornamenti degli scambi saranno notificati nell'evento **OnKrakenData**.

```

{  "feed": "trade_snapshot",

  "product_id": "PI_XBTUSD",
  "trades": [
    {
      "feed": "trade",
      "product_id": "PI_XBTUSD",
      "uid": "caa9c653-420b-4c24-a9f1-462a054d86f1",
      "side": "sell",
      "type": "fill",
      "seq": 655508,
      "time": 1612269657781,
      "qty": 440,
      "price": 34893
    },
    {

```

```

"feed": "trade",
"product_id": "PI_XBTUSD",
"uid": "45ee9737-1877-4682-bc68-e4ef818ef88a",
"side": "sell",
"type": "fill",
"seq": 655507,
"time": 1612269656839,
"qty": 9643,
"price": 34891
}
]
}

```

Libro

Questo feed restituisce informazioni sull'order book.

Sottoscriva un Book chiamando il metodo `SubscribeBook`, deve passare il Symbol.

Se la sottoscrizione ha esito positivo, verrà chiamato l'evento **OnKrakenFuturesSubscribed**:

Annullare la sottoscrizione chiamando il metodo **UnSubscribeBook**:

Se l'annullamento dell'iscrizione ha esito positivo, verrà chiamato l'evento **OnKrakenFuturesUnSubscribed**:

Se si verifica un errore durante la sottoscrizione o la cancellazione della sottoscrizione, verrà chiamato l'evento **On-KrakenFuturesError**.

Gli aggiornamenti del book verranno notificati nell'evento **OnKrakenData**.

```

{
"feed": "book_snapshot",
"product_id": "PI_XBTUSD",
"timestamp": 1612269825817,
"seq": 326072249,
"tickSize": null,
"bids": [{

```

```

"price": 34892.5,
"qty": 6385
},
{
"price": 34892,
"qty": 10924
}
],
"asks": [{
"price": 34911.5,
"qty": 20598
},
{
"price": 34912,
"qty": 2300
}
]
}

```

Ticker Lite

Il feed ticker lite restituisce informazioni ticker sui prodotti elencati.
Sottoscrivere il feed Spread chiamando il metodo **SubscribeTickerLite**.

Se la sottoscrizione ha esito positivo, verrà chiamato l'evento **OnKrakenFuturesSubscribed**:

Annullare la sottoscrizione chiamando il metodo **UnSubscribeTickerLite**:

Se l'annullamento dell'iscrizione ha esito positivo, verrà chiamato l'evento **OnKrakenFuturesUnSubscribed**:

Se si verifica un errore durante la sottoscrizione o la cancellazione della sottoscrizione, verrà chiamato l'evento **OnKrakenFuturesError**.

Gli aggiornamenti dello spread verranno notificati nell'evento **OnKrakenData**.

```
{ "feed": "ticker_lite",
```



```

"product_id": "PI_XBTUSD",
"bid": 34932,
"ask": 34949.5,
"change": 3.3705205220015966,
"premium": 0.1,
"volume": 264126741,
"tag": "perpetual",
"pair": "XBT:USD",
"dtm": 0,
"maturityTime": 0
}
{
"feed": "ticker_lite",
"product_id": "FI_ETHUSD_210625",
"bid": 1753.45,
"ask": 1760.35,
"change": 13.448175559936647,
"premium": 9.1,
"volume": 6899673.0,
"tag": "semiannual",
"pair": "ETH:USD",
"dtm": 141,
"maturityTime": 1624633200000
}

```

HeartBeat

Il feed heartbeat pubblica un messaggio heartbeat a intervalli regolari.

Eventi

OnConnect: quando il client WebSocket è connesso al server.

OnKrakenFuturesConnect: chiamato dopo una connessione websocket riuscita e quando il server invia lo stato del sistema.

OnKrakenFuturesSubscribed: chiamato dopo una sottoscrizione riuscita a un canale.

OnKrakenFuturesUnSubscribed: chiamato dopo un'annullamento riuscito della sottoscrizione a un canale.

OnKrakenFuturesError: chiamato se si verifica un errore durante la sottoscrizione/annullamento della sottoscrizione.

OnKrakenData: chiamato ogni volta che un'iscrizione a un canale ha un aggiornamento.

API Kraken Futures | API privata WebSocket

Connessione

URL: `wss://futures.kraken.com/ws/v1`

Autenticazione

Le richieste di sottoscrizione e cancellazione della sottoscrizione ai feed privati WebSocket richiedono un messaggio di challenge firmato con l'`api_secret` dell'utente.

La sfida viene ottenuta come mostrato nella sezione WebSocket API Public (utilizzando `api_key`).

Le richieste autenticate devono includere sia il messaggio di sfida originale (`original_challenge`) che quello firmato (`signed_challenge`) in formato JSON.

Per ottenere una WebSockets Challenge, è necessario impostare una chiave API e un segreto API nel componente Kraken Options; la chiave API è fornita da Kraken nel proprio account.

```
Kraken.ApiKey = "api key";
Kraken.ApiSecret = "api secret";
```

Metodi

Open Orders Verbose

Questo feed di sottoscrizione pubblica informazioni sugli ordini aperti dell'utente. Questo feed aggiunge informazioni extra su tutti gli ordini post-only che non sono riusciti ad attraversare il book.

```
SubscribeOpenOrdersVerbose();
```

In seguito, potrà annullare la sottoscrizione a `OpenOrdersVerbose` chiamando il metodo **UnSubscribeOpenOrdersVerbose**

```
UnSubscribeOpenOrdersVerbose();
```

Esempio di risposta dal server

```
{
  'feed': 'open_orders_verbose_snapshot',
  'instrument': 'PI_XBTUSD',
  'account': '0f9c23b8-63e2-40e4-9592-6d5aa57c12',
  'time': 1567428848005,
  'last_update': ...
}
```

Posizioni Aperte

Questo feed di sottoscrizione pubblica le posizioni aperte dell'account utente.

```
SubscribeOpenPositions();
```

In seguito, è possibile annullare la sottoscrizione a `OpenPositions` chiamando il metodo **UnSubscribeOpenPositions**

```
UnSubscribeOpenPositions();
```

Esempio di risposta dal server

```
{
  "feed": "open_positions",
  "account": "DemoUser",
  "positions": [{
    "instrument": "fi_xbtusd_180316",
    "balance": 2000.0,
    "entry_price": 11675.86541981,
    "mark_price": 11090.0,
    "index_price": 12290.5500000000001,
    "pnl": -0.00905299
  }]
}
```

Log account

Questo feed di sottoscrizione pubblica le informazioni sull'account.

```
SubscribeAccountLog();
```

In seguito, è possibile annullare la sottoscrizione da AccountLog chiamando il metodo **UnSubscribeAccountLog**

```
UnSubscribeAccountLog();
```

Esempio di risposta dal server

```
{
  'feed': 'account_log_snapshot',
  'logs': [{
    'id': 1690,
    'date': '2019-07-11T08:00:00.000Z',
    'asset': 'bch',
    'info': 'funding
rate change ',
    'booking_uid ':
86 fdc252 - 1 b6e - 40 ec - ac1d - c7bd46ddeebf ',
    'margin_account ':
f - bch: usd ',
    'old_balance ':0.01215667051,
    'new_balance ':0.01215736653,
    'old_average_entry_price ':0.0,
    'new_average_entry_price ':0.0,
    'trade_price ':0.0,
    'mark_price ':0.0,
    'realized_pnl ':0.0,
    'fee ':0.0,
    'execution ':
',
    'collateral ':
bch ',
    'funding_rate ':-8.7002552653e-08,
    'realized_funding ':6.9602e-07}]
}
```

Fills

Questo feed di sottoscrizione pubblica informazioni sui fills.

```
SubscribeFills();
```

In seguito è possibile annullare la sottoscrizione ai Fill chiamando il metodo **UnSubscribeFills**

```
UnSubscribeFills();
```

Esempio di risposta dal server

```
{
  "feed": "fills_snapshot",
  "account": "DemoUser",
  "fills": [
    {
      "instrument": "FI_XBTUSD_200925",
      "time": 1600256910739,
      "price": 10937.5,
      "seq": 36,
      "buy": true,
      "qty": 5000.0,
      "order_id": "9e30258b-5a98-4002-968a-5b0e149bcfbf",
      "fill_id": "cad76f07-814e-4dc6-8478-7867407b6bff",
      "fill_type": "maker",
      "fee_paid": -0.00009142857,
      "fee_currency": "BTC"
    }
  ]
}
```

Ordini aperti

Questo feed di sottoscrizione pubblica informazioni sugli ordini aperti dell'utente.

```
SubscribeOpenOrders();
```

Successivamente, è possibile annullare la sottoscrizione da OpenOrders chiamando il metodo **UnSubscribeOpenOrders**

```
UnSubscribeOpenOrders();
```

Esempio di risposta dal server

```
{
  "feed": "open_orders_snapshot",
```

```

"account": "e258dba9-4dd4-4da5-bfef-75beb91c098e",
"orders": [
{
"instrument": "PI_XBTUSD",
"time": 1612275024153,
"last_update_time": 1612275024153,
"qty": 1000,
"filled": 0,
"limit_price": 34900,
"stop_price": 13789,
"type": "stop",
"order_id": "723ba95f-13b7-418b-8fcf-ab7ba6620555",
"direction": 1,
"reduce_only": false,
"triggerSignal": "last"
}
]
}

```

Saldo Account e Margini

Questo feed di sottoscrizione restituisce le informazioni sul saldo e sul margine per l'account del client.

```
SubscribeAccountBalanceAndMargins();
```

Successivamente, è possibile annullare la sottoscrizione da AccountBalance chiamando il metodo **UnSubscribeAccountBalanceAndMargins**

```
UnSubscribeAccountBalanceAndMargins();
```

Esempio di risposta dal server

```

{
"feed": "account_balances_and_margins",
"account": "DemoUser",
"margin_accounts": [
{

```

```

"name": "xbt",
"balance": 0,
"pnl": 0,
"funding": 0,
"pv": 0,
"am": 0,
"im": 0,
"mm": 0
},
{
"name": "f-xbt:usd",
"balance": 9.99730211055,
"pnl": -0.00006034858674327812,
"funding": 0,
"pv": 9.997241761963258,
"am": 9.99666885201038,
"im": 0.0005729099528781564,
"mm": 0.0002864549764390782
},
],
"seq": 14
}

```

Notifiche

Questo feed di sottoscrizione pubblica notifiche al client.

```
SubscribeNotifications();
```

In seguito, è possibile annullare l'iscrizione alle notifiche chiamando il metodo **UnSubscribeNotifications**

```
UnSubscribeNotifications();
```

Esempio di risposta dal server

```

{

"feed": "notifications_auth",

```

```
"notifications":[  
  {  
    "id":5,  
    "type":"market",  
    "priority":"low",  
    "note":"Una nota che descrive la notifica.",  
    "effective_time":1520288300000  
  },  
  ...  
]  
}
```


API Kraken Futures | REST Public API

Connessione

URL: <https://futures.kraken.com/derivatives/api/v3>

L'API pubblica di Kraken Futures non richiede alcuna autenticazione.

Configurazione

L'unica configurazione riguarda se abilitare un log per le richieste HTTP REST. Abilitare HTTPLogOptions se si desidera salvare in un file di testo log tutte le richieste/risposte HTTP

Eventi

OnKrakenHTTPException: questo evento viene chiamato se si verifica un'eccezione durante una richiesta HTTP dall'API REST.

Metodi

GetFeeSchedules

Questo endpoint elenca tutti i piani tariffari. Non è richiesta l'autenticazione.

```
KrakenFutures.REST_API.GetFeeSchedules();
```

Libro degli ordini

Questo endpoint restituisce l'intero order book non cumulativo dei contratti Futures attualmente quotati.

```
KrakenFutures.REST_API.GetFeeSchedules("PI_XBTUSD");
```

Ticker

Questo endpoint restituisce i dati di mercato correnti per tutti i contratti Futures e gli indici attualmente quotati.

```
KrakenFutures.REST_API.GetTickers();
```

Strumenti

Questo endpoint restituisce le specifiche per tutti i contratti Futures e gli indici attualmente quotati.

```
KrakenFutures.REST_API.GetInstruments();
```

Cronologia

Questo endpoint restituisce gli ultimi 100 scambi a partire dal valore lastTime specificato - se non viene specificato alcun valore, restituisce gli ultimi 100 scambi. Questo endpoint restituisce la cronologia degli scambi per un massimo di 7 giorni dal momento in cui viene chiamato o dall'ultima versione del motore di trading (a seconda di quale sia più recente).

```
KrakenFutures.REST_API.GetHistory("PI_XBTUSD");
```

API Kraken Futures | REST Private API

Connessione

URL: <https://futures.kraken.com/derivatives/api/v3>

Autenticazione

La REST Private API richiede una API Key e un API Secret, questi valori sono forniti da Kraken nel suo account.

```
Kraken.ApiKey = "api key";  
Kraken.ApiSecret = "api secret";
```

Metodi

EditOrderByOrderId

Questo endpoint consente di modificare un ordine esistente per un contratto Futures attualmente quotato.

aOrderId: ID dell'ordine che si desidera modificare

aSize: La dimensione associata all'ordine

aLimitPrice: Il prezzo limite associato all'ordine.

aStopPrice: Il prezzo stop associato a un ordine stop. Obbligatorio se il vecchio tipo di ordine è Stop.

```
KrakenFutures.REST_API.EditOrderByOrderId("Order_Id", 2, 1000);
```

EditOrderByCliOrderId

Questo endpoint consente di modificare un ordine esistente per un contratto Futures attualmente quotato.

aCliOrderId: L'identità dell'ordine specificata dall'utente. Deve essere globalmente univoca.

aSize: La dimensione associata all'ordine

aLimitPrice: Il prezzo limite associato all'ordine.

aStopPrice: Il prezzo di stop associato a un ordine stop. Obbligatorio se il tipo di ordine è Stop.

```
KrakenFutures.REST_API.EditOrderByCliOrderId("Cli_Order_Id", 2, 1000);
```

SendMarketOrder

Questo endpoint consente di inviare un ordine di mercato.

aSide: La direzione dell'ordine: acquisto o vendita.

aSymbol: Il simbolo dei futures

aSize: La dimensione associata all'ordine.

```
KrakenFutures.REST_API.SendMarketOrder(kosfBuy, "PI_XBTUSD", 1);
```

SendLimitOrder

Questo endpoint consente di inviare un Limit Order.

aSide: La direzione dell'ordine: acquisto o vendita.

aSymbol: Il simbolo dei futures

aSize: La dimensione associata all'ordine.

aLimitPrice: Il prezzo limite associato all'ordine.

```
KrakenFutures.REST_API.SendLimitOrder(kosfBuy, "PI_XBTUSD", 1, 1000);
```

SendStopOrder

Questo endpoint consente di inviare un ordine Stop.

aSide: La direzione dell'ordine: acquisto o vendita.

aSymbol: Il simbolo dei futures

aSize: La dimensione associata all'ordine.

aStopPrice: Il prezzo stop associato a un ordine stop.

aLimitPrice: Il prezzo limite associato all'ordine.

```
KrakenFutures.REST_API.SendStopOrder(kosfBuy, "PI_XBTUSD", 1, 1000, 900);
```

SendTakeProfitOrder

Questo endpoint consente di inviare un ordine Take Profit.

aSide: La direzione dell'ordine: acquisto o vendita.

aSymbol: Il simbolo dei futures

aSize: La dimensione associata all'ordine.

aStopPrice: Il prezzo stop associato a un ordine stop.

aLimitPrice: Il prezzo limite associato all'ordine.

```
KrakenFutures.REST_API.SendTakeProfitOrder(kosfBuy, "PI_XBTUSD", 1, 1000, 900);
```

SendOrder

Questo endpoint consente di inviare un ordine limit, stop, take profit o immediate-or-cancel per un contratto Futures attualmente quotato.

OrderType: selezioni uno dei seguenti: kotfLMT, kotfPOST, kotfMKT, kotfSTP, kotfTAKE_PROFIT, kotfIOC

Symbol: Il simbolo dei futures

Side: La direzione dell'ordine (acquisto o vendita).

Dimensione: La dimensione associata all'ordine.

StopPrice: Il prezzo stop associato a un ordine stop.

LimitPrice: Il prezzo limite associato all'ordine.

TriggerSignal: Se si inserisce un ordine Stop o TakeProfit, il segnale utilizzato per il trigger; selezionare uno tra kotsMark, kotsIndex, kotsLast

CliOrderId: L'identità dell'ordine specificata dall'utente. Deve essere globalmente univoca.

ReduceOnly: Impostare su true se si desidera che l'ordine riduca solo una posizione esistente. Qualsiasi ordine che aumenta una posizione esistente verrà rifiutato. Il valore predefinito è false.

```
TsgcHTTPKrakenFuturesOrder oOrder = new TsgcHTTPKrakenFuturesOrder();
oOrder.Side = TsgcHTTPKrakenFuturesOrderSide.kosfBuy;
oOrder.Symbol = "PI_XBTUSD";
oOrder.OrderType = TsgcHTTPKrakenFuturesOrderType.kotfMKT;
oOrder.Size = 1;
KrakenFutures.REST_API.SendOrder(oOrder);
```

CancelOrderByOrderId

Questo endpoint consente di annullare un ordine aperto per un contratto Futures.

aOrderId: ID dell'ordine che si desidera modificare

```
KrakenFutures.REST_API.CancelOrderByOrderId("Order_Id");
```

CancelOrderByCliOrderId

Questo endpoint consente di annullare un ordine aperto per un contratto Futures.

aCliOrderId: L'identità dell'ordine specificata dall'utente. Deve essere globalmente univoca.

```
KrakenFutures.REST_API.CancelOrderByCliOrderId("Cli_Order_Id");
```

GetFills

Questo endpoint restituisce informazioni sugli ordini evasi per tutti i contratti futures.

aLastFillDate: Se non fornito, restituisce gli ultimi 100 fill in qualsiasi contratto futures. Se fornito, restituisce i 100 movimenti precedenti a lastFillTime.

```
KrakenFutures.REST_API.GetFills("2020-07-22T13:45:00.000Z");
```

Transfer

Questo endpoint consente di trasferire fondi tra due account margin con la stessa valuta di garanzia, oppure tra un account margin e il proprio account cash.

aFromAccount: Il nome dell'account di cassa o margine da cui trasferire i fondi.

aToAccount: Il nome del conto di liquidità o del conto a margine verso cui spostare i fondi.

aUnit: L'unità da trasferire.

aAmount: L'importo da trasferire.

```
KrakenFutures.REST_API.Transfer("FI_XBTUSD", "cash", "xbt", 1.5);
```

GetOpenPositions

Questo endpoint restituisce la dimensione e il prezzo medio di ingresso di tutte le posizioni aperte nei contratti Futures. Sono inclusi i contratti Futures scaduti ma non ancora regolati.

```
KrakenFutures.REST_API.GetOpenPositions();
```

GetNotifications

Questo endpoint fornisce le notifiche della piattaforma.

```
KrakenFutures.REST_API.GetNotifications();
```

GetAccounts

Questo endpoint restituisce informazioni chiave relative a tutti i conti Kraken Futures, che possono essere conti in contanti o conti a margine. Ciò include i saldi in asset digitali, i saldi degli strumenti, i requisiti di margine, le stime di attivazione del margine e informazioni ausiliarie come i fondi disponibili, il PnL delle posizioni aperte e il valore del portafoglio.

```
KrakenFutures.REST_API.GetAccounts();
```

CancelAllOrders

Questo endpoint consente di annullare un ordine aperto per un contratto Futures.

Symbol: Un prodotto futures per cui annullare tutti gli ordini aperti (opzionale)

```
KrakenFutures.REST_API.CancelAllOrders();
```

CancelAllOrdersAfter

Questo endpoint fornisce un meccanismo Dead Man's Switch per proteggere il client da malfunzionamenti della rete. Il client può inviare una richiesta con un timeout in secondi che farà partire un timer di conto alla rovescia che annullerà tutti gli ordini del client alla scadenza del timeout.

aTimeout: Il timeout specificato in secondi.

```
KrakenFutures.REST_API.CancelAllOrdersAfter(60);
```

GetOpenOrders

Questo endpoint restituisce informazioni su tutti gli ordini aperti per tutti i contratti Futures.

```
KrakenFutures.REST_API.OpenOrders();
```

GetHistoricalOrders

Questo endpoint restituisce gli ordini storici effettuati su un account.

aSince: Il DateTime Since

aBefore: Il DateTime precedente

aSort: "asc" per ordine crescente, "desc" per ordine decrescente

aContinuationToken: Token di continuazione fornito da una risposta precedente che può essere utilizzato nella chiamata per restituire il successivo insieme di risultati disponibili

```
KrakenFutures.REST_API.GetHistoricalOrders(1604937694000, 1604937700000);
```

GetHistoricalTriggers

Questo endpoint restituisce i trigger storici effettuati su un account.

aSince: Il DateTime Since

aBefore: Il DateTime precedente

aSort: "asc" per ordine crescente, "desc" per ordine decrescente

aContinuationToken: Token di continuazione fornito da una risposta precedente che può essere utilizzato nella chiamata per restituire il successivo insieme di risultati disponibili

```
KrakenFutures.REST_API.GetHistoricalTriggers(1604937694000, 1604937700000);
```

GetHistoricalExecutions

Questo endpoint restituisce le esecuzioni storiche effettuate su un account.

aSince: Il DateTime Since

aBefore: Il DateTime precedente

aSort: "asc" per ordine crescente, "desc" per ordine decrescente

aContinuationToken: Token di continuazione fornito da una risposta precedente che può essere utilizzato nella chiamata per restituire il successivo insieme di risultati disponibili

```
KrakenFutures.REST_API.GetHistoricalExecutions(1604937694000, 1604937700000);
```

WithdrawalToSpotWallet

Questo endpoint consente di inviare una richiesta di prelievo di asset digitali da un wallet Kraken Futures al proprio wallet Kraken Spot.

aCurrency: La risorsa digitale da prelevare, ad es. xbt o xrp.

aAmount: L'importo della valuta da prelevare.

```
KrakenFutures.REST_API.WithdrawalToSpotWallet("xbt", 1000);
```

GetFeeScheduleVolumes

Questo endpoint restituisce il volume USD degli ultimi 30 giorni.

```
KrakenFutures.REST_API.GetFeeScheduleVolumes();
```

GetAccountLogCSV

Questo endpoint consente ai client di scaricare un file csv dei log del proprio account.

```
KrakenFutures.REST_API.GetAccountLogCSV();
```

API Pusher

Pusher

Pusher è una piattaforma facile e affidabile con funzionalità interessanti basate sul protocollo WebSocket: messaggistica pub/sub flessibile, liste utenti in tempo reale (presenza), autenticazione...

La versione dell'API WebSocket Pusher è 7.

I dati vengono inviati in modo bidirezionale tramite WebSocket come dati di testo contenenti JSON codificato in UTF8 (i frame WebSocket binari non sono supportati).

È possibile chiamare il metodo **Ping** per testare la connessione al server. In sostanza, qualsiasi messaggio ricevuto dall'altra parte è considerato un segnale che la connessione è attiva. In assenza di messaggi, entrambe le parti possono verificare che l'altra stia rispondendo inviando un messaggio ping, al quale l'altra parte dovrebbe rispondere con un pong.

Prima di connettersi, è necessario completare i seguenti campi:

Importante

Pusher richiede che il client websocket si connetta a un URL usando i campi precedenti (key, cluster...), questi campi sono usati per costruire l'url e ciò avviene quando si assegna il client nel componente pusher. Quindi, per essere sicuri che l'URL sia costruito correttamente, imposti il client dopo aver compilato i campi di configurazione di pusher. Trovi di seguito lo pseudo-codice:

```
// configura i campi pusher
pusher.cluster = ...
pusher.key = ...
// imposta client
pusher.client = websocket client
// start connection
websocket client.Active = true;
```

Dopo una connessione riuscita, viene generato l'evento **OnPusherConnect** e si ottengono i seguenti campi:

- Socket ID: Un identificatore univoco per il client connesso.
- Timeout: Il numero di secondi di inattività del server dopo i quali il client deve avviare un messaggio ping (questo è gestito automaticamente dal componente).

In caso di errore, verrà generato **OnPusherError** con le informazioni sull'errore. Un errore può essere inviato da Pusher in risposta a un'autenticazione non valida, un comando non valido, ecc.

4000-4099

Indica un errore che ha causato la chiusura della connessione da parte di Pusher e che un tentativo di riconnessione con gli stessi parametri non avrà successo.

- 4000: L'applicazione accetta solo connessioni SSL, riconnettersi utilizzando wss://
- 4001: L'applicazione non esiste
- 4003: Applicazione disabilitata
- 4004: L'applicazione ha superato la quota di connessioni
- 4005: Percorso non trovato
- 4006: Formato della stringa di versione non valido
- 4007: Versione del protocollo non supportata
- 4008: Nessuna versione di protocollo fornita

4100-4199

Indica un errore che ha comportato la chiusura della connessione da parte di Pusher e che il client può riconnettersi dopo 1 secondo o più.

- 4100: Capacità esaurita

4200-4299

Indica un errore che ha causato la chiusura della connessione da parte di Pusher, e che il client può riconnettersi immediatamente.

- 4200: Riconnessione generica immediata
- 4201: Risposta Pong non ricevuta: è stato inviato un ping al client, ma non è stata ricevuta alcuna risposta - vedere i messaggi ping e pong
- 4202: Chiuso per inattività: Il client è rimasto inattivo per un lungo periodo (attualmente 24 ore) e non supporta il ping. Aggiornare a una versione più recente di WebSocket o implementare la versione 5 o superiore di questo protocollo.

4300-4399

Qualsiasi altro tipo di errore.

- 4301: Evento client rifiutato per limite di velocità

Canali

I canali sono un concetto fondamentale in Pusher. Ogni applicazione ha un certo numero di canali, e ogni client può scegliere a quali canali iscriversi.

I canali forniscono:

- Un modo per filtrare i dati. Ad esempio, in un'applicazione di chat, potrebbe esserci un canale per le persone che desiderano discutere di 'cani'.
- Un modo per controllare l'accesso a diversi flussi di informazioni. Ad esempio, un'applicazione di gestione dei progetti vorrà autorizzare gli utenti a ricevere aggiornamenti su "projectX".

È fortemente raccomandato utilizzare i canali per filtrare i dati e non farlo tramite gli eventi. Questo perché tutti gli eventi pubblicati su un canale vengono inviati a tutti i sottoscrittori, indipendentemente dal loro binding agli eventi.

I canali non devono essere creati esplicitamente e vengono istanziati su richiesta del client. Ciò significa che creare un canale è semplice. È sufficiente indicare a un client di iscriversi ad esso.

Sono supportati i seguenti tipi di canali:

- **Canali pubblici** possono essere sottoscritti da chiunque ne conosca il nome.
- I **canali privati** introducono un meccanismo che consente al Suo server di controllare l'accesso ai dati che sta trasmettendo.
- I **canali presence** sono un'estensione dei canali privati. Consentono di registrare le informazioni dell'utente al momento dell'iscrizione e permettono agli altri membri del canale di sapere chi è online.
- I **Cache channels** ricordano l'ultimo evento attivato e lo inviano come primo evento ai nuovi sottoscrittori (varianti public, private e presence).
- I **canali privati cifrati** forniscono la crittografia end-to-end tramite NaCl secretbox, garantendo che nemmeno Pusher possa leggere i dati del messaggio.

Canali pubblici

I canali pubblici devono essere utilizzati per dati accessibili pubblicamente in quanto non richiedono alcuna forma di autorizzazione per essere sottoscritti.

È possibile sottoscrivere e annullare la sottoscrizione ai canali in qualsiasi momento. Non è necessario attendere il completamento della connessione di Pusher.

Esempio: sottoscrizione al canale "my-channel".

Se la sottoscrizione va a buon fine, verrà generato l'evento **OnPusherSubscribe**; in caso di errore si riceverà un messaggio nell'evento **OnPusherError**.

Tutti i messaggi del canale sottoscritto verranno ricevuti nell'evento **OnPusherEvent**.

Quando viene chiamato il metodo Publish e il canale è Public, il componente, anziché utilizzare il protocollo WebSocket, utilizza il protocollo HTTP e chiama il metodo TriggerEvent (la pubblicazione non è consentita utilizzando il protocollo websocket).

Canali privati

Richiede Indy 10.5.7 o versione successiva

I canali privati devono essere utilizzati quando l'accesso al canale deve essere limitato in qualche modo. Affinché un utente possa iscriversi a un canale privato, l'autorizzazione deve essere verificata.

Esempio: iscrizione al canale "my-private-channel".

Se la sottoscrizione va a buon fine, verrà generato l'evento **OnPusherSubscribe**; in caso di errore si riceverà un messaggio nell'evento **OnPusherError**.

Tutti i messaggi del canale sottoscritto verranno ricevuti nell'evento **OnPusherEvent**.

Canali Presence

Richiede Indy 10.5.7 o versione successiva

I canali Presence si basano sulla sicurezza dei canali Private e aggiungono la funzionalità di conoscere chi è iscritto a quel canale. Questo rende estremamente semplice implementare funzionalità di chat room e "chi è online" nella propria applicazione. Si pensi a chat room, collaboratori su un documento, persone che visualizzano la stessa pagina web, concorrenti in un gioco e situazioni simili.

I canali Presence vengono sottoscritti dall'API client nello stesso modo dei canali privati, ma il nome del canale deve essere preceduto dal prefisso presence-. Come per i canali privati, viene effettuata una richiesta HTTP a un URL di autenticazione configurabile per determinare se l'utente corrente dispone delle autorizzazioni per accedere al canale.

Le informazioni sugli utenti che si iscrivono e si discrivono da un canale possono essere accessibili associandosi agli eventi sul canale presence, e lo stato corrente degli utenti iscritti al canale è disponibile tramite la proprietà channel.members.

Esempio: sottoscrivi il canale "my-presence-channel".

Se la sottoscrizione va a buon fine, verrà generato l'evento **OnPusherSubscribe**; in caso di errore si riceverà un messaggio nell'evento **OnPusherError**.

Tutti i messaggi del canale sottoscritto verranno ricevuti nell'evento **OnPusherEvent**.

Canali Cache

Un canale cache ricorda l'ultimo evento generato e lo invia come primo evento ai nuovi sottoscrittori.

Quando un evento viene attivato su un canale cache, Pusher Channels memorizza questo evento nella cache e, quando un client si sottoscrive a un canale cache, se esiste un valore memorizzato nella cache, questo viene inviato al client come primo evento su quel canale. Questo comportamento aiuta gli sviluppatori a fornire lo stato iniziale senza aggiungere logica aggiuntiva per recuperarlo da altrove.

Sono supportati i seguenti canali Cache:

- Canale Cache Pubblico
- Canale cache privato
- Canale Cache Presence

Esempio: sottoscrizione al canale cache pubblico "my-cache-channel".

Se la sottoscrizione va a buon fine, verrà generato l'evento **OnPusherSubscribe**; in caso di errore si riceverà un messaggio nell'evento **OnPusherError**.

Tutti i messaggi del canale sottoscritto verranno ricevuti nell'evento **OnPusherEvent**.

Se non è presente alcun evento memorizzato nella cache al momento della sottoscrizione a un canale cache, verrà generato l'evento **OnPusherCacheMiss**, fornendo il nome del canale. Ciò consente all'applicazione di gestire il caso in cui non siano disponibili dati memorizzati nella cache.

Canali privati cifrati

I canali Private-Encrypted forniscono la crittografia end-to-end per i messaggi. Come i canali privati, richiedono l'autenticazione, ma in aggiunta tutti i payload di dati vengono crittografati utilizzando NaCl secretbox, in modo che solo i subscriber autorizzati possano leggere il contenuto. Nemmeno Pusher stessa può decrittografare i messaggi. Per utilizzare i canali privati cifrati, è necessario fornire una **SharedSecret** durante l'autenticazione. Il segreto condiviso viene utilizzato per cifrare e decifrare i dati dei messaggi.

Esempio: sottoscrizione a un canale privato cifrato "my-encrypted-channel".

È disponibile anche una variante private-encrypted-cache, che combina la crittografia con il comportamento del canale cache:

Quando si utilizza l'evento **OnPusherAuthentication** con canali privati cifrati, è possibile impostare la proprietà **SharedSecret** sull'oggetto di risposta per fornire la chiave di cifratura:

Eventi di Presenza

I canali Presence forniscono eventi aggiuntivi che notificano all'applicazione quando gli utenti entrano o lasciano un canale, e consentono di monitorare il numero di sottoscrizioni.

OnPusherMemberAdded

Generato quando un nuovo membro si sottoscrive a un canale di presenza. Fornisce il nome del canale, l'ID utente e le informazioni sull'utente del membro che si è unito.

OnPusherMemberRemoved

Viene generato quando un membro si disiscrive da un canale presence. Fornisce il nome del canale, l'ID utente e le informazioni utente del membro che ha abbandonato.

OnPusherSubscriptionCount

Generato quando il conteggio delle sottoscrizioni cambia su un canale. Fornisce il nome del canale e il numero corrente di sottoscrittori. Questo evento deve essere abilitato nella Sua dashboard Pusher.

OnPusherCacheMiss

Attivato quando ci si sottoscrive a un canale cache che non ha eventi memorizzati nella cache. Fornisce il nome del canale. Questo consente all'applicazione di gestire il caso in cui non siano disponibili dati memorizzati nella cache, ad esempio recuperando i dati da un'altra fonte.

Pubblicazione di messaggi

Non solo è possibile ricevere messaggi dai canali sottoscritti, ma è anche possibile inviare messaggi ad altri utenti sottoscritti.

Chiamare il metodo **Publish** per inviare un messaggio a tutti gli utenti sottoscritti al canale.

Esempio: inviare un evento a tutti gli utenti iscritti a "my-channel"

Pubblichi non più di 10 messaggi al secondo per client (connessione). Qualsiasi evento generato al di sopra di questo limite di rate verrà rifiutato dall'API Pusher. Questo non è un problema del sistema, è un problema del client. 100 client in un canale che inviano messaggi a questa velocità dovrebbero ciascuno anche elaborare 1.000 messaggi al secondo! Sebbene alcuni browser moderni potrebbero essere in grado di gestire questo, è molto probabilmente una cattiva idea.

REST API

L'API è ospitata su <http://api-CLUSTER.pusher.com>, dove CLUSTER viene sostituito con il cluster della propria app (ad esempio, eu).

I codici di stato HTTP vengono utilizzati per indicare il successo o meno delle richieste. Di seguito sono riportati i codici di stato più comuni:

200 Richiesta riuscita. Il body conterrà un hash JSON di dati di risposta

400 Errore: dettagli nel corpo della risposta

401 Errore di autenticazione: il corpo della risposta conterrà una spiegazione

403 Forbidden: app disabilitata o quota di messaggi superata

Sono state implementate le seguenti funzioni delle API REST.

Funzione	Description
TriggerEvent	Attiva un nuovo evento sul canale specificato. Supporta i parametri opzionali SocketId (per escludere un client) e Info.
TriggerBatchEvents	Genera più eventi in una singola richiesta HTTP. Accetta un array JSON di oggetti evento.
GetChannels	Fornisce un elenco di tutti i canali attivi. Supporta i parametri opzionali FilterByPrefix e Info.

GetChannel	Fornisce informazioni su un canale specifico. Supporta un parametro Info opzionale.
GetUsers	Fornisce un elenco di tutti gli utenti connessi a un canale.
TerminateUserConnections	Termina tutte le connessioni di un dato utente tramite il suo ID utente.

TriggerEvent

Attiva un evento su uno o più canali. Richiede il nome dell'evento, il nome del canale e il payload di dati.

Parametro	Description
aEventName	Il nome dell'evento da attivare.
aChannel	Il nome del canale su cui attivare l'evento.
aData	I dati dell'evento (stringa JSON).
aSocketId (facoltativo)	Un ID socket da escludere dalla ricezione dell'evento. Utile per evitare che il mittente riceva il proprio messaggio.
aInfo (facoltativo)	Un elenco separato da virgole di attributi da includere nella risposta (ad es. "subscription_count").

TriggerBatchEvents

Attiva più eventi in una singola chiamata API, il che è più efficiente rispetto all'invio di richieste separate per ogni evento. Il parametro batch deve essere una stringa JSON contenente un array di oggetti evento, dove ogni oggetto ha i campi "channel", "name" e "data".

GetChannels

Restituisce un elenco di canali attivi. Supporta parametri opzionali per filtrare i risultati e richiedere informazioni aggiuntive.

Parametro	Description
aFilterByPrefix (opzionale)	Filtra i canali in base a un prefisso del nome (ad esempio "presence-" per elencare solo i canali presence).
aInfo (facoltativo)	Un elenco separato da virgole di attributi da includere nella risposta (ad es. "user_count").

GetChannel

Restituisce informazioni su un canale specifico.

Parametro	Description

aChannel	Il nome del canale di cui si vogliono ottenere informazioni.
alInfo (facoltativo)	Un elenco di attributi separati da virgola da includere (ad es. "user_count,subscription_count").

GetUsers

Restituisce un elenco di utenti connessi a un canale presence. Il nome del canale deve includere il prefisso completo (es. "presence-my-channel").

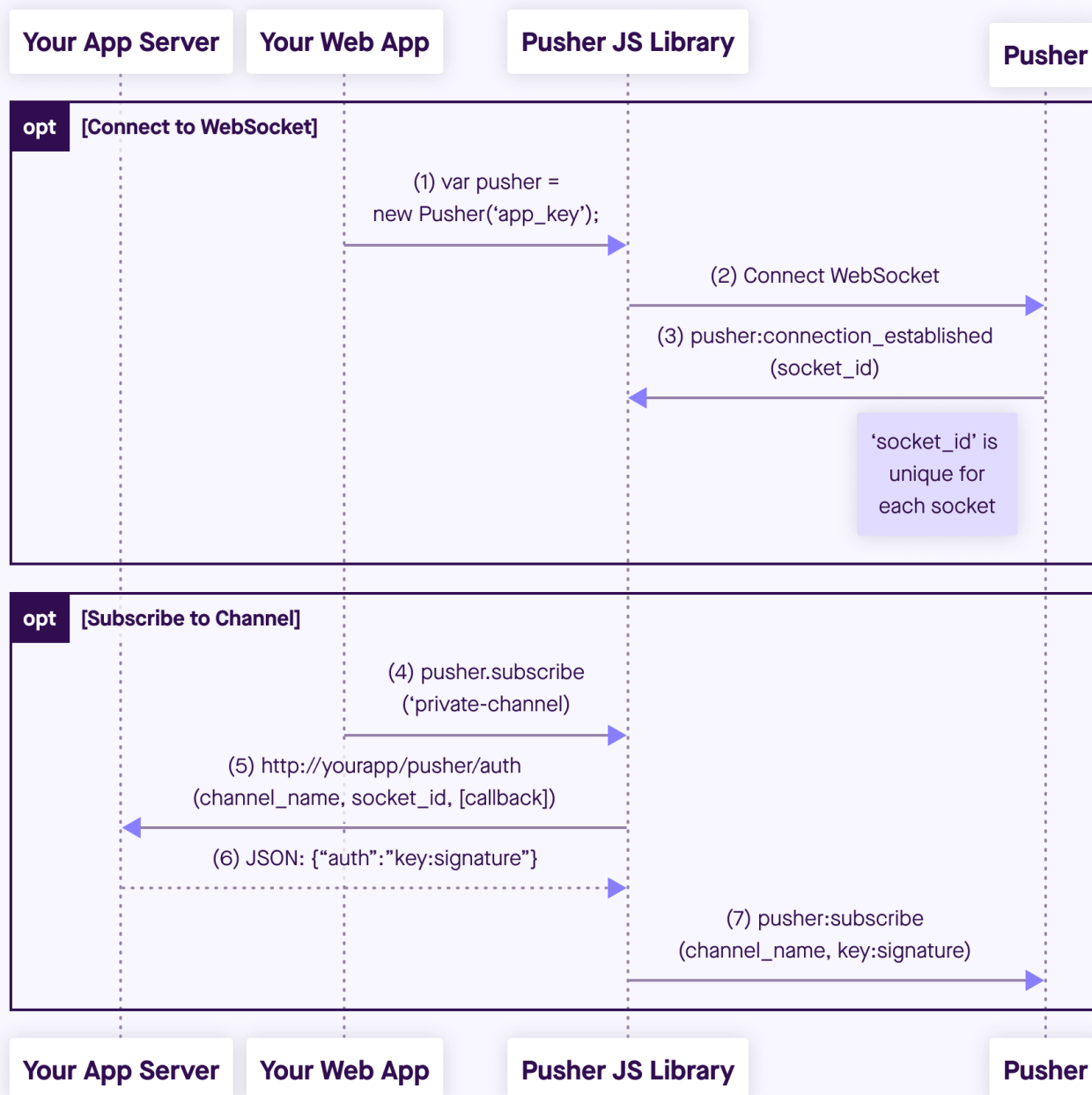
TerminateUserConnections

Termina tutte le connessioni stabilite da un determinato utente. Può essere utilizzato per forzare la disconnessione di un utente specifico da tutti i canali. L'ID utente deve corrispondere allo "user_id" utilizzato quando l'utente si è iscritto a un canale presence.

Autenticazione personalizzata

Pusher consente la sottoscrizione solo a canali privati o presence; se la connessione fornisce un token di autenticazione, ciò consente di limitare l'accesso.

È possibile creare un proprio flusso di autenticazione, utilizzando l'evento **OnPusherAuthentication**; questo evento viene chiamato prima che il messaggio di sottoscrizione venga firmato con la chiave segreta fornita da Pusher. Questo evento ha 2 parametri: una richiesta di autenticazione con campi come SocketId, nome del canale... che possono essere utilizzati dal proprio server di autenticazione per autenticare o meno la richiesta. Di seguito è riportato uno screenshot che mostra il flusso di autenticazione Pusher



Quando un client si connette al server pusher, invia la chiave fornita da pusher e il server restituisce un id di identificazione (`socket_id`).

Quando un client si iscrive a un canale privato (o presence), il client `sgcWebSockets` utilizza la Secret Key fornita da pusher per creare una firma che viene inclusa nel messaggio di sottoscrizione. Utilizzando l'evento `OnPusherAuthentication`, è possibile acquisire i campi necessari per firmare il messaggio, implementare i propri metodi di autenticazione e, in caso di successo, restituire la firma che verrà inclusa nel messaggio di sottoscrizione e inviata al server.

Esempio:

Il formato della firma è:

Canali privati: `key:HMAC256(SocketID, ChannelName)`

Canali Presence: chiave: `HMAC256(SocketID, ChannelName, Data)`

L'oggetto **`TsgcWSPusherResponseAuthentication`** fornisce le seguenti proprietà:

Proprietà	Description
Segreto	La chiave segreta Pusher utilizzata per calcolare la firma HMAC. Precompilata con <code>Pusher.Secret</code> se configurata.

Signature	La firma di autenticazione calcolata. Se lasciata vuota, il componente la calcolerà automaticamente utilizzando il Secret.
SharedSecret	La chiave segreta condivisa per i canali private-encrypted. Obbligatoria quando si sottoscrive pscPrivateEncryptedChannel o pscPrivateEncryptedCacheChannel. Utilizzata per la crittografia end-to-end dei dati dei messaggi.

API Bitmex

Bitmex

Bitmex è una piattaforma di scambio di criptovalute e trading di derivati.

Sono supportate le seguenti API:

1. **Stream WebSocket:** consente di sottoscrivere ad alcuni metodi e ricevere dati in tempo reale. Gli eventi vengono inviati ai client dal server agli iscritti. Utilizza WebSocket come protocollo.
2. **REST API:** i client possono richiedere al server dati di mercato e dell'account. Richiede una API Key e una Secret per l'autenticazione e utilizza HTTPS come protocollo.

Proprietà

L'API di Bitmex ha 2 tipi di metodi: pubblici e privati. I metodi pubblici sono accessibili senza autenticazione, ad esempio: ottenere i prezzi dei ticker. Alcuni sono privati e relativi ai dati utente; tali metodi richiedono l'uso delle chiavi API di Bitmex.

- **ApiKey:** può richiedere una nuova chiave API nel suo account Bitmex; basta copiare il valore in questa proprietà.
- **ApiSecret:** è il segreto dell'API; conservarlo al sicuro.
- **TestNet:** se abilitato, si conatterà all'account demo di Bitmex (per impostazione predefinita false).
- **HTTPLogOptions:** archivia in un file di testo un log delle richieste HTTP
 - **Enabled:** se abilitato, archiverà tutte le richieste HTTP dell'API WebSocket.
 - **FileName:** percorso completo del file in cui verranno archiviati i log

Usi più comuni

- **WebSockets API**
 - [Come Connettersi alla WebSocket API](#)
 - [Come Sottoscrivere un Canale WebSocket](#)
- **REST API**
 - [Come Inserire un Ordine Bitmex](#)

WebSocket API

Sottoscrivi / Annulla sottoscrizione

BitMEX consente di sottoscrivere dati in tempo reale. Questo accesso non è soggetto a limitazione di frequenza una volta connessi ed è il modo migliore per ottenere i dati più aggiornati nei propri programmi. In alcuni argomenti, è possibile passare un Symbol per filtrare gli eventi per simbolo, ad esempio: trades, quotes...

I seguenti topic di sottoscrizione sono disponibili senza autenticazione:

- **btmAnnouncement:** Annunci del sito
- **btmChat:** Chat Trollbox
- **btmConnected:** Statistiche degli utenti/bot connessi
- **btmFunding:** Aggiornamenti dei tassi di finanziamento degli swap. Inviato a ogni intervallo di finanziamento (solitamente 8 ore)
- **btmInstrument:** Aggiornamenti dello strumento inclusi turnover e bid/ask
- **btmInsurance:** Aggiornamenti giornalieri del fondo assicurativo
- **btmLiquidation:** Ordini di liquidazione nel momento in cui vengono inseriti nel book

- **btmOrderBookL2_25**: Primi 25 livelli del book degli ordini di livello 2
- **btmOrderBookL2**: Libro degli ordini completo di livello 2
- **btmOrderBook10**: I primi 10 livelli con aggiornamento tradizionale del book completo
- **btmPublicNotifications**: Notifiche di sistema (utilizzate per messaggi di breve durata)
- **btmQuote**: Livello superiore del book
- **btmQuoteBin1m**: bin di quotazione da 1 minuto
- **btmQuoteBin5m**: Bin di quotazione di 5 minuti
- **btmQuoteBin1h**: bin di quotazione a 1 ora
- **btmQuoteBin1d**: Quote bin giornaliera
- **btmSettlement**: Liquidazioni
- **btmTrade**: Transazioni in tempo reale
- **btmTradeBin1m**: bin di trading da 1 minuto
- **btmTradeBin5m**: bin di trading da 5 minuti
- **btmTradeBin1h**: bin di trading a 1 ora
- **btmTradeBin1d**: bin di trading giornalieri

I seguenti argomenti richiedono l'autenticazione:

- **btmAffiliate**: Stato di affiliazione, ad esempio numero totale di utenti referenziati e percentuale di pagamento
- **btmExecution**: Esecuzioni singole; possono essere multiple per ordine
- **btmOrder**: Aggiornamenti in tempo reale sui propri ordini
- **btmMargin**: Aggiornamenti sul saldo corrente dell'account e sui requisiti di margine
- **btmPosition**: Aggiornamenti sulle proprie posizioni
- **btmPrivateNotifications**: Notifiche individuali - attualmente non utilizzate
- **btmTransact**: aggiornamenti di deposito/prelievo
- **btmWallet**: Dati del saldo dell'indirizzo Bitcoin, inclusi depositi & prelievi totali

Esempio di messaggi ricevuti:

```
{
  "table": "orderBookL2_25",
  "keys": ["symbol", "id", "side"],
  "types": {"id": "long", "price": "float", "side": "symbol", "size": "long", "symbol": "symbol"},
  "foreignKeys": {"side": "side", "symbol": "instrument"},
  "attributes": {"id": "sorted", "symbol": "grouped"},
  "action": "partial",
  "data": [
    {"symbol": "XBTUSD", "id": 17999992000, "side": "Sell", "size": 100, "price": 80},
    {"symbol": "XBTUSD", "id": 17999993000, "side": "Sell", "size": 20, "price": 70},
    {"symbol": "XBTUSD", "id": 17999994000, "side": "Sell", "size": 10, "price": 60},
    {"symbol": "XBTUSD", "id": 17999995000, "side": "Buy", "size": 10, "price": 50},
    {"symbol": "XBTUSD", "id": 17999996000, "side": "Buy", "size": 20, "price": 40},
    {"symbol": "XBTUSD", "id": 17999997000, "side": "Buy", "size": 100, "price": 30}
  ]
}

{
  "table": "orderBookL2_25",
  "action": "update",
  "data": [
    {"symbol": "XBTUSD", "id": 17999995000, "side": "Buy", "size": 5}
  ]
}

{
  "table": "orderBookL2_25",
  "action": "delete",
  "data": [
    {"symbol": "XBTUSD", "id": 17999995000, "side": "Buy"}
  ]
}

{
  "table": "orderBookL2_25",
  "action": "insert",
  "data": [
    {"symbol": "XBTUSD", "id": 17999995500, "side": "Buy", "size": 10, "price": 45},
  ]
}
```

Autenticazione

Per iscriversi a stream bloccati dall'utente è necessario autenticarsi prima. Si noti che un'autenticazione non valida chiuderà la connessione.

L'utilizzo dell'API BitMEX richiede una chiave API.

Le chiavi API permanenti possono essere bloccate su intervalli di indirizzi IP e revocate a piacere senza compromettere le credenziali principali. Non richiedono nemmeno il rinnovo.

Per utilizzare l'autenticazione tramite chiave API, è necessario generare una chiave API nel proprio account.

Chiamare il metodo **Authenticate** prima di iscriversi a qualsiasi Authenticated Topic.

CancelAllAfter (Dead Man's Switch)

Il metodo **CancelAllAfter** implementa la funzione Dead Man's Switch. Quando viene chiamato con un valore di timeout (in millisecondi), indica al server di annullare tutti gli ordini aperti se non viene ricevuta alcuna successiva chiamata CancelAllAfter entro il periodo di timeout. Questa funzione è utile per garantire che gli ordini vengano annullati in caso di disconnessione dalla rete.

REST API

Metodo	Descrizione
GetExecutions	Restituisce tutte le transazioni grezze, incluse l'apertura e la cancellazione degli ordini e le modifiche dello stato degli ordini.
GetExecutionsTrade-History	Restituisce transazioni più mirate.
GetInstruments	Restituisce tutti gli strumenti e gli indici, inclusi quelli che sono stati liquidati o non sono in lista. Utilizzi questo endpoint se si desidera interrogare singoli strumenti o applicare un filtro complesso.
GetOrders	Per ottenere solo gli ordini aperti
PlaceOrder	Inserisce un ordine grezzo utilizzando l'oggetto TsgcHTTPBitmexOrder.
PlaceMarketOrder	Piazza un nuovo ordine MARKET.
PlaceLimitOrder	Inserisce un nuovo ordine LIMIT.
PlaceStopOrder	Inserisce un nuovo ordine STOP.
PlaceStopLimitOrder	Inserisci un nuovo ordine STOPLIMIT.
AmendOrder	Modifica un ordine esistente.
CancelOrder	Annulla un Order attivo.
CancelAllOrders	Annulla tutti gli ordini attivi.
CancelAllOrdersAfter	Annulla tutti gli ordini dopo un certo tempo.
ClosePosition	Chiude una posizione aperta.
GetOrderBook	Ottieni il libro degli ordini corrente in formato verticale
GetPosition	Ottiene le Sue posizioni.
SetPositionIsolate	Abilitare il margine isolato o incrociato per posizione.
SetPositionLeverage	Scegliere la leva finanziaria per posizione.
SetPositionRiskLimit	Aggiornare il limite di rischio.
SetPositionTransfer-Margin	Trasferire equity in entrata o in uscita da una posizione.
GetQuotes	Ottieni quotazioni
GetTrades	Ottieni transazioni
GetFunding	Ottiene i dati di funding.

GetInsurance	Recupero dei dati del fondo assicurativo.
GetTradeBucketed	Recupera i dati di trading raggruppati (OHLCV) con dimensioni di bin configurabili.
GetQuoteBucketed	Recupera dati di quotazione raggruppati con dimensioni dei contenitori configurabili.
GetSettlement	Recupera i dati di liquidazione.
GetLiquidation	Recupera gli ordini di liquidazione.
GetInstrumentIndices	Ottenere gli indici degli strumenti.
GetInstrumentCompositeIndex	Ottiene i dati dell'indice composito per gli strumenti.
GetStats	Ottiene le statistiche complessive dell'exchange.
GetStatsHistory	Ottieni statistiche storiche sugli exchange.
GetStatsHistoryUSD	Ottiene le statistiche storiche di cambio in USD.
GetUserMargin	Ottieni i dati sul margine del tuo account.
GetUserWallet	Ottieni le informazioni del tuo wallet.
GetUserWalletHistory	Ottenere la cronologia delle transazioni del proprio portafoglio.
GetUserWalletSummary	Ottiene un riepilogo del Suo portafoglio.

Bitmex | Connetti API WebSocket

Per connettersi all'API WebSocket di Bitmex, creare un nuovo client Bitmex API e collegarlo a TsgcWebSocketClient.

Vedere di seguito un esempio:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bitmex oBitmex = new TsgcWSAPI_Bitmex();
oBitmex.Client = oClient;
oClient.Active = true;
```

Bitmex | Sottoscrizione al canale WebSocket

Bitmex offre una varietà di canali a cui è possibile iscriversi per ricevere aggiornamenti in tempo reale su dati di mercato, ordini... Di seguito è riportato un esempio di come iscriversi a un canale Trade:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bitmex oBitmex = new TsgcWSAPI_Bitmex();
oBitmex.Client = oClient;
oBitmex.Subscribe(btmTrade, "xbtusd");
void OnBitmexMessage(Sender: TObject; const aTopic: TwsBitmexTopics; const aMessage: string)
{
    // here you will receive the tradeupdates
}
```

Bitmex | Come inserire ordini

La REST API di Bitmex offre endpoint pubblici e privati. Gli endpoint privati richiedono che i messaggi siano firmati per aumentare la sicurezza delle transazioni.

Per prima cosa deve accedere al Suo account Bitmex e creare una nuova API; otterrà i seguenti valori:

- ApiKey
- ApiSecret

Questi campi devono essere configurati nella proprietà Bitmex del componente client API Bitmex. Una volta configurati, è possibile iniziare a effettuare richieste private all'API REST di Bitmex.

Tipi di Ordine

Tutti gli ordini richiedono un simbolo. Tutti gli altri campi sono facoltativi, salvo diversa indicazione.

Di seguito sono riportati i tipi di ordine validi:

- **Limit:** Il tipo di ordine predefinito. Specificare un orderQty e un prezzo.
- **Market:** Un ordine di mercato tradizionale. Un ordine di mercato verrà eseguito fino al riempimento o finché non viene raggiunto il prezzo di liquidazione, a quel punto verrà annullato.
- **Stop:** Un ordine Stop Market. Specificare orderQty e stopPx. Quando viene raggiunto stopPx, l'ordine verrà inserito nel book.
 - Per gli ordini di vendita, l'ordine verrà attivato se il prezzo di attivazione è inferiore a stopPx. Per gli acquisti, se è superiore.
 - Nota: gli ordini Stop non consumano margine fino all'attivazione. Assicurarsi che il margine richiesto sia disponibile nel proprio account affinché possa essere attivato completamente.
 - Gli ordini Stop di chiusura non richiedono un orderQty. Vedere le Istruzioni di esecuzione di seguito.
- **StopLimit:** Come uno Stop Market, ma immette un ordine Limit invece di un ordine Market. Specificare orderQty, stopPx e price.
- **MarketIfTouched:** Simile a uno Stop, ma gli attivatori avvengono nella direzione opposta. Utile per gli ordini Take Profit.
- **LimitIfTouched:** Come sopra; da utilizzare per gli ordini Take Profit Limit.
- **Pegged:** Gli ordini Pegged consentono agli utenti di impostare un prezzo limite relativo al prezzo di mercato corrente. Specificare un pegPriceType e un pegOffsetValue.
 - Gli ordini Pegged devono avere un execlnst impostato su Fixed. Ciò significa che il prezzo limite viene fissato al momento dell'accettazione dell'ordine e non cambia al variare del prezzo di riferimento.
 - PrimaryPeg: Il prezzo è impostato in relazione al prezzo near touch.
 - MarketPeg: Il prezzo è impostato in relazione al prezzo far touch.
 - Un pegPriceType inviato senza ordType viene trattato come un ordine Pegged.

Istruzioni di esecuzione

Sono supportati i seguenti execlnst. Se ne utilizza più di uno, li separi con una virgola (ad esempio LastPrice,Close).

- **ParticipateDoNotInitiate:** Noto anche come ordine Post-Only. Se questo ordine fosse stato eseguito al momento del posizionamento, verrà annullato. Questo serve a proteggerla dalla mossa del prezzo far touch mentre l'ordine è in transito. Non è pensato per speculare sulla mossa del prezzo far touch dopo l'invio — tale comportamento è considerato abusivo e viene monitorato.
- **MarkPrice, LastPrice, IndexPrice:** Utilizzati dagli ordini stop e if-touched per determinare il prezzo di attivazione. Utilizzarne solo uno. Per impostazione predefinita viene utilizzato MarkPrice. Utilizzati anche per gli ordini Pegged per definire il valore di LastPeg.
- **ReduceOnly:** Un ordine ReduceOnly può solo ridurre la propria posizione, non aumentarla. Se si dispone di un ordine limite ReduceOnly nel book degli ordini mentre la posizione viene ridotta da altri ordini, la quantità dell'ordine verrà ridotta o annullata. Se sono presenti più ordini ReduceOnly, verrà modificato per primo quello meno aggressivo.
- **Close:** Close implica ReduceOnly. Un ordine Close annullerà altri ordini limite attivi con lo stesso verso e simbolo se la quantità aperta supera la posizione corrente. Ciò è utile per gli stop: annullando questi ordini,

uno Stop Close è garantito di disporre del margine necessario per essere eseguito e può essere eseguito solo fino all'intera dimensione della posizione. Se orderQty non è specificato, un ordine Close ha un orderQty pari alla dimensione della posizione corrente.

- Si noti che un ordine Close senza orderQty richiede un verso, affinché BitMEX sappia se deve essere attivato sopra o sotto lo stopPx.
- **LastWithinMark:** Utilizzato dagli stop order con LastPrice per consentire l'attivazione degli stop solo quando:
 - Per Sell Stop Market / Stop Limit Order
 - Last Price $\leq$ Stop Price
 - Last Price $\geq$ Mark Price $\times (1 - 5\%)$
 - Per Buy Stop Market / Stop Limit Order:
 - Last Price $\geq$ Stop Price
 - Last Price $\leq$ Mark Price $\times (1 + 5\%)$
- **Fixed:** Gli ordini ancorati devono avere un execInst di Fixed. Ciò significa che il prezzo limite è impostato al momento dell'accettazione dell'ordine e non cambia al variare del prezzo di riferimento.

Pegged Orders

Gli ordini pegged consentono agli utenti di inviare un prezzo limite relativo al prezzo corrente di mercato. Il prezzo limite viene impostato una volta al momento dell'invio dell'ordine e non cambia con il prezzo di riferimento. Questo tipo di ordine non è destinato alla speculazione sul movimento del far touch dopo l'invio — tale comportamento è considerato abusivo e viene monitorato.

Gli ordini pegged hanno un ordType di Pegged e un execInst di Fixed.

È necessario fornire anche pegPriceType e pegOffsetValue:

- PrimaryPeg - il prezzo è impostato in relazione al prezzo di contatto più vicino
- MarketPeg - il prezzo è impostato in relazione al prezzo far touch

Ordini Trailing Stop Pegged

Utilizzare pegPriceType di TrailingStopPeg per creare Trailing Stop.

Il prezzo viene impostato al momento dell'invio e si aggiorna una volta al secondo se il prezzo sottostante (last/mark/index) si è mosso di oltre lo 0,1%. stopPx si sposta quindi man mano che il mercato si allontana dal peg, e si blocca man mano che il mercato si avvicina ad esso.

Utilizzi pegOffsetValue per impostare lo stopPx del Suo ordine. Il peg viene impostato sul prezzo di attivazione specificato in execInst (predefinito MarkPrice). Utilizzi un offset negativo per gli ordini stop-sell e buy-if-touched.

Richiede ordType: Stop, StopLimit, MarketIfTouched, LimitIfTouched.

Trailing Stops

È possibile utilizzare pegPriceType 'TrailingStopPeg' per creare Trailing Stop. Lo stopPx agganciato si sposta man mano che il mercato si allontana dall'ancoraggio, e si blocca quando il mercato si avvicina ad esso.

Per utilizzarlo, lo combini con pegOffsetValue per impostare lo stopPx del Suo ordine. Il peg viene impostato sul prezzo di trigger specificato nell'execInst (predefinito 'MarkPrice'). Utilizzi un offset negativo per gli ordini stop-sell e buy-if-touched.

Richiede ordType: 'Stop', 'StopLimit', 'MarketIfTouched', 'LimitIfTouched'.

Monitoraggio degli ordini

Se si desidera tenere traccia degli ID degli ordini in modo autonomo, impostare un clOrdID univoco per ogni ordine. Questo clOrdID verrà restituito come proprietà dell'ordine e di qualsiasi esecuzione correlata (incluse quelle sul WebSocket), e può essere utilizzato per recuperare o annullare l'ordine. La lunghezza massima è di 36 caratteri.

Esempi:

```
// buy market order
BITMEX.REST_API.PlaceMarketOrder(bmosBuy, "XBTUSD", 100);
// sell limit order at 45000
BITMEX.REST_API.PlaceLimitOrder(bmosSell, "XBTUSD", 100, 45000.00);
// stop order at 48000
BITMEX.REST_API.PlaceStopOrder(bmosSell, "XBTUSD", 100, 48000.00);
```


API Bitfinex

Bitfinex

Bitfinex è una delle piattaforme di trading di criptovalute più grandi e avanzate al mondo. Gli utenti possono scambiare Bitcoin, Ethereum, Ripple, EOS, Bitcoin Cash, Iota, NEO, Litecoin, Ethereum Classic...

La versione dell'API WebSocket di Bitfinex è 2.0

Ogni messaggio inviato e ricevuto tramite il canale WebSocket di Bitfinex è codificato in formato JSON

Un simbolo può essere una coppia di trading o una valuta di margine:

- I simboli delle coppie di trading si formano aggiungendo una "t" prima della coppia (es. tBTCUSD, tETHUSD).
- I simboli delle valute di margine sono formati aggiungendo una "f" prima della valuta (es. fUSD, fBTC, ...)

Dopo una connessione riuscita, viene generato l'evento **OnBitfinexConnect** e si ottiene il numero di versione dell'API Bitfinex come parametro.

È possibile chiamare il metodo **Ping** per verificare la connessione al server.

Se il server invia informazioni, queste possono essere gestite tramite l'evento **OnBitfinexInfoMessage**, dove Code e Message sono parametri contenenti informazioni sul messaggio inviato dal server. Codici di esempio:

```
20051 : Arresto/Riavvio del server WebSocket (si riconnetta)
20060 : Ingresso in modalità manutenzione. Si prega di sospendere qualsiasi attività e riprendere dopo aver ricevuto il messaggio informativo 20061 (dovrebbe richiedere al massimo 120 secondi).
20061 : Manutenzione terminata. È possibile riprendere la normale attività. Si consiglia di annullare l'iscrizione/iscriversi nuovamente a tutti i canali.
```

In caso di errore, verrà generato **OnBitfinexError** con le informazioni sull'errore. Esempi di codici di errore:

```
10000 : Evento sconosciuto
10001 : Coppia sconosciuta
```

Per modificare la configurazione, chiamare il metodo **Configuration** e passare come parametro uno dei seguenti flag:

```
CS_DEC_S = 8; // Enable all decimal as strings.
CS_TIME_S = 32; // Abilita tutti gli orari come stringhe data.
CS_SEQ_ALL = 65536; // Abilita il sequenziamento FUNZIONE BETA
CHECKSUM = 131072; // Enable checksum for every book iteration. Checks the top 25 entries for each side of the book. The checksum is a signed int.
```

Iscrizione ai canali pubblici

Esistono canali pubblici per i quali non è necessaria l'autenticazione con il server. Tutti i messaggi vengono generati tramite l'evento **OnBitfinexUpdate**.

SubscribeTicker

Il ticker fornisce una panoramica ad alto livello dello stato del mercato. Mostra il miglior bid e ask correnti, nonché il prezzo dell'ultimo scambio. Include anche informazioni come il volume giornaliero e la variazione del prezzo nell'ultimo giorno.

```
// Trading pairs
[
  CHANNEL_ID,
  [
    BID,
    BID_SIZE,
    ASK,
    ASK_SIZE,
    DAILY_CHANGE,
    DAILY_CHANGE_PERC,
    LAST_PRICE,
    VOLUME,
    HIGH,
    LOW
  ]
]
// Funding pairs
[
  CHANNEL_ID,
  [
    FRR,
    BID,
    BID_PERIOD,
    BID_SIZE,
    ASK,
    ASK_PERIOD,
    ASK_SIZE,
    DAILY_CHANGE,
    DAILY_CHANGE_PERC,
    LAST_PRICE,
    VOLUME,
    HIGH,
    LOW
  ]
]
```

SubscribeTrades

Questo canale invia un messaggio di trade ogni volta che avviene un'operazione su Bitfinex. Include tutti i dettagli pertinenti dell'operazione, come prezzo, quantità e orario.

```
// on trading pairs (ex. tBTCUSD)
[
  CHANNEL_ID,
  [
    [
      ID,
      MTS,
      AMOUNT,
      PRICE
    ],
    ...
  ]
]
// on funding currencies (ex. fUSD)
[
  CHANNEL_ID,
  [
    [
      ID,
      MTS,
      AMOUNT,
      RATE,
      PERIOD
    ],
    ...
  ]
]
```

SubscribeOrderBook

Il canale Order Books consente di tenere traccia dello stato del book degli ordini di Bitfinex. Viene fornito su base aggregata per prezzo, con precisione personalizzabile. Dopo aver ricevuto la risposta, si riceverà uno snapshot del book, seguito da aggiornamenti ad ogni modifica del book.

```
// on trading pairs (ex. tBTCUSD)
[
  CHANNEL_ID,
  [
    [
      PRICE,
      COUNT,
      AMOUNT
    ],
    ...
  ]
]

// on funding currencies (ex. fUSD)
[
  CHANNEL_ID,
  [
    [
      RATE,
      PERIOD,
      COUNT,
      AMOUNT
    ],
    ...
  ]
]
```

SubscribeRawOrderBook

Questi sono i libri più granulari.

```
// on trading pairs (ex. tBTCUSD)
[
  CHANNEL_ID,
  [
    [
      ORDER_ID,
      PRICE,
      AMOUNT
    ],
    ...
  ]
]

// on funding currencies (ex. fUSD)
[
  CHANNEL_ID,
  [
    [
      OFFER_ID,
      PERIOD,
      RATE,
      AMOUNT
    ],
    ...
  ]
]
```

SubscribeCandles

Fornisce un modo per accedere alle informazioni sui candlestick dei grafici. Intervalli di tempo:

1m: un minuto
 5m : cinque minuti
 15m : 15 minuti
 30m : 30 minuti
 1h : un'ora
 3h : 3 ore
 6h : 6 ore
 12h : 12 ore
 1D : un giorno
 7D : una settimana
 14D: due settimane
 1M : un mese

```
[
  CHANNEL_ID,
  [
    [
      MTS,
      OPEN,
      CLOSE,
      HIGH,
      LOW,
      VOLUME
    ],
    ...
  ]
]
```

Sottoscrivi canali autenticati

Questo canale consente di mantenersi aggiornati sullo stato del proprio account. Si possono ricevere aggiornamenti sulle posizioni, i saldi, gli ordini e le transazioni.

Utilizzare il metodo **Authenticate** per eseguire l'autenticazione presso il server e impostare i parametri richiesti.

Una volta autenticati, si riceveranno aggiornamenti su: ordini, posizioni, transazioni, offerte di finanziamento, crediti di finanziamento, prestiti di finanziamento, portafogli, informazioni sul saldo, informazioni sui margini, informazioni sul finanziamento, transazioni di finanziamento...

È possibile richiamare il metodo **UnAuthenticate** se si desidera effettuare il logout dal server.

API Kucoin

Kucoin

Kucoin è un exchange di criptovalute internazionale e multilingue. Offre alcune API per accedere ai dati di Kucoin. Sono supportate le seguenti API:

1. **Stream WebSocket:** consente di sottoscrivere ad alcuni metodi e ricevere dati in tempo reale. Gli eventi vengono inviati ai client dal server agli iscritti. Utilizza WebSocket come protocollo.
2. **REST API:** i client possono richiedere al server dati di mercato e dell'account. Richiede una API Key, un Secret e una Passphrase per l'autenticazione e utilizza HTTPS come protocollo.

Proprietà

L'API Kucoin dispone di 2 tipi di metodi: pubblici e privati. I metodi pubblici sono accessibili senza autenticazione, ad esempio per ottenere i prezzi dei ticker. I metodi privati relativi ai dati utente richiedono l'uso delle chiavi API Kucoin.

- **ApiKey:** è possibile richiedere una nuova chiave API nel proprio account KuCoin, è sufficiente copiare il valore in questa proprietà.
- **ApiSecret:** Il segreto API è richiesto solo per REST_API, l'api websocket richiede solo ApiKey per alcuni metodi.
- **Passphrase:** stringa richiesta per connettersi ai server Kucoin.
- **Sandbox:** se abilitato si conatterà al Demo Account Kucoin (per impostazione predefinita false).
 - **HTTPLogOptions:** memorizza in un file di testo un log delle richieste HTTP
 - **Enabled:** se abilitato, memorizzerà tutte le richieste HTTP della WebSocket API.
 - **FileName:** percorso completo del nome file in cui verranno memorizzati i log
 - **REST:** memorizza in un file di testo un log delle richieste REST API
 - **Enabled:** se abilitato, memorizzerà tutte le richieste HTTP della REST API.
 - **FileName:** percorso completo del nome file in cui verranno memorizzati i log.

Usi più comuni

- **WebSockets API**
 - [Come Connettersi alla WebSocket API](#)
 - [Come Iscrivere a un Canale WebSocket](#)
- **REST API**
 - [Come ottenere i dati di mercato](#)
 - [Come utilizzare la REST API privata](#)
 - [Come fare trading Spot](#)
 - [Tempo delle richieste private](#)

WebSocket Feed

Per sottoscrivere i messaggi di un canale da un determinato server, il lato client deve inviare un messaggio di sottoscrizione al server.

Se la sottoscrizione ha esito positivo, il sistema invierà messaggi ack quando la risposta è impostata su true.

```
{
  "id": "1545910660739",
  "type": "ack"
}
```

Mentre vengono generati messaggi di topic, il sistema invierà i messaggi corrispondenti al lato client.

Sono supportati i seguenti metodi di sottoscrizione/cancellazione della sottoscrizione.

Canali pubblici

Metodo	Parametri	Descrizione
SubscribeSymbolTicker	Symbol	Sottoscrivi a questo topic per ricevere le notifiche push delle modifiche BBO. Se non si verificano modifiche entro un secondo, non verrà inviato nulla. Verrà inviato ogni 100ms con il BBO più recente. Se non vi sono state modifiche rispetto agli ultimi dati, non verrà inviato nulla.
SubscribeAllSymbolsTicker		Si iscriva a questo topic per ricevere il push di tutte le variazioni BBO dei simboli di mercato.
SubscribeSymbolSnapshot	Symbol	Iscrizione per ricevere i dati di snapshot per un singolo simbolo. I dati di snapshot vengono inviati a intervalli di 2 secondi.
SubscribeMarketSnapshot	Mercato	Sottoscrivere questo topic per ottenere i dati snapshot dell'intero mercato. I dati snapshot vengono inviati a intervalli di 2 secondi.
SubscribeLevel2MarketData	Symbol	Si iscriva a questo topic per ottenere i dati del book ordini Level2. Quando l'iscrizione websocket ha successo, il sistema Le invierà i dati di variazione incrementale inviati dal websocket.
SubscribeLevel2_5BestAskBid	Symbol	Il sistema restituirà i dati dei 5 migliori ordini ask/bid, che sono i dati snapshot ogni 100 millisecondi (in altre parole, i dati dei 5 migliori ordini ask/bid restituiti ogni 100 millisecondi in tempo reale).
SubscribeLevel2_50BestAskBid	Symbol	Il sistema restituirà i dati dei 50 migliori ordini ask/bid, che sono i dati snapshot di ogni 100 millisecondi (in altre parole, i dati dei 50 migliori ordini ask/bid restituiti ogni 100 millisecondi in tempo reale).
SubscribeKlines	Symbol	Sottoscrivere questo topic per ricevere i dati K-Line.
SubscribeMatchExecutionData	Symbol	Sottoscrivere questo topic per ricevere il flusso di dati degli eventi corrispondenti del Livello 3. Per ogni ordine eseguito, il sistema invierà i messaggi di corrispondenza nel seguente formato.
SubscribeIndexPrice	Symbol	Sottoscrivere a questo topic per ottenere il prezzo dell'indice per il trading su margine.
SubscribeMarkPrice	Symbol	Iscrizione a questo topic per ricevere il prezzo mark per il trading a margine.
SubscribeOrderBookChanged	Symbol	Sottoscrivi questo argomento per ricevere le modifiche al book degli ordini sul trading a margine.
SubscribeLevel1	Symbol	Sottoscrivere ai dati best bid/ask di Livello 1 per un simbolo.

Se il parametro ACK viene inviato a true, dopo una sottoscrizione/annullamento della sottoscrizione riuscita, il client riceve un messaggio in merito.

Canali privati

Richiede una ApiKey valida ottenuta dal Suo account Kucoin. ApiKey, ApiSecret e Passphrase devono essere impostati nella proprietà Kucoin del componente API client.

I seguenti dati vengono inviati al client ogni volta che si verifica una modifica. Non è necessario sottoscrivere alcun metodo; questa operazione viene eseguita automaticamente se si imposta un ApiKey valido.

Metodo	Descrizione
SubscribeTradeOrders	Questo topic invierà tutti gli eventi di modifica degli ordini.
SubscribeAccountBalance	Si riceverà questo messaggio quando cambia il saldo di un account. Il messaggio contiene i dettagli della modifica.
SubscribePositionStatus	Il sistema invierà l'evento di modifica quando lo stato della posizione cambia.
SubscribeMarginTradeOrders	Il sistema invierà questo messaggio ai prestatori quando l'ordine entra nel libro degli ordini.
SubscribeStopOrder	Quando un ordine stop viene ricevuto dal sistema, si riceverà un messaggio con tipo "open". Ciò significa che l'ordine è entrato nel sistema ed è in attesa di essere attivato.
SubscribeTradeOrder-v2	Sottoscrive il canale V2 degli ordini di trade per notifiche avanzate di aggiornamento degli ordini.
SubscribeCrossMarginPosition	Sottoscrizione agli aggiornamenti della posizione su margine incrociato. Il sistema invierà l'evento di modifica quando la posizione su margine incrociato cambia.
SubscribeIsolatedMarginPosition	Sottoscrivi gli aggiornamenti delle posizioni in margin isolato. Il sistema invierà l'evento di modifica quando la posizione in margin isolato cambia.

REST API

Tutti gli endpoint restituiscono un oggetto o un array JSON.

EndPoint API Pubblici

Questi endpoint sono accessibili senza alcuna autorizzazione.

EndPoint generali

Metodo	Parametri	Descrizione
GetServiceStatus		Verifica la connettività all'API Rest e ottieni lo stato del servizio
GetServerTime		Verifica la connettività alla Rest API e ottiene l'ora corrente del server.

EndPoint dei dati di mercato

Metodo	Parametri	Descrizione
GetSymbolList	Mercato	Effettui una richiesta tramite questo endpoint per ottenere un elenco delle coppie di valute disponibili per il trading. Se desidera ottenere le informazioni di mercato del simbolo di trading
GetTicker	Symbol	Richiedere tramite questo endpoint i Dati di Mercato di Livello 1. Il valore restituito include il miglior prezzo e dimensione in acquisto, il miglior prezzo e dimensione in vendita, nonché l'ultimo prezzo negoziato e l'ultima dimensione negoziata.
GetAllTickers		Richiedere i ticker di mercato per tutte le coppie di trading nel mercato (incluso il volume delle ultime 24 ore).

Get24hrStats	Symbol	Richiedere tramite questo endpoint per ottenere le statistiche del ticker specificato nelle ultime 24 ore.
GetMarketList		Richiedere tramite questo endpoint per ottenere la valuta delle transazioni per l'intero mercato di trading.
GetPartOrder-Book20	Symbol	Effettuare una richiesta tramite questo endpoint per ottenere un elenco degli ordini aperti per un simbolo. Il book degli ordini di livello 2 include tutte le offerte e domande (aggregate per prezzo); questo livello restituisce una sola dimensione per ogni prezzo attivo (come se ci fosse un solo ordine per quel prezzo). Il sistema restituirà 20 dati (dati ask e bid) del book degli ordini.
GetPartOrder-Book100	Symbol	Effettua una richiesta tramite questo endpoint per ottenere un elenco di ordini aperti per un simbolo. Il book degli ordini di Livello 2 include tutti i bid e gli ask (aggregati per prezzo); questo livello restituisce una sola dimensione per ogni prezzo attivo (come se ci fosse un solo ordine per quel prezzo). Il sistema restituirà 100 record di dati (dati ask e bid) del book degli ordini.
GetFullOrder-Book	Symbol	Richiedere tramite questo endpoint per ottenere il book degli ordini del simbolo specificato. Il book degli ordini di livello 2 include tutte le offerte di acquisto e di vendita (aggregate per prezzo). Questo livello restituisce una sola dimensione aggregata per ciascun prezzo (come se ci fosse un solo ordine per quel prezzo). Questa API restituirà dati con profondità completa.
GetKLines	Symbol	Effettua una richiesta tramite questo endpoint per ottenere la kline del simbolo specificato. I dati vengono restituiti in bucket raggruppati in base al tipo richiesto.
GetCurrencies		Effettuare la richiesta tramite questo endpoint per ottenere l'elenco delle valute.
GetCurrency-Detail	Valuta	Richiedere tramite questo endpoint per ottenere i dettagli della valuta di una valuta specificata
GetFiatPrice		Richiedere tramite questo endpoint per ottenere i dettagli della valuta di una valuta specificata
GetPartOrder-Book1	Symbol	Effettui una richiesta tramite questo endpoint per ottenere il miglior prezzo bid/ask di Livello 1 per un simbolo.

EndPoint API privati

Richiede un APIKey e un APISecret per essere autorizzato dal server.

EndPoint Utente

Metodo	Parametri	Descrizione
GetAllSubAccounts		È possibile ottenere le informazioni utente di tutti i sotto-utenti tramite questa interfaccia.
GetListAccounts		Ottiene un elenco di account.
GetAccount	AccountId	Informazioni per un singolo account. Utilizzare questo endpoint quando si conosce l'accountId.

GetAccountBalanceSubAccount	SubUserId	Questo endpoint restituisce le informazioni sull'account di un sotto-utente specificato tramite subUserId.
InnerTransfer		Questo endpoint API può essere utilizzato per trasferire fondi tra account internamente. Gli utenti possono trasferire fondi tra il proprio account principale, account di trading, account margin cross e account margin isolato senza costi. È supportato il trasferimento di fondi dall'account principale, dall'account margin cross e dall'account di trading all'account futures, ma non è supportato il trasferimento di fondi dagli account futures verso altri account.
GetDepositAddresses	Valuta	Ottieni gli indirizzi di deposito per una valuta.
CreateDepositAddress	Valuta	Creare un nuovo indirizzo di deposito per una valuta.
GetDepositList		Ottiene la cronologia dei depositi.
GetAccountLedgers		Ottiene le voci del ledger dell'account.
GetTradeFees	Symbols	Recupera le commissioni di trading per i simboli specificati.

Endpoint di prelievo

Metodo	Parametri	Descrizione
GetWithdrawalList		Ottieni un elenco dei prelievi.
GetHistoricalWithdrawalsList		Elenco dei prelievi storici V1 di KuCoin.
GetWithdrawalQuotas	Valuta	Ottieni le quote di prelievo
ApplyWithdraw	Currency, Address, Amount	Crea un prelievo
CancelWithdraw	WithdrawalId	Solo le richieste di prelievo con stato PROCESSING possono essere annullate.

Endpoint di trading

Metodo	Parametri	Descrizione
PlaceOrder		È possibile inserire due tipi di ordini: limite e mercato. Gli ordini possono essere inseriti solo se il conto dispone di fondi sufficienti. Una volta inserito un ordine, i fondi del conto vengono bloccati per la durata dell'ordine. L'importo e il tipo di fondi bloccati dipendono dal tipo di ordine e dai parametri specificati
PlaceMarketOrder		Inserisce un ordine a mercato.
PlaceLimitOrder		Inserisce un Ordine Limit.

PlaceMarginOrder		Inserisce un Ordine a Margine.
CancelOrder		Annulla un ordine tramite ID ordine.
CancelOrderBy-ClientOid		Annulla un Order tramite il Client Order Id.
CancelAllOrders		Annulla tutti gli ordini aperti.
ListOrders		Richiedere tramite questo endpoint per ottenere l'elenco degli ordini correnti. Gli elementi sono paginati e ordinati per mostrare i più recenti per primi
GetRecentOrders		Richiedere tramite questo endpoint per ottenere 1000 ordini nelle ultime 24 ore.
GetOrder		Effettuare una richiesta tramite questo endpoint per ottenere le informazioni su un singolo ordine tramite il suo ID.
GetOrderBy-ClientOid		Richiedere tramite questo endpoint per ottenere le informazioni di un singolo ordine tramite il Client order ID.
ListFills		Richiedere tramite questo endpoint per ottenere i fill recenti.
GetRecentFills		Effettuare una richiesta tramite questo endpoint per ottenere un elenco di 1000 operazioni nelle ultime 24 ore.
PlaceStopOrder		Inserisce un ordine Stop.
PlaceStopMarketOrder		Inserisce un ordine Stop Market.
PlaceStopLimitOrder		Inserisce un ordine Stop Limit.
CancelStopOrder		Annulla un ordine Stop aperto tramite l'ID ordine
CancelStopOrderByClientOid		Annulla un ordine Stop aperto tramite l'ID ordine del client
CancelAllStopOrders		Annulla tutti gli ordini Stop
GetStopOrder		Richiede tramite questa interfaccia le informazioni di uno stop order tramite l'ID dell'ordine.
GetStopOrder-ByClientOid		Richiedere tramite questa interfaccia le informazioni su un ordine stop tramite l'ID ordine client.
ListStopOrders		Richiedere tramite questo endpoint per ottenere l'elenco attuale degli ordini stop non ancora attivati. Gli elementi sono paginati e ordinati in modo da mostrare i più recenti per primi.
PlaceHFOOrder		Inserire un ordine ad alta frequenza.
CancelHFOOrder		Annulla un ordine ad alta frequenza tramite ID ordine.
CancelHFOrderByClientOid		Annulla un ordine ad alta frequenza tramite l'ID dell'ordine del client.
CancelAllHFOrders		Annulla tutti gli ordini ad alta frequenza.
GetHFActiveOrders		Ottieni gli ordini ad alta frequenza attivi.
GetHFDoneOrders		Ottiene gli ordini ad alta frequenza completati.
GetHFOOrder		Ottieni un ordine ad alta frequenza specifico tramite l'ID ordine.

Eventi

I messaggi Kucoin vengono ricevuti nel componente `TsgcWebSocketClient`; è possibile utilizzare i seguenti eventi:

OnConnect

Dopo una connessione riuscita al server Kucoin.

OnDisconnect

Dopo una disconnessione dal server Kucoin

OnMessage

I messaggi inviati dal server al client vengono gestiti in questo evento.

OnError

In caso di qualsiasi errore nel protocollo, verrà chiamato questo evento.

OnException

Se si verifica un'eccezione non gestita, verrà chiamato questo evento.

Inoltre, è disponibile un evento specifico nel componente API Kucoin, denominato **OnKcoinHTTPException**, che viene generato ogni volta che si verifica un errore durante la chiamata a una richiesta HTTP (REST API o WebSocket Feed).

Kucoin | Connetti WebSocket API

Per connettersi all'API WebSocket di Kucoin, è sufficiente creare un nuovo client API Kucoin e collegarlo a Tsgc-WebSocketClient.

Vedere di seguito un esempio:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Kucoin oKucoin = new TsgcWSAPI_Kucoin();
oKucoin.Client = oClient;
oClient.Active = true;
```

Kucoin | Subscribe WebSocket Channel

Kucoin offre una varietà di canali a cui è possibile sottoscrivere per ricevere aggiornamenti in tempo reale sui dati di mercato, ordini... Di seguito un esempio di come sottoscrivere a un Ticker:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Kucoin oKucoin = new TsgcWSAPI_Kucoin();
oKucoin.Client = oClient;
oKucoin.SubscribeSymbolTicker("BTC-USDT");
void OnMessage(TsgcWSConnection Connection, const string aText)
{
    // here you will receive the ticker updates
}
```

Kucoin | Ottenere dati di mercato

Kucoin offre dati di mercato pubblici tramite endpoint REST; quando si chiama uno di questi endpoint, si ottiene un'istantanea dei dati di mercato richiesti.

Gli endpoint di dati di mercato non richiedono autenticazione e sono quindi liberamente accessibili a tutti gli utenti.

Esempio: per ottenere lo snapshot del ticker BTC-USDT, effettuare la seguente chiamata

```
TsgcWSAPI_Kucoin oKucoin = new TsgcWSAPI_Kucoin();  
MessageBox.Show(oKucoin.REST_API.GetTicker("BTC-USDT  
"));
```

Kucoin | Private REST API

La REST API di Kucoin offre endpoint pubblici e privati. Gli endpoint privati richiedono che i messaggi siano firmati per aumentare la sicurezza delle transazioni.

Per prima cosa è necessario accedere al proprio account Kucoin e creare una nuova API; si otterranno i seguenti valori:

- ApiKey
- ApiSecret
- Passphrase

Questi campi devono essere configurati nella proprietà Kucoin del componente client API Kucoin. Una volta configurato, è possibile iniziare a effettuare richieste private alle API REST di Kucoin Pro

*Le richieste Private richiedono che la macchina locale abbia l'orario locale sincronizzato; in caso contrario, le richieste verranno rifiutate dal server Kucoin. Consultare il seguente articolo a riguardo: [Kucoin Private Requests Time](#).

```
TsgcWSAPI_Kucoin oKucoin = new TsgcWSAPI_Kucoin();
oKucoin.Kucoin.ApiKey = "<your api key>";
oKucoin.Kucoin.ApiSecret = "<your api secret>";
oKucoin.Kucoin.Passphrase = "<your passphrase>";
MessageBox.Show(oKucoin.REST_API.GetListAccounts());
```

Kucoin | Trade Spot

Kucoin consente di fare trading spot tramite la sua REST API.

Configurazione

Per prima cosa deve creare una **API Key** nel suo account Kucoin e aggiungere i privilegi per il trading con Spot. Una volta completato, è possibile iniziare il trading spot.

Prima, **impostare ApiKey, ApiSecret e Passphrase** nel Componente Client Kucoin; questi verranno utilizzati per firmare le richieste inviate al server Kucoin.

Inserisci un Ordine

Per inserire un nuovo ordine, chiamare il metodo **REST_API.PlaceOrder** del componente client Kucoin.

A seconda del tipo di ordine (market, limit...) l'API richiede più o meno campi.

Parametri

Pa-ram	ti-po	Descrizione
clien-tOid	String	ID ordine univoco creato dagli utenti per identificare i propri ordini, ad esempio UUID.
side	String	buy o sell
symbol	String	un codice di simbolo di trading valido. es. ETH-BTC
tipo	String	[Opzionale] limit o market (predefinito è limit)
nota	String	[Facoltativo] nota per l'ordine, la lunghezza non può superare i 100 caratteri utf8
stp	String	[Facoltativo] prevenzione del self trade, CN , CO , CB o DC
trade-Type	String	[Opzionale] Il tipo di trading: TRADE (Trading Spot) , MARGIN_TRADE (Trading a Margine). Il valore predefinito è TRADE . Nota: Per migliorare le prestazioni del sistema e accelerare l'inserimento e l'elaborazione degli ordini, KuCoin ha aggiunto una nuova interfaccia per l'inserimento degli ordini a margine. Per i trader che utilizzano ancora l'interfaccia attuale, si prega di passare a quella nuova il prima possibile. Quella attuale non accetterà più ordini a margine dal 1° maggio 2021 (UTC). In quel momento, KuCoin informerà gli utenti tramite annuncio; si prega di prestare attenzione.

PARAMETRI ORDINE LIMITE

Param	tipo	Descrizione
price	String	prezzo per valuta di base
size	String	quantità di valuta di base da acquistare o vendere
timeInForce	String	[Facoltativo] GTC , GTT , IOC , oppure FOK (il valore predefinito è GTC), consultare Time In Force .
cancelAfter	long	[Facoltativo] annulla dopo n secondi, richiede che timeInForce sia GTT

Param	tipo	Descrizione
postOnly	boolean	<i>[Opzionale]</i> Flag post-only, non valido quando timeInForce è IOC o FOK
hidden	boolean	<i>[Opzionale]</i> L'ordine non verrà visualizzato nel libro degli ordini
iceberg	boolean	<i>[Opzionale]</i> Solo una parte dell'ordine viene visualizzata nell'order book
visibleSize	String	<i>[Facoltativo]</i> La dimensione massima visibile di un ordine iceberg

PARAMETRI ORDINE DI MERCATO

Param	tipo	Descrizione
size	String	<i>[Opzionale]</i> Importo desiderato nella valuta di base
fondi	String	<i>[Opzionale]</i> La quantità desiderata di valuta di quotazione da utilizzare

Quando si invia un ordine, ci sono 2 possibilità:

1. **Esito positivo:** la funzione PlaceOrder restituisce il messaggio inviato dal server Kucoin.
2. **Error:** l'eccezione viene restituita nell'evento OnKcoinHTTPException.

Esegui ordine a mercato 1 BTC-USDT

```
TsgcWSAPI_Kucoin oKucoin = new TsgcWSAPI_Kucoin();
oKucoin.Kucoin.ApiKey = "<api key>";
oKucoin.Kucoin.ApiSecret = "<api secret>";
oKucoin.Kucoin.Passphrase = "<passphrase>";
MessageBox.Show(oKucoin.REST_API.

PlaceMarketOrder

(
    kosBuy, "BTC-USDT", 1
));
```

Inserisci ordine limite 1 BTC-USDT a 40000

```
TsgcWSAPI_Kucoin oKucoin = new TsgcWSAPI_Kucoin();
oKucoin.Kucoin.ApiKey = "<api key>";
oKucoin.Kucoin.ApiSecret = "<api secret>";
oKucoin.Kucoin.Passphrase = "<passphrase>";
MessageBox.Show(oKucoin.REST_API.

PlaceLimitOrder

(
    kosBuy, "BTC-USDT", 1, 40000
));
```

Kucoin | Tempo delle Richieste Private

Quando si effettua una richiesta privata a Kucoin, il messaggio viene firmato per aumentare la sicurezza delle richieste. Il messaggio prende l'ora locale e la invia all'interno del messaggio firmato; se l'ora locale presenta una differenza superiore a 5 secondi rispetto ai server Kucoin, la richiesta verrà rifiutata. Pertanto, è importante verificare che la propria ora locale sia sincronizzata; è possibile farlo utilizzando il metodo di sincronizzazione dell'ora del proprio sistema operativo.

È possibile verificare l'orario del server Kucoin chiamando il metodo **GetServerTime**, che restituirà l'orario del server Kucoin

API Kucoin Futures

Kucoin Futures

Kucoin è un exchange di criptovalute internazionale e multilingue. Offre alcune API per accedere ai dati di Kucoin. Sono supportate le seguenti API:

1. **Stream WebSocket:** consente di sottoscrivere ad alcuni metodi e ricevere dati in tempo reale. Gli eventi vengono inviati ai client dal server agli iscritti. Utilizza WebSocket come protocollo.
2. **REST API:** i client possono richiedere al server dati di mercato e dell'account. Richiede una API Key, un Secret e una Passphrase per l'autenticazione e utilizza HTTPS come protocollo.

Proprietà

L'API Kucoin ha 2 tipi di metodi: pubblici e privati. I metodi pubblici sono accessibili senza autenticazione, ad esempio: ottenere i prezzi dei ticker. I metodi privati e relativi ai dati dell'utente richiedono l'uso delle chiavi API Kucoin.

- **ApiKey:** è possibile richiedere una nuova chiave API nel proprio account KuCoin, è sufficiente copiare il valore in questa proprietà.
- **ApiSecret:** Il segreto API è richiesto solo per REST_API, l'api websocket richiede solo ApiKey per alcuni metodi.
- **Passphrase:** stringa richiesta per connettersi ai server Kucoin.
- **Sandbox:** se abilitato si conatterà al Demo Account Kucoin (per impostazione predefinita false).
 - **HTTPLogOptions:** memorizza in un file di testo un log delle richieste HTTP
 - **Enabled:** se abilitato, memorizzerà tutte le richieste HTTP della WebSocket API.
 - **FileName:** percorso completo del nome file in cui verranno memorizzati i log
 - **REST:** memorizza in un file di testo un log delle richieste REST API
 - **Enabled:** se abilitato, memorizzerà tutte le richieste HTTP della REST API.
 - **FileName:** percorso completo del nome file in cui verranno memorizzati i log.

Usi più comuni

- **WebSockets API**
 - [Come connettersi alla WebSocket API](#)
 - [Come abbonarsi a un canale WebSocket](#)
- **REST API**
 - [Come ottenere i dati di mercato](#)
 - [Come utilizzare l'API REST privata](#)
 - [Come fare trading di futures](#)
 - [Tempo delle richieste private](#)

WebSocket Feed

Per sottoscrivere i messaggi di un canale da un determinato server, il lato client deve inviare un messaggio di sottoscrizione al server.

Se la sottoscrizione ha esito positivo, il sistema invierà messaggi ack quando la risposta è impostata su true.

```
{
  "id": "1545910660739",
  "type": "ack"
}
```

Mentre vengono generati messaggi di topic, il sistema invierà i messaggi corrispondenti al lato client.

Sono supportati i seguenti metodi di sottoscrizione/cancellazione della sottoscrizione.

Canali pubblici

Metodo	Parametri	Descrizione
SubscribeSymbolTickerV2	Symbol	Sottoscriviti a questo topic per ricevere gli aggiornamenti in tempo reale delle variazioni BBO. Dopo la sottoscrizione, quando si verificano modifiche nel book degli ordini, il sistema invierà in tempo reale le informazioni sul simbolo del ticker. Si consiglia di utilizzare il nuovo topic per informazioni tempestive.
SubscribeSymbolTicker	Symbol	Sottoscrivere questo topic per ricevere gli aggiornamenti in tempo reale delle variazioni BBO. Il canale ticker fornisce aggiornamenti dei prezzi in tempo reale ogni volta che avviene un'esecuzione. Se vengono abbinati più ordini contemporaneamente, verrà inviato solo l'ultimo evento di abbinamento.
SubscribeLevel2MarketData	Symbol	Sottoscrivere questo topic per ottenere i dati dell'order book di Livello 2.
SubscribeExecutionData	Symbol	Per ogni ordine eseguito, il sistema invierà i messaggi di corrispondenza nel formato seguente.
SubscribeLevel2_5BestAskBid	Symbol	Restituito al massimo ogni 100 millisecondi.
SubscribeLevel2_50BestAskBid	Symbol	Restituito al massimo ogni 100 millisecondi.
SubscribeContractMarketData	Symbol	Sottoscrivere questo topic per ottenere i dati di mercato del contratto.
SubscribeSystemAnnouncements	Symbol	Sottoscrivere questo topic per ricevere gli annunci di sistema.
SubscribeTransactionStatistics	Symbol	Le statistiche delle transazioni verranno inviate agli utenti ogni 5 secondi.
SubscribeKlines	Symbol	Effettua la sottoscrizione ai dati delle klines (candlestick) dei contratti.
SubscribeFundingFeeSettlement	Symbol	Sottoscrivere le notifiche di liquidazione delle commissioni di finanziamento.

Se il parametro ACK viene inviato a true, dopo una sottoscrizione/annullamento della sottoscrizione riuscita, il client riceve un messaggio in merito.

Canali privati

Richiede una ApiKey valida ottenuta dal Suo account Kucoin. ApiKey, ApiSecret e Passphrase devono essere impostati nella proprietà Kucoin del componente API client.

I seguenti dati vengono inviati al client ogni volta che si verifica una modifica. Non è necessario sottoscrivere alcun metodo; questa operazione viene eseguita automaticamente se si imposta un ApiKey valido.

Metodo	Descrizione
SubscribeTradeOrders	Questo topic invierà tutti gli eventi di modifica degli ordini.
SubscribeAccountBalance	Si riceverà questo messaggio quando cambia il saldo di un account. Il messaggio contiene i dettagli della modifica.
SubscribePositionChange	Il sistema invierà l'evento di modifica quando lo stato della posizione cambia.

SubscribeStopOrder	Quando un ordine stop viene ricevuto dal sistema, si riceverà un messaggio con tipo "open". Ciò significa che l'ordine è entrato nel sistema ed è in attesa di essere attivato.
SubscribeMarginMode	Sottoscrizione alle modifiche della modalità di margine. Il sistema invierà l'evento di modifica quando la modalità di margine viene aggiornata.
SubscribeCrossMargin-Leverage	Iscriversi alle variazioni della leva cross margin. Il sistema invierà l'evento di modifica quando la leva cross margin viene aggiornata.

REST API

Tutti gli endpoint restituiscono un oggetto o un array JSON.

EndPoint API Pubblici

Questi endpoint sono accessibili senza alcuna autorizzazione.

EndPoint generali

Metodo	Parametri	Descrizione
GetServiceStatus		Verifica la connettività all'API Rest e ottieni lo stato del servizio
GetServerTime		Verifica la connettività alla Rest API e ottiene l'ora corrente del server.

EndPoint dei dati di mercato

Metodo	Parametri	Descrizione
GetOpenContractList		Invia una richiesta per ottenere le informazioni su tutti i contratti aperti.
GetOrderInfoContract		Invia una richiesta per ottenere informazioni sul contratto specificato.
GetTicker	Symbol	Il ticker in tempo reale include l'ultimo prezzo negoziato, l'ultima dimensione negoziata, l'ID transazione, il lato del taker di liquidità, il miglior prezzo e dimensione bid, il miglior prezzo e dimensione ask, nonché il momento della transazione degli ordini. Questi messaggi possono essere ottenuti anche tramite WebSocket. Il Sequence Number viene utilizzato per verificare se i messaggi inviati tramite WebSocket sono continui.
GetPartOrderBook20	Symbol	Ottieni uno snapshot degli ordini aperti aggregati per un simbolo.
GetPartOrderBook100	Symbol	Ottieni uno snapshot degli ordini aperti aggregati per un simbolo.
GetFullOrderBook	Symbol	Ottieni uno snapshot degli ordini aperti aggregati per un simbolo.

GetLevel2PullingMessages	Symbol	Se i messaggi inviati tramite WebSocket non sono continui, è possibile inviare la seguente richiesta e recuperare nuovamente i dati per garantire che la sequenza non presenti lacune. Nella richiesta, il parametro start corrisponde al numero di sequenza dell'ultimo messaggio ricevuto più 1, mentre il parametro end corrisponde al numero di sequenza del messaggio ricevuto corrente meno 1. Dopo aver recuperato nuovamente i messaggi e averli applicati al proprio order book locale, è possibile continuare ad aggiornare l'order book tramite il feed incrementale WebSocket. Se la differenza tra il parametro end e il parametro start è superiore a 500, interrompere l'utilizzo di questa richiesta e si consiglia di ricostruire il Level 2 orderbook.
GetTradeHistory	Symbol	Elenca le ultime 100 operazioni per un simbolo.
GetInterestRateList	Symbol	Verificare l'elenco dei tassi di interesse.
GetIndexList	Symbol	Verifica elenco indici
GetCurrentMarkPrice	Symbol	Verifica il mark price corrente.
GetPremiumIndex	Symbol	Invia una richiesta per ottenere l'indice premium.
GetCurrentFundingRate	Symbol	Invia richiesta per verificare il prezzo mark corrente.
GetKLine	Symbol	Ottieni i Dati K Line del Contratto

EndPoint API privati

Richiede un APIKey e un APISecret per essere autorizzato dal server.

EndPoint Utente

Metodo	Parametri	Descrizione
GetAccountOverview		Ottieni panoramica dell'account
GetTransactionHistory		Se ci sono posizioni aperte, lo stato della prima pagina restituita sarà Pending, indicando il profitto e la perdita realizzati nel periodo di liquidazione corrente di 8 ore. Specificare il numero di offset minimo della pagina corrente nel campo offset per voltare pagina.

Endpoint di trading

Metodo	Parametri	Descrizione
PlaceOrder		Può inserire due tipi di ordini: limit e market. Gli ordini possono essere inseriti solo se il suo account dispone di fondi sufficienti. Una volta inserito un ordine, i suoi fondi verranno bloccati per la durata dell'ordine. L'importo dei fondi bloccati dipende dal tipo di ordine e dai parametri specificati.

PlaceMarketOrder		Inserisce un ordine a mercato.
PlaceLimitOrder		Inserisce un Ordine Limit.
CancelOrder		Annulla un ordine tramite ID ordine.
LimitOrderMassCancellation		Annulla tutti gli ordini aperti (esclusi gli ordini stop). La risposta è un elenco di orderID degli ordini annullati.
StopOrderMassCancellation		Annulla tutti gli ordini stop non attivati. La risposta è un elenco di orderID degli ordini stop annullati. Per annullare gli ordini stop attivati, utilizzare 'Limit Order Mass Cancellation'.
GetOrderList		Elenca gli ordini correnti.
GetUntriggeredStopOrderList		Ottiene l'elenco degli ordini stop non attivati.
GetListOrdersCompleted24hr		Ottieni un elenco dei 1000 ordini recenti nelle ultime 24 ore. Se è necessario ottenere la cronologia degli ordini negoziati di recente con bassa latenza, è possibile interrogare questo endpoint.
GetOrder		Recupero di un singolo ordine tramite id ordine (incluso un ordine stop).
GetOrderByClientOid		Recupera un singolo ordine tramite l'ID ordine del client (incluso un ordine stop).
GetFills		Recupera un elenco delle transazioni recenti.
GetRecentFills		Ottieni un elenco degli ultimi 1000 eseguiti nelle ultime 24 ore. Se è necessario ottenere la cronologia degli ordini negoziati di recente con bassa latenza, è possibile interrogare questo endpoint.
ActiveOrderValueCalculation		È possibile interrogare questo endpoint per ottenere il numero totale e il valore di tutti gli ordini attivi.
GetPositionDetails		Ottieni i dettagli di una posizione specificata.
GetPositionList		Ottieni i dettagli di una posizione specificata.
AutoDepositMargin		Abilitazione/Disabilitazione del margine a deposito automatico
AddMarginManually		Aggiungi Margine Manualmente
ObtainFuturesRiskLimitLevel		Questa interfaccia può essere utilizzata per ottenere informazioni sul livello del limite di rischio di uno specifico contratto
AdjustRiskLimitLevel		Questa interfaccia serve per la regolazione del livello del limite di rischio. La regolazione del livello annullerà l'ordine aperto; la risposta può solo indicare se l'invio della richiesta di regolazione è andato a buon fine o meno.
GetFundingHistory		Invia richiesta per ottenere la cronologia del finanziamento.
GetMaxOpenSize		Ottieni la dimensione massima della posizione aperta per un contratto.
SwitchMarginMode		Passa dalla modalità cross margin alla modalità isolated margin e viceversa.
GetMarginMode		Ottiene la modalità di margine corrente per un contratto.

Eventi

I messaggi Kucoin vengono ricevuti nel componente `TsgcWebSocketClient`; è possibile utilizzare i seguenti eventi:

OnConnect

Dopo una connessione riuscita al server Kucoin.

OnDisconnect

Dopo una disconnessione dal server Kucoin

OnMessage

I messaggi inviati dal server al client vengono gestiti in questo evento.

OnError

In caso di qualsiasi errore nel protocollo, verrà chiamato questo evento.

OnException

Se si verifica un'eccezione non gestita, verrà chiamato questo evento.

Inoltre, è disponibile un evento specifico nel componente API Kucoin, denominato **OnKucoinHTTPException**, che viene generato ogni volta che si verifica un errore durante la chiamata a una richiesta HTTP (REST API o WebSocket Feed).

Kucoin | Futures Connect WebSocket API

Per connettersi all'API WebSocket di Kucoin, è sufficiente creare un nuovo client API Kucoin e collegarlo a Tsgc-WebSocketClient.

Vedere di seguito un esempio:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();  
TsgcWSAPI_Kucoin_Futures  
oKucoin = new  
TsgcWSAPI_Kucoin_Futures  
(  
);  
oKucoin.Client = oClient;  
oClient.Active = true;
```

Kucoin | Futures Sottoscrizione Canale WebSocket

Kucoin offre una varietà di canali a cui è possibile sottoscrivere per ricevere aggiornamenti in tempo reale sui dati di mercato, ordini... Di seguito un esempio di come sottoscrivere a un Ticker:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Kucoin oKucoin = new TsgcWSAPI_Kucoin();
oKucoin.Client = oClient;
oKucoin.SubscribeSymbolTickerV2("XBTUSDM");
void OnMessage(TsgcWSConnection Connection, const string aText)
{
    // here you will receive the ticker updates
}
```

Kucoin | Futures Ottieni Dati di Mercato

Kucoin offre dati di mercato pubblici tramite endpoint REST; quando si chiama uno di questi endpoint, si ottiene un'istantanea dei dati di mercato richiesti.

Gli endpoint di dati di mercato non richiedono autenticazione e sono quindi liberamente accessibili a tutti gli utenti.

Esempio: per ottenere lo snapshot del ticker BTC-USDT, effettuare la seguente chiamata

```
TsgcWSAPI_Kucoin_Futures  
oKucoin = new  
TsgcWSAPI_Kucoin_Futures  
(  
);  
MessageBox.Show(oKucoin.REST_API.GetTicker("XBTUSDM  
"));
```

Kucoin | REST API Private Futures

La REST API di Kucoin offre endpoint pubblici e privati. Gli endpoint privati richiedono che i messaggi siano firmati per aumentare la sicurezza delle transazioni.

Per prima cosa è necessario accedere al proprio account Kucoin e creare una nuova API; si otterranno i seguenti valori:

- ApiKey
- ApiSecret
- Passphrase

Questi campi devono essere configurati nella proprietà Kucoin del componente client API Kucoin. Una volta configurato, è possibile iniziare a effettuare richieste private alle API REST di Kucoin Pro

*Le richieste private richiedono che il computer locale abbia l'orario locale sincronizzato; in caso contrario, le richieste verranno rifiutate dal server Kucoin. Consultare il seguente articolo a riguardo, [Kucoin Private Requests Time](#).

```
TsgcWSAPI_Kucoin_Futures  
oKucoin = new  
TsgcWSAPI_Kucoin_Futures  
(  
    oKucoin.Kucoin.ApiKey = "<your api key>";  
    oKucoin.Kucoin.ApiSecret = "<your api secret>";  
    oKucoin.Kucoin.Passphrase = "<your passphrase>";  
    MessageBox.Show(oKucoin.REST_API.GetAccountOverview());  
);
```

Kucoin | Futures Trade

Kucoin consente di fare trading di Futures tramite la sua REST API.

Configurazione

Per prima cosa deve creare una **API Key** nel Suo account Kucoin e aggiungere i privilegi per il trading con i Futures.

Una volta completato, è possibile iniziare il trading di futures.

Prima, **impostare ApiKey, ApiSecret e Passphrase** nel Componente Client Kucoin; questi verranno utilizzati per firmare le richieste inviate al server Kucoin.

Inserisci un Ordine

Per inserire un nuovo ordine, chiamare il metodo **REST_API.PlaceOrder** del componente client Kucoin.

A seconda del tipo di ordine (market, limit...) l'API richiede più o meno campi.

Parametri

Pa-ram	ty-pe	Descrizione
clien-tOid	String	Id univoco dell'ordine creato dagli utenti per identificare i propri ordini, ad es. UUID. Sono consentiti solo numeri, caratteri, trattino basso (_) e separatore (-)
side	String	buy o sell
symbol	String	un codice contratto valido. es. XBTUSD
type	String	[<i>opzionale</i>] Sia limit che market
leverage	String	Leva finanziaria dell'ordine
nota	String	[<i>facoltativo</i>] nota per l'ordine, la lunghezza non può superare 100 caratteri utf8
stop	String	[<i>facoltativo</i>] Può essere down o up . Richiede che siano definiti stopPrice e stopPriceType
stopPriceType	String	[<i>facoltativo</i>] Può essere TP , IP o MP . Deve essere definito se viene specificato stop .
stopPrice	String	[<i>facoltativo</i>] Deve essere definito se è specificato stop .
reduceOnly	boolean	[<i>opzionale</i>] Un contrassegno per ridurre solo la dimensione della posizione. Impostato su false per impostazione predefinita. È necessario impostare la dimensione della posizione quando reduceOnly è true.
closeOrder	boolean	[<i>opzionale</i>] Un contrassegno per chiudere la posizione. Impostato su false per impostazione predefinita. Chiuderà tutte le posizioni quando closeOrder è true.
forceHold	boolean	[<i>opzionale</i>] Un contrassegno per trattenere forzatamente i fondi per un ordine, anche se si tratta di un ordine per ridurre la dimensione della posizione. Ciò aiuta l'ordine a rimanere nel book degli ordini e a non essere annullato quando la dimensione della posizione cambia. Impostato su false per impostazione predefinita.

Pa- ram	ty- pe	Descrizione

PARAMETRI ORDINE LIMITE

Param	type	Descrizione
price	String	Prezzo limite
dimensio- ne	Integer	Dimensione dell'ordine. Deve essere un numero positivo
timeInFor- ce	String	<i>[facoltativo]</i> GTC, IOC (predefinito è GTC), vedere Time In Force
postOnly	boo- lean	<i>[opzionale]</i> Flag post only, non valido quando timeInForce è IOC . Quando si sceglie postOnly, non è consentito scegliere hidden o iceberg.
nascosto	boo- lean	<i>[facoltativo]</i> Ordini non visualizzati nel libro degli ordini. Se si sceglie nascosto, non è consentito scegliere postOnly.
iceberg	boo- lean	<i>[opzionale]</i> Nel book degli ordini viene visualizzata solo la porzione visibile dell'ordine. Se si sceglie iceberg, non è consentito scegliere postOnly.
visibleSize	Integer	<i>[opzionale]</i> La dimensione visibile massima di un ordine iceberg

PARAMETRI ORDINE DI MERCATO

Param	tipo	Descrizione
size	Integer	<i>[opzionale]</i> quantità del contratto da acquistare o vendere

Quando si invia un ordine, ci sono 2 possibilità:

1. **Esito positivo:** la funzione PlaceOrder restituisce il messaggio inviato dal server Kucoin.
2. **Error:** l'eccezione viene restituita nell'evento OnKucoinHTTPException.

Place Market Order 1 XBTUSDM

```

TsgcWSAPI_Kucoin_Futures
oKucoin = new
TsgcWSAPI_Kucoin_Futures
();
oKucoin.Kucoin.ApiKey = "<api key>";
oKucoin.Kucoin.ApiSecret = "<api secret>";
oKucoin.Kucoin.Passphrase = "<passphrase>";
MessageBox.Show(oKucoin.REST_API.
PlaceMarketOrder
(

```

```
kosBuy, "XBTUSD", 1
));
```

Piazzare un ordine limite da 1 XBTUSD a 40000

```
TsgcWSAPI_Kucoin_Futures
oKucoin = new
TsgcWSAPI_Kucoin_Futures
();
oKucoin.Kucoin.ApiKey = "<api key>";
oKucoin.Kucoin.ApiSecret = "<api secret>";
oKucoin.Kucoin.Passphrase = "<passphrase>";
MessageBox.Show(oKucoin.REST_API.
PlaceLimitOrder
(
kosBuy, "XBTUSD", 1, 40000
));
```

Kucoin | Tempo per le richieste private Futures

Quando si effettua una richiesta privata a Kucoin, il messaggio viene firmato per aumentare la sicurezza delle richieste. Il messaggio prende l'ora locale e la invia all'interno del messaggio firmato; se l'ora locale presenta una differenza superiore a 5 secondi rispetto ai server Kucoin, la richiesta verrà rifiutata. Pertanto, è importante verificare che la propria ora locale sia sincronizzata; è possibile farlo utilizzando il metodo di sincronizzazione dell'ora del proprio sistema operativo.

È possibile verificare l'orario del server Kucoin chiamando il metodo **GetServerTime**, che restituirà l'orario del server Kucoin

API 3Commas

3Commas

API supportate

- **WebSockets API:** connessione a un server websocket pubblico che fornisce aggiornamenti in tempo reale sui dati di mercato.
- **REST API:** La REST API dispone di endpoint per la gestione dell'account e degli ordini nonché per i dati di mercato pubblici.

WebSockets API

Il feed WebSocket fornisce aggiornamenti in tempo reale dei dati di mercato per Trade e Deal

È possibile sottoscrivere ai seguenti **canali pubblici**:

Method	Argomen- ti	Description
Subscri- beSmart- Trades		
Subscri- beDeals		

Questi canali richiedono l'**autenticazione** rispetto ai server 3Commas. Quindi richiedi per prima cosa le Sue chiavi API nel Suo account 3Commas e poi imposti i valori nella proprietà ThreeComas del componente:

- ApiKey
- ApiSecret

Se la sottoscrizione ha esito positivo, verrà chiamato l'evento **OnThreeCommasConfirmSubscription**. In caso contrario, viene chiamato l'evento **OnThreeCommasRejectSubscription**; è possibile ottenere il motivo del rifiuto utilizzando il parametro **aRawMessage**.

REST API

Test di connettività

Method	Argomenti	Description
GetPing		
GetSer- verTime		Restituisce l'ora del server

Account

Method	Argomenti	Description

GetAccounts		Elenco degli exchange connessi dall'utente
GetMarketList		Elenco dei mercati supportati
GetMarketPairs	aMarketCode: codice del mercato	Tutte le coppie di mercato
GetCurrentRatesWithLeverageData	aMarketCode: codice del mercato aPair: nome della coppia	Tassi di cambio e limiti con dati sulla leva finanziaria
GetCurrentRates	aMarketCode: codice del mercato aPair: nome della coppia	Tariffe e limiti valutari
GetBalances	aAccountId: id dell'account	Bilanciamento del carico per l'exchange specificato
GetAccountTableData	aAccountId: id dell'account	Informazioni su tutti i saldi dell'utente sull'exchange specificato
GetAccountLeverage	aAccountId: id dell'account aPair: nome della coppia	Informazioni sulla leva dell'account
GetAccountInfo	aAccountId: id dell'account	Informazioni account singolo

Smart Trades

Method	Argomenti	Description
GetSmartTradeHistory		Recupera la cronologia dei trade
PlaceMarketOrder	aAccountId: id dell' account aOrderSide: acquisto o vendita aPair: nome della coppia aQuantity: quantità	Inserisce un ordine a mercato
PlaceLimitOrder	aAccountId: id dell'account aOrderSide: acquisto o vendita aPair: nome della coppia aQuantity: quantità aPrice: prezzo limite	Inserisce un ordine Limit
GetSmartTrade	aid: id della transazione	Ottieni uno Smart Trade tramite l'Id del Trade

Cancel-SmartTrade	ald: id della transazione	Annulla un Smart Trade tramite l'Id del Trade
CloseBy-MarketSmartTrade	ald: id della transazione	
Edi-tSmart-Trade	ald: id della transazione	Modifica un Smart Trade esistente
Force-StartSmartTrade	ald: id della transazione	Avvia forzatamente uno Smart Trade
Add-FundsSmartTrade	ald: id della transazione	Aggiungere fondi a uno Smart Trade
GetSmart-TradeTrades	ald: id della transazione	Ottieni i trade di uno Smart Trade

DCA Bot

Method	Argomenti	Description
CreateD-CABot		Crea un nuovo bot DCA
GetDCA-Bot	ald: id del bot	Ottieni un bot DCA per ID
GetDCA-Bots		Ottieni tutti i Bot DCA
EnableD-CABot	ald: id del bot	Abilitare un DCA Bot
Disabled-CABot	ald: id del bot	Disabilitare un Bot DCA
DeleteD-CABot	ald: id del bot	Eliminare un DCA Bot
CancelD-CABot	ald: id del bot	Annulla un bot DCA
GetDCA-BotStats		Ottiene le statistiche del DCA Bot
GetAvailableStrategyList		Ottieni l'elenco delle strategie disponibili
GetBlacklistPairs		Ottieni coppie nella blacklist
AddBlacklistPairs		Aggiungere coppie alla blacklist

Operazioni

Method	Argomenti	Description
GetDeals		Ottiene tutte le operazioni
GetDeal	ald: id del deal	Ottiene un deal per Id
Update-Deal	ald: id del deal	Aggiorna un'operazione
Cancel-Deal	ald: id del deal	Annulla un'operazione
CloseAt-Market-Deal	ald: id del deal	Chiude un'operazione al prezzo di mercato

Grid Bot

Method	Argomenti	Description
Create-GridBot		Crea un nuovo Grid Bot
GetGrid-Bot	ald: id del bot	Recupera un Grid Bot tramite Id
GetGrid-Bots		Recupera tutti i Grid Bot
Enable-GridBot	ald: id del bot	Abilita un Grid Bot
Disable-GridBot	ald: id del bot	Disabilita un Grid Bot
Delete-GridBot	ald: id del bot	Elimina un Grid Bot

Eventi

OnConnect

Quando una nuova connessione WebSocket viene aperta

OnDisconnect

Quando una connessione WebSocket viene chiusa

OnThreeCommasConnect

Quando il client riceve un messaggio Welcome dal server 3Commas, significa che la connessione è pronta.

OnThreeCommasConfirmSubscription

Conferma una precedente sottoscrizione inviata dal client.

OnThreeCommasRejectSubscription

Si è verificato un errore durante il tentativo di sottoscrivere un canale 3Commas

OnThreeCommasMessage

Qui il client riceve i dati inviati dal server relativi ai canali sottoscritti

OnThreeCommasPing

Ping inviato dal server al client.

OnThreeCommasHTTPException

Se si verifica un errore durante la chiamata ai metodi HTTP REST, verrà chiamato questo evento.

API OKX

OKX

API supportate

- **WebSockets API:** connette a un server websocket e fornisce aggiornamenti in tempo reale sui dati di mercato, modifiche dell'account e inserimento di ordini di trading.

Proprietà

I canali WebSocket sono divisi in due categorie: canali pubblici e canali privati.

- **Canali pubblici:** includono il canale ticker, il canale K-Line, il canale del prezzo limite, il canale del libro degli ordini e il canale del prezzo mark, ecc. -- non richiedono l'accesso.
- **Canali privati:** incluso il canale account, il canale ordini e il canale posizioni, ecc. -- richiedono il login.

È possibile configurare le seguenti proprietà nella proprietà OKX.

- **ApiKey:** è possibile richiedere una nuova chiave API nel proprio account OKX, basta copiare il valore in questa proprietà.
- **ApiSecret:** è il valore segreto dell'api.
- **Passphrase:** è la stringa personalizzata definita durante la creazione di una nuova chiave API.
- **IsDemo:** se abilitato, si conatterà all'account Demo OKX (disabilitato per impostazione predefinita).
- **IsPrivate:** se abilitato, sarà possibile connettersi a canali privati (disabilitato per impostazione predefinita).
- **IsBusiness:** se abilitato, si conatterà all'endpoint WebSocket Business (wss://ws.okx.com:8443/ws/v5/business). Utilizzare questa opzione per i canali candela, i canali di ordini algo e altri canali avanzati (disabilitato per impostazione predefinita).

Connessione

Quando il client si connette con successo ai server OKX, viene attivato l'evento **OnOKXConnect**. Se si verifica un errore durante il tentativo di connessione, l'evento **OnOKXError** verrà attivato con i dettagli dell'errore.

Dopo che l'evento **OnOKXConnect** viene attivato, è possibile iniziare a **inviare e ricevere messaggi** dai server OKX.

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_OKX oOKX = new TsgcWSAPI_OKX();
oOKX.Client = oClient;
oOKX.OKX.ApiKey = "alsdj23kandfnasbdfdkjhdsf";
oOKX.OKX.ApiSecret = "alidskjfk3jkadknfajndsjsfj23j";
oOKX.OKX.Passphrase = "secret_passphrase";
oClient.Active = true;
void OnOKXConnect(TsgcWSAPI_OKX Sender, string aMessage, string aCode, string aRawMessage)
{
    DoLog("#OKX Connected");
}
void OnOKXError(TsgcWSAPI_OKX Sender, string aCode, string aMessage, string aRawMessage)
{
    DoLog("#error: " + aMessage);
}
```

Canali pubblici

Il feed websocket fornisce aggiornamenti in tempo reale dei dati di mercato per ordini e trade. Il feed websocket ha alcuni canali pubblici come ticker, trade...

È possibile sottoscrivere ai seguenti **canali pubblici**:

Method	Description
SubscribeInstruments	L'elenco completo degli strumenti verrà inviato per la prima volta dopo la sottoscrizione. Successivamente, gli strumenti verranno inviati in caso di modifiche allo stato dello strumento (come la consegna di FUTURES, l'esercizio di OPTION, la quotazione di nuovi contratti/coppie di trading, la sospensione delle negoziazioni, ecc.).
SubscribeTicker	Recupera l'ultimo prezzo negoziato, il prezzo bid, il prezzo ask e il volume di scambi delle ultime 24 ore degli strumenti. I dati vengono aggiornati ogni 100 ms.
SubscribeOpenInterest	Recupera l'open interest. I dati verranno inviati ogni 3 secondi.
SubscribeCandlestick	Recupera i dati delle candele di uno strumento. La frequenza di push è l'intervallo più veloce, con push dei dati ogni 500ms.
SubscribeTrades	Recupera i dati dei trade recenti. I dati vengono inviati ogni volta che avviene un trade.
SubscribeEstimatedPrices	Recupera il prezzo di consegna/esercizio stimato dei contratti FUTURES e delle OPTION. Solo il prezzo stimato di consegna/esercizio verrà inviato tramite push un'ora prima della consegna/esercizio, e verrà inviato tramite push in caso di qualsiasi variazione di prezzo.
SubscribeMarkPrice	Recupera il prezzo mark. I dati vengono inviati ogni 200 ms quando il prezzo mark cambia, e ogni 10 secondi quando il prezzo mark non cambia.
SubscribeMarkPrice-Candlestick	Recupera i dati candlestick del prezzo mark. I dati verranno inviati ogni 500 ms.
SubscribePriceLimit	Recuperare il prezzo massimo di acquisto e il prezzo minimo di vendita dello strumento. I dati vengono inviati ogni 5 secondi quando ci sono variazioni nei limiti, e non vengono inviati in assenza di variazioni.
SubscribeOrderBook	Recupera i dati del book degli ordini. Utilizzare books per 400 livelli di profondità, book5 per 5 livelli di profondità, bbo-tbt tick-by-tick con 1 livello di profondità, books50-l2-tbt tick-by-tick con 50 livelli di profondità, e books-l2-tbt per tick-by-tick con 400 livelli di profondità. • books: 400 livelli di profondità verranno inviati nello snapshot completo iniziale. I dati incrementali verranno inviati ogni 100 ms quando c'è un cambiamento nel libro degli ordini. • books5: 5 livelli di profondità verranno inviati ogni volta. I dati verranno inviati ogni 100 ms quando c'è un cambiamento nel libro degli ordini. • bbo-tbt: 1 livello di profondità verrà inviato ogni volta. I dati verranno inviati ogni 10 ms quando c'è un cambiamento nel libro degli ordini. • books-l2-tbt: 400 livelli di profondità verranno inviati nello snapshot completo iniziale. I dati incrementali verranno inviati ogni 10 ms quando c'è un cambiamento nel libro degli ordini. • books50-l2-tbt: 50 livelli di profondità verranno inviati nello snapshot completo iniziale. I dati incrementali verranno inviati ogni 10 ms quando c'è un cambiamento nel libro degli ordini. Se asks o bids è un array vuoto, significa che ci sono cambiamenti a 400 livelli di profondità, invece che a 50. Se si mantengono i dati del libro degli ordini localmente, ignorare asks e bids vuoti.
SubscribeOptionSummary	Recupera informazioni dettagliate sui prezzi di tutti i contratti OPTION. I dati verranno inviati tutti in una volta.
SubscribeFundingRate	Recupera il tasso di finanziamento. I dati verranno inviati ogni 30-90 secondi.
SubscribeIndexCandlestick	Recupera i dati candlestick dell'indice. I dati verranno inviati ogni 500 ms.
SubscribeIndexTicker	Recupera i dati dei ticker dell'indice
SubscribeStatus	Ottieni lo stato della manutenzione del sistema e ricevi notifiche push quando lo stato cambia. Prima sottoscrizione: "Invia gli ultimi dati modificati"; ogni volta che si verifica un cambiamento di stato, invia il contenuto modificato

SubscribePublicStructureBlockTrades	I dati vengono inviati ogni volta che si verifica un block trade.
SubscribeBlockTickers	Recupera l'ultimo volume di block trading nelle ultime 24 ore. I dati verranno inviati quando attivati da un evento di esecuzione di transazioni. Inoltre, verranno inviati anche con un intervallo di 5 minuti in base alla granularità della sottoscrizione.
SubscribeAllTrades	Recupera tutti i dati delle operazioni. I dati verranno inviati ogni volta che si verifica un'operazione.
SubscribeLiquidationOrders	Recupera gli ordini di liquidazione. I dati verranno trasmessi quando è presente un ordine di liquidazione.
SubscribeADLWarning	Recupera i dati di avviso ADL. I dati verranno inviati quando il rischio di auto-deleveraging aumenta.
SubscribeEconomicCalendar	Recupera gli eventi del calendario economico. I dati verranno inviati quando vi sono aggiornamenti agli eventi economici.
SubscribePublicBlockTrades	Recuperare le negoziazioni a blocchi pubbliche. I dati verranno inviati ogni volta che si verifica una negoziazione a blocchi.
SubscribeOptionTrades	Recupera i dati delle negoziazioni su opzioni. I dati verranno inviati ogni volta che si verifica una negoziazione su opzioni.
SubscribeCallAuctionDetails	Recupera i dettagli dell'asta a chiamata. I dati verranno inviati quando ci sono aggiornamenti alle informazioni sull'asta a chiamata.

```

TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_OKX oOKX = new TsgcWSAPI_OKX();
oOKX.Client = oClient;
oOKX.OKX.ApiKey = "alsdk23kandfnasbdfdkjhdsf";
oOKX.OKX.ApiSecret = "aldskjfk3jkadknfajndsifj23j";
oOKX.OKX.Passphrase = "secret_passphrase";
oClient.Active = true;
void OnOKXConnect(TsgcWSAPI_OKX Sender, string aMessage, string aCode, string aRawMessage)
{
    oOKX.SubscribeInstruments(okxitFutures);
}

```

Canali privati

Inclusi il canale account, il canale ordini e il canale posizioni, ecc. -- richiedono l'accesso.

È possibile iscriversi ai seguenti **canali Privati**:

Method	Description
SubscribeAccount	Recupera le informazioni dell'account. I dati verranno inviati quando attivati da eventi come l'inserimento di un ordine, l'annullamento di un ordine, l'esecuzione di una transazione, ecc. Verranno inoltre inviati a intervalli regolari in base alla granularità della sottoscrizione.
SubscribePositions	Recupera le informazioni sulla posizione. Lo snapshot iniziale verrà inviato in base alla granularità della sottoscrizione. I dati verranno inviati quando attivati da eventi quali l'inserimento/annullamento di ordini, e verranno inviati anche a intervalli regolari in base alla granularità della sottoscrizione.
SubscribeBalanceAndPosition	Recupera le informazioni sul saldo e sulla posizione dell'account. I dati vengono inviati quando attivati da eventi quali ordini eseguiti, trasferimenti di fondi.
SubscribeOrders	Recupera le informazioni sull'ordine. I dati non verranno inviati alla prima sottoscrizione. I dati verranno inviati solo quando attivati da eventi come il piazzamento/annullamento di un ordine.

SubscribeOrdersAlgo	Recupera gli ordini algo (include ordini trigger, ordini oco, ordini condizionali). I dati non vengono inviati al primo abbonamento. I dati vengono inviati solo quando attivati da eventi come il piazzamento/cancellazione di ordini.
SubscribeAdvanceAlgo	Recupera gli ordini algo avanzati (inclusi ordine Iceberg, ordine TWAP, ordine Trailing). I dati verranno inviati al primo abbonamento. I dati verranno inviati quando attivati da eventi quali l'inserimento/annullamento di un ordine.
SubscribePositionRisk	Questo canale push viene utilizzato solo come avviso di rischio e non è raccomandato come giudizio di rischio per il trading strategico Nel caso in cui il mercato non si muova violentemente, potrebbe esserci la possibilità che la posizione sia stata liquidata nello stesso momento in cui viene inviato questo messaggio.
SubscribeAccount-Greeks	Recupera le informazioni greche dell'account. I dati verranno inviati quando attivati da eventi quali aumento/diminuzione delle posizioni o saldo in contanti nell'account, e verranno inviati anche a intervalli regolari in base alla granularità della sottoscrizione.
SubscribeRfq	Recupera gli Rfq.
SubscribeQuotes	Recuperare le quotazioni.
SubscribePrivate-StructureBlockTrades	Recupero delle operazioni di struttura a blocchi.
SubscribeSpotGridAlgoOrders	Recupera gli ordini di algoritmi spot grid. I dati vengono inviati alla prima sottoscrizione. I dati vengono inviati quando attivati da eventi come il piazzamento/annullamento di un ordine.
SubscribeContactGridAlgoOrders	Recupera gli ordini algo a griglia per contratti. I dati verranno inviati alla prima iscrizione. I dati verranno inviati quando attivati da eventi come il piazzamento o l'annullamento di un ordine.
SubscribeGridPositions	Recupera le posizioni della griglia. I dati vengono inviati al primo abbonamento. I dati vengono inviati quando attivati da eventi come il piazzamento/cancellazione di ordini.
SubscribeGridSubOrders	Recupera gli ordini secondari della griglia. I dati vengono inviati al primo abbonamento. I dati vengono inviati quando attivati da eventi come il piazzamento di ordini.
SubscribeFills	Recupera i dati degli ordini eseguiti. I dati verranno inviati quando un ordine viene eseguito.
SubscribeDepositInfo	Recupera le informazioni sui depositi. I dati verranno inviati quando si verifica un aggiornamento dello stato di un deposito.
SubscribeWithdrawalInfo	Recupera le informazioni sui prelievi. I dati verranno inviati quando vi è un aggiornamento dello stato di un prelievo.

```

TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_OKX oOKX = new TsgcWSAPI_OKX();
oOKX.Client = oClient;
oOKX.OKX.ApiKey = "alsdk23kandfnasbdfdkjhdsf";
oOKX.OKX.ApiSecret = "aldskjfk3jkdaknfajndsxfj23j";
oOKX.OKX.Passphrase = "secret_passphrase";
oClient.Active = true;
void OnOKXConnect(TsgcWSAPI_OKX Sender, string aMessage, string aCode, string aRawMessage)
{
    oOKX.SubscribeOrders(okxifutures, "BTC-USD", "BTC-USD-200329");
}

```

Trading

Il WebSocket Trade richiede l'**autenticazione**.

È possibile inserire un ordine solo se si dispone di fondi sufficienti. Di seguito una tabella con i parametri della richiesta:

Para-metro	Ti-po	Obbli-gatorio	Descrizione
id	String	Sì	Identificatore univoco del messaggio Fornito dal client. Verrà restituito nel messaggio di risposta per identificare la richiesta corrispondente. Una combinazione di caratteri alfanumerici con distinzione tra maiuscole e minuscole, tutti numeri o tutte lettere, fino a 32 caratteri.
> instId	String	Sì	ID strumento, ad es. <code>BTC-USD-190927-5000-C</code>
> tdMode	String	Sì	Modalità di trading Modalità margine <code>isolated</code> <code>cross</code> Modalità senza margine <code>cash</code>
> ccy	String	No	Valuta del margine Applicabile solo agli ordini <code>cross</code> <code>MARGIN</code> in <code>Single-currency margin</code> .
> clOrdId	String	No	ID dell'ordine fornito dal client Una combinazione di caratteri alfanumerici con distinzione tra maiuscole e minuscole, tutti numeri o tutte lettere, fino a 32 caratteri.
> tag	String	No	Tag dell'ordine Una combinazione di caratteri alfanumerici sensibili alle maiuscole, tutti numeri o tutte lettere fino a 16 caratteri.
> lato	String	Sì	Lato dell'ordine, <code>buy</code> <code>sell</code>
> posSide	String	Opzionale	Lato della posizione Il valore predefinito è <code>net</code> nella modalità <code>net</code> È richiesto nella modalità <code>long/short</code> e può essere solo <code>long</code> o <code>short</code> . Applicabile solo a <code>FUTURES/SWAP</code> .
> ordType	String	Sì	Tipo di ordine <code>market</code> : ordine di mercato <code>limit</code> : ordine limite <code>post_only</code> : ordine Post-only <code>fok</code> : ordine Fill-or-kill <code>ioc</code> : ordine Immediate-or-cancel <code>optimal_limit_ioc</code> :Ordine di mercato con ordine immediate-or-cancel
> sz	String	Sì	Quantità da acquistare o vendere.
> px	String	Opzionale	Price Applicabile solo agli ordini <code>limit</code> , <code>post_only</code> , <code>fok</code> , <code>ioc</code> .
> reduceOnly	Boolean	No	Se ridurre solo la posizione o meno, <code>true</code> <code>false</code> , il valore predefinito è <code>false</code> Applicabile solo agli ordini <code>MARGIN</code> , e agli ordini <code>FUTURES/SWAP</code> in modalità <code>net</code> Applicabile solo al margine <code>Single-currency margin</code> e <code>Multi-currency margin</code>
> tgtCcy	String	No	Tipo di quantità <code>base_ccy</code> : Valuta base , <code>quote_ccy</code> : Valuta di quotazione Applicabile solo a <code>SPOT</code> negoziato con ordine Market L'impostazione predefinita è <code>quote_ccy</code> per l'acquisto, <code>base_ccy</code> per la vendita
> banAmend	Boolean	No	Se impedire la modifica degli ordini spot o meno, true o false, il valore predefinito è false. Il piazzamento degli ordini non riuscirà se il saldo non è sufficiente quando <code>banAmend</code> è true. Applicabile solo agli ordini di mercato SPOT

Esempio di inserimento ordine

È possibile inserire un ordine solo se si dispone di fondi sufficienti.

```
// Place Market Order
TsgcWSAPI_OKX1.PlaceMarketOrder(okxosBuy, "ETH-BTC", 1);
// Place Limit Order
TsgcWSAPI_OKX1.PlaceLimitOrder(okxosBuy, "ETH-BTC", 1, 0.25);
```

Esempio di cancellazione ordine

Annulla un ordine incompleto

```
TsgcWSAPI_OKX1.CancelOrder("ETH-BTC", "457589362405027840");
```

Modifica ordine

Modifica un ordine incompleto.

```
TsgcWSAPI_OKX1.AmendOrder("ETH-BTC", "457589362405027840", "", 2);
```

Operazioni di Trading in Batch

L'API WebSocket Trade supporta anche operazioni batch per il piazzamento, la cancellazione e la modifica di più ordini contemporaneamente. Queste operazioni richiedono l'**Autenticazione**.

Method	Description
BatchPlaceOrders	Invia più ordini in una singola richiesta. È possibile inviare un massimo di 20 ordini alla volta.
BatchCancelOrders	Annulla più ordini in una singola richiesta. È possibile annullare al massimo 20 ordini alla volta.
BatchAmendOrders	Modifica più ordini incompleti in una singola richiesta. È possibile modificare un massimo di 20 ordini alla volta.
MassCancelOrders	Annulla in massa tutti gli ordini in sospeso per un tipo di strumento specifico.

API XTB

XTB

API supportate

- **API WebSockets:** si connette a un server websocket e fornisce aggiornamenti dei dati di mercato in tempo reale, variazioni dell'account e inserimento di ordini di trading.

Proprietà

Il protocollo WebSocket consente 2 tipi di richieste: **Comandi in streaming** (ricezione di aggiornamenti in tempo reale) e **Recupero dati di trading** (invio di una richiesta al server per recuperare alcune informazioni).

È possibile configurare le seguenti proprietà nella proprietà XTB.

- **Utente:** il nome utente che identifica la connessione.
- **Password:** è il valore segreto dell'utente.
- **Demo:** se abilitato, si conatterà all'account Demo XTB (disabilitato per impostazione predefinita).

Connessione

Quando il client si connette con successo ai server XTB, viene attivato l'evento **OnXTBConnect**. Se si verifica un errore durante il tentativo di connessione, verrà attivato l'evento **OnXTBError** con i dettagli dell'errore.

Dopo che l'evento **OnXTBConnect** viene generato, è possibile iniziare a **inviare e ricevere messaggi** dai server XTB.

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_XTB oXTB = new TsgcWSAPI_XTB();
oXTB.Client = oClient;
oXTB.XTB.User = "user_0001";
oXTB.XTB.Password = "secret_0001";
oClient.Active = true;
void OnXTBConnect(TsgcWSAPI_XTB Sender, string aStreamSessionId)
{
    DoLog("#XTB Connected");
}
void OnXTBError(TsgcWSAPI_XTB Sender, string aCode, string aDescription, string aRawMessage)
{
    DoLog("#error: " + aDescription);
}
```

Comandi di connessione

Method	Description
Login	Per eseguire qualsiasi azione, l'applicazione client deve effettuare il processo di login. Nessuna funzionalità è disponibile prima di un corretto processo di login. Il metodo di login viene chiamato automaticamente dopo che il client si connette al server WebSocket e i valori User/Password sono impostati.
Logout	

Comandi di streaming

È possibile sottoscrivere ai seguenti canali:

Method	Description
SubscribeBalance	Consente di ottenere i valori effettivi degli indicatori dell'account in tempo reale, non appena disponibili nel sistema.
SubscribeCandles	Sottoscrive e annulla la sottoscrizione ai grafici candela dell'API. L'intervallo di ogni candela è di 1 minuto. Una nuova candela arriva ogni minuto.
SubscribeKeepAlive	Sottoscrive e annulla la sottoscrizione ai messaggi 'keep alive'. Un nuovo messaggio 'keep alive' viene inviato dall'API ogni 3 secondi.
SubscribeNews	Sottoscrive e annulla la sottoscrizione alle notizie.
SubscribeProfits	Sottoscrive e annulla la sottoscrizione ai profitti.
SubscribeTickPrices	Stabilisce una sottoscrizione per le quotazioni e consente di ottenere le informazioni pertinenti in tempo reale, non appena disponibili nel sistema. Il comando <code>getTickPrices</code> può essere richiamato più volte per lo stesso simbolo, ma verrà creata una sola sottoscrizione per un determinato simbolo. Si tenga presente che quando sono disponibili più record, l'ordine in cui vengono ricevuti non è garantito.
SubscribeTrades	Stabilisce una sottoscrizione per i dati sullo stato degli scambi dell'utente e consente di ottenere le informazioni pertinenti in tempo reale, non appena disponibili nel sistema. Si prega di notare che quando sono disponibili più record, l'ordine in cui vengono ricevuti non è garantito.
SubscribeTradeStatus	Consente di ottenere lo stato delle richieste di trade inviate in tempo reale, non appena disponibile nel sistema. Si tenga presente che quando sono disponibili più record, l'ordine in cui vengono ricevuti non è garantito.
SubscribePing	Chiamare regolarmente questa funzione è sufficiente per aggiornare lo stato interno di tutti i componenti nel sistema. La connessione in streaming, quando nessun comando viene inviato dal client nella sessione, genera traffico di rete solo in una direzione. Si raccomanda che qualsiasi applicazione che non esegue altri comandi chiami questo comando almeno una volta ogni 10 minuti.

Recupero dei Dati di Trading

Può inviare le seguenti Request:

Method	Description
GetAllSymbols	Restituisce un array di tutti i simboli disponibili per l'utente.
GetCalendar	Restituisce il calendario con gli eventi di mercato.
GetChartLastRequest	Si noti che questa funzione può di solito essere sostituita dal suo equivalente in streaming <code>getCandles</code> , che è il metodo preferito per recuperare i dati delle candele correnti. Restituisce le informazioni del grafico, dalla data di inizio fino all'ora corrente. Se il periodo scelto di <code>CHART_LAST_INFO_RECORD</code> è superiore a 1 minuto, l'ultima candela restituita dall'API può cambiare fino alla fine del periodo (la candela viene aggiornata automaticamente ogni minuto).

GetChartRangeRequest	Si noti che questa funzione può solitamente essere sostituita dal suo equivalente in streaming getCandles, che è il modo preferito per recuperare i dati delle candele correnti. Restituisce le informazioni del grafico con i dati compresi tra le date di inizio e fine specificate.
GetCommissionDef	Restituisce il calcolo della commissione e del tasso di cambio. Il valore è calcolato come valore atteso e pertanto potrebbe non essere perfettamente accurato.
GetCurrentUserData	Restituisce informazioni sulla valuta e la leva dell'account.
GetIbsHistory	Restituisce i dati IB dell'intervallo di tempo specificato.
GetMarginLevel	Si noti che questa funzione può essere solitamente sostituita dal suo equivalente in streaming getBalance, che è il metodo preferito per recuperare gli indicatori dell'account. Restituisce vari indicatori dell'account
GetMarginTrade	Restituisce il margine atteso per un determinato strumento e volume. Il valore è calcolato come valore di margine atteso e pertanto potrebbe non essere perfettamente accurato.
GetNews	Si noti che questa funzione può solitamente essere sostituita dall'equivalente in streaming getNews, che è il metodo preferito per recuperare dati sulle notizie. Restituisce le notizie dal server di trading che sono state inviate nell'arco di tempo specificato.
GetProfitCalculation	Calcola il profitto stimato per i dati di un'operazione. Da utilizzare solo per applicazioni di tipo calcolatrice. Il profitto per le transazioni aperte deve essere ottenuto dal server, per via della maggiore precisione del calcolo lato server.
GetServerTime	Restituisce l'orario corrente sul server di trading
GetStepRules	Restituisce un elenco di regole di step per le DMA
GetSymbol	Restituisce informazioni sul simbolo disponibile per l'utente
GetTickPrices	Si noti che questa funzione può di solito essere sostituita dal suo equivalente in streaming getTickPrices, che è il modo preferito per recuperare i dati di tick. Restituisce un array di quotazioni correnti per i simboli indicati; vengono restituite solo le quotazioni modificate rispetto al timestamp indicato. Il nuovo timestamp ottenuto dall'output verrà utilizzato come argomento della chiamata successiva a questo comando.
GetTradeRecords	Restituisce un array di trade elencati nell'argomento orders
GetTrades	Si noti che questa funzione può solitamente essere sostituita dal suo equivalente in streaming getTrades, che è il metodo preferito per recuperare i dati sulle transazioni. Restituisce un array delle transazioni dell'utente.
GetTradesHistory	Si noti che questa funzione può solitamente essere sostituita dal suo equivalente in streaming getTrades, che è il metodo preferito per recuperare i dati delle transazioni. Restituisce un array delle transazioni dell'utente che sono state chiuse nel periodo di tempo specificato.
GetTradingHours	Restituisce quotazioni e orari di trading.
GetVersion	Restituisce la versione corrente dell'API.
Ping	Chiamare regolarmente questa funzione è sufficiente per aggiornare lo stato interno di tutti i componenti del sistema. Si raccomanda che qualsiasi applicazione che non esegua altri comandi chiami questo comando almeno una volta ogni 10 minuti. Si noti che la controparte in streaming di questa funzione è una combinazione di ping e getKeepAlive
TradeTransaction	Avvia la transazione commerciale. tradeTransaction invia le informazioni principali della transazione al server.

TradeTransactionStatus

Si noti che questa funzione può solitamente essere sostituita dal suo equivalente in streaming `getTradeStatus`, che è il metodo preferito per recuperare i dati sullo stato delle transazioni. Restituisce lo stato corrente della transazione. In qualsiasi momento durante l'elaborazione della transazione, il client può verificare lo stato della transazione lato server. A tal fine, il client deve fornire l'ordine univoco ottenuto dall'invocazione di `tradeTransaction`.

API Bybit

Bybit

API supportate

- **WebSocket API:** connette a un server websocket e fornisce aggiornamenti in tempo reale dei dati di mercato, variazioni dell'account e altro.
- **REST API:** invia richieste HTTP per ottenere dati di mercato, piazzare ordini, dati dell'account...

Attualmente, la versione dell'API supportata è V5. L'API V5 introduce uniformità ed efficienza nelle linee di prodotto di Bybit, unificando Spot, Derivati e Opzioni in un unico insieme di specifiche.

Versione OpenAPI	Tipo di conto	Linear			Inverso		Spot	Opzioni
		USDT Perpetual	USDC Perpetual	USDC Futures	Perpetual	Futures		
V5	Account di trading unificato	✓	✓	✓	vedi nota		✓	✓
	Account classico	✓			✓	✓	✓	
V3	Account di trading unificato	✓	✓					✓
	Account classico	✓			✓	✓	✓	

*Nota: l'account Unified supporta il trading inverso. Tuttavia, il margine utilizzato proviene dal wallet dei derivati inversi anziché dal wallet unificato.

Proprietà

È possibile configurare le seguenti proprietà nella proprietà Bybit.

- **ApiKey:** è possibile richiedere una nuova api key nel proprio account Bybit, basta copiare il valore in questa proprietà. Se APIKey è impostata, il client si conatterà al server WebSocket privato. Se è vuota, si conatterà al server WebSocket pubblico.
- **ApiSecret:** è il valore segreto dell'api.
- **SignatureExpires:** numero di secondi dopo i quali la firma scade (per impostazione predefinita 10 secondi).
- **TestNet:** se abilitato, si conatterà all'account demo Bybit TestNet (disabilitato per impostazione predefinita).

Connessione

Quando il client si connette con successo ai server Bybit, viene generato l'evento **OnConnect**. Dopo la generazione dell'evento **OnConnect**, è possibile iniziare a **inviare e ricevere messaggi** da/verso i server Bybit. Se ci si connette al canale websocket privato, è necessario attendere che venga generato l'evento **OnBybitAuthentication** e verificare che il parametro **success** sia **true**, prima di sottoscrivere qualsiasi canale.

Il client supporta diverse API; utilizzare la proprietà **BybitClient** per impostare quale API si desidera usare:

- **bybSpot**
- **bybInverse**
- **bybLinear**
- **bybPerpetual**

Di seguito è riportato un esempio di connessione all'API privata WebSocket Spot.

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bybit oBybit = new TsgcWSAPI_Bybit();
oBybit.Client = oClient;
oBybit.Bybit.ApiKey = "alsdk23kandfnasbdfdkjhdsf";
oBybit.Bybit.ApiSecret = "aldskjfk3jkadknfajndsffj23j";
oBybit.BybitClient = bybSpot;
oClient.Active = true;
void OnBybitConnect(TsgcWSConnection Connection)
{
    DoLog("#Bybit Connected");
}
```

Dopo una connessione riuscita al Server WebSocket Spot, è possibile iniziare a sottoscrivere i canali WebSocket; accedere alla proprietà **SPOT** e quindi chiamare uno qualsiasi dei metodi **subscribe/unsubscribe** disponibili.

Eventi

Il client bybit implementa i seguenti eventi per controllare il flusso della connessione e ottenere i dati inviati dal server WebSocket:

- **OnConnect**: attivato quando il client websocket si connette al server WebSocket.
- **OnDisconnect**: attivato quando il client websocket si disconnette dal WebSocket Server.
- **OnBybitAuthentication**: viene attivato quando il client si autentica presso il server WebSocket privato di Bybit.
- **OnBybitSubscribe**: quando il client si sottoscrive a un canale websocket.
- **OnBybitUnSubscribe**: generato quando il client annulla la sottoscrizione a un canale websocket.
- **OnBybitData**: quando il client riceve dati dal server.
- **OnBybitError**: quando si verifica un errore durante la connessione websocket bybit.
- **OnBybitHTTPException**: quando si verifica un errore durante la richiesta REST.

WebSocket API

Il feed websocket fornisce aggiornamenti in tempo reale dei dati di mercato per ordini e trade. Il feed websocket ha alcuni canali pubblici come ticker, trade...

Può sottoscrivere i seguenti **canali**:

Method	Pubblico o Privato	Description
SubscribeOrderBook	Public	Sottoscrivere il flusso dell'order book. Supporta profondità diverse.
SubscribeTrade	Public	Sottoscrivi allo stream degli scambi recenti.
SubscribeTicker	Public	Effettua la sottoscrizione allo stream del ticker.

SubscribeKLine	Public	Sottoscriviti allo stream klines.
SubscribeLiquidation	Public	Iscrizione allo stream di liquidazione
SubscribeLT_KLine	Public	Si iscrive allo stream kline dei leveraged token.
SubscribeLT_Ticker	Public	Iscrivi al flusso del ticker del token con leva finanziaria.
SubscribeLT_Nav	Public	Iscrivi al flusso del ticker del token con leva finanziaria.
SubscribePosition	Privato	Si iscrive allo stream nav dei leveraged token.
SubscribeExecution	Privato	Subscribe
SubscribeOrder	Privato	Subscribe
SubscribeWallet	Privato	Subscribe
SubscribeGreek	Privato	Subscribe
SubscribeDcp	Privato	Subscribe
SubscribeInsurance	Public	Sottoscrive il flusso del fondo assicurativo.
SubscribeOrderPriceLimit	Public	Iscriviti allo stream del limite di prezzo degli ordini.
SubscribeADLAlert	Public	Sottoscrivere lo stream di alert di auto-deleveraging.
SubscribeFastExecution	Privato	Abbonati allo stream di esecuzione rapida.

Di seguito è riportato un esempio di sottoscrizione ai canali WebSocket privati dopo un'autenticazione riuscita.

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bybit oBybit = new TsgcWSAPI_Bybit();
oBybit.Client = oClient;
oBybit.Bybit.ApiKey = "alsdj23kandfnasbdfdkjhsdf";
oBybit.Bybit.ApiSecret = "aldskjfk3jkadknfajnds23j";
oBybit.BybitClient = bybSpot;
oClient.Active = true;
void OnBybitAuthentication(TObject Sender, bool aSuccess, const string aError, const string aRawMessage)
{
    if (aSuccess == true)
    {
        oClient.SubscribeOrderBook("BTCUSD");
        oClient.SubscribeTrade("BTCUSD");
    }
}
```

REST API

La REST API dispone di un elenco di metodi pubblici e privati per richiedere dati da: mercati, account privato e wallet. Di seguito è riportato un elenco dei metodi disponibili.

Method	Public / Private
GetServerTime	Public
GetKLine	Public
GetMarkPriceKLine	Public
GetIndexPriceKLine	Public
GetPremiumIndexPriceKLine	Public
GetInstrumentsInfo	Public
GetOrderBook	Public
GetTickers	Public
GetFundingRateHistory	Public
GetPublicRecentTradingHistory	Public
GetOpenInterest	Public
GetHistoricalVolatility	Public
GetInsurance	Public
GetRiskLimit	Public
GetDeliveryPrice	Public

GetLongShortRatio	Public
PlaceOrder	Privato
PlaceMarketOrder	Privato
PlaceLimitOrder	Privato
AmendOrder	Privato
CancelOrder	Privato
GetOpenOrders	Privato
CancelAllOrders	Privato
GetOrderHistory	Privato
GetPositionInfo	Privato
SetLeverage	Privato
SwitchCrossIsolatedMargin	Privato
SetTPSLMode	Privato
SwitchPositionMode	Privato
SetRiskLimit	Privato
SetTradingStop	Privato
SetAutoAddMargin	Privato
AddOrReduceMargin	Privato
GetExecution	Privato
GetClosedPNL	Privato
ConfirmNewRiskLimit	Privato
GetWalletBalance	Privato
GetAccountInfo	Privato
GetTransactionLog	Privato
BatchPlaceOrder	Privato
BatchAmendOrder	Privato
BatchCancelOrder	Privato
SetDCP	Privato
GetFeeRate	Privato
GetCollateralInfo	Privato
SetMarginMode	Privato
GetBorrowHistory	Privato
GetCoinGreeks	Privato
GetCoinInfo	Privato
GetAllCoinsBalance	Privato
CreateInternalTransfer	Privato
GetInternalTransferList	Privato
GetDepositRecords	Privato
GetDepositAddress	Privato
CreateWithdrawal	Privato
CancelWithdrawal	Privato
GetWithdrawalRecords	Privato

Di seguito è riportato un esempio di recupero degli ordini aperti.

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bybit oBybit = new TsgcWSAPI_Bybit();
oBybit.Client = oClient;
oBybit.Bybit.ApiKey = "alsdk23kandfnasbdfdkjhdsf";
oBybit.Bybit.ApiSecret = "aldskjfk3jkadknfajndsifj23j";
oBybit.BybitClient = bybSpot;
```

```
oBybit.REST_API.GetAccountInfo();
```

API Forex

[Forex.com](#) (StoneX / CityIndex)

Forex.com è una piattaforma di trading al dettaglio gestita da StoneX. Il componente racchiude la Trading REST API e il servizio di streaming basato su Lightstreamer utilizzato da Forex.com, StoneX e CityIndex (tutte e tre condividono la stessa piattaforma). Sono supportate le seguenti API:

1. **API in streaming:** Lightstreamer TLCP 2.5 su WebSocket. I tick di prezzo, gli aggiornamenti degli ordini, gli aggiornamenti delle posizioni, il margine del conto e le quotazioni vengono inviati in tempo reale dopo un accesso REST riuscito.
2. **REST API:** login di sessione (UserName + Password + AppKey), gestione di ordini trade e stop-limit, query sull'account e dati di mercato. Utilizza HTTPS.

Componenti

- **TsgcWSAPI_Forex:** client principale. Contiene le **Credentials**, espone **REST_API**, il client di streaming interno **Lightstreamer** e gli eventi **OnForex***.
- **TsgcHTTP_API_Forex:** componente REST, accessibile tramite la proprietà **REST_API** del client principale.
- **TsgcWSPClient_Lightstreamer:** client interno del sottoprotocollo TLCP 2.5 utilizzato per le sottoscrizioni in streaming (PRICES / ORDERS / QUOTES / CLIENTACCOUNTMARGIN / TRADEMARGIN).

Proprietà pubblicate

- **Credentials:** gruppo di campi necessari per accedere alla REST API.
 - **UserName:** nome utente del Suo account Forex.com / StoneX / CityIndex.
 - **Password:** password dell'account.
 - **AppKey:** chiave applicazione rilasciata dalla piattaforma.
 - **AppVersion:** stringa opzionale della versione del client.
 - **AppComments:** testo libero registrato al login.
 - **StreamingHost:** host Lightstreamer (predefinito all'host di streaming pubblico).
- **AutoReconnect:** se abilitato, invia al WatchDog del client WebSocket sottostante in modo che la sessione di streaming si riconnetta automaticamente dopo un'interruzione di rete (per impostazione predefinita true).
- **REST_API:** riferimento al componente **TsgcHTTP_API_Forex**.
- **Lightstreamer:** riferimento al client interno del sottoprotocollo **TsgcWSPClient_Lightstreamer** (AdapterSet è fissato su STREAMINGALL).
- **SessionToken:** sola lettura. Token di sessione restituito dalla chiamata REST LogOn; utilizzato come *LS_password* per Lightstreamer.
- **ClientAccountId / TradingAccountId:** di sola lettura. Popolati dopo un LogOn riuscito.

Eventi pubblicati

OnForexConnect

Attivato una volta che la sessione di streaming è stata creata e le sottoscrizioni possono iniziare.

OnForexDisconnect

Viene attivato al termine della sessione di streaming (chiusura del server, interruzione di rete o Disconnect esplicito).

OnForexException

Generato in caso di errori REST (inclusi errori di LogOn) e in caso di guasti della sessione di streaming.

OnForexPriceTick

Generato ad ogni aggiornamento PRICES: Bid, Offer, Price, Change, Direction, AuditId, TickDate.

OnForexOrderUpdate

Generato per ogni aggiornamento ORDERS (nuovo, eseguito, annullato, rifiutato).

OnForexPositionUpdate

Generato per ogni aggiornamento di posizione aperta sul TradingAccount attivo.

OnForexAccountMargin

Generato per ogni aggiornamento CLIENTACCOUNTMARGIN / TRADEMARGIN (equity, margin, free margin, currency).

OnForexQuote

Generato per ogni aggiornamento QUOTES (flusso di lavoro richiesta-di-quotazione).

OnForexSubscribe

Generato ogni volta che il server conferma una sottoscrizione Lightstreamer (SUBOK / SUBCMD).
 Parametri: *aAdapter* (PRICES, ORDERS, QUOTES, CLIENTACCOUNTMARGIN o TRADEMARGIN),
aGroup (gruppo di elementi, ad es. *ID.<ClientAccountId>*), *altems*, *aFields*.

OnForexUnsubscribe

Si attiva quando una sottoscrizione termina — tramite **UnwatchMarket**, alla disconnessione dallo streaming, o quando il server invia un frame UNSUB. Parametri: *aAdapter*, *aGroup*.

Usi più comuni

- REST API
 - [Come connettersi alla REST API](#)
 - [Come inserire ordini di trading e Stop-Limit](#)
 - [Come interrogare conti, posizioni e storico ordini](#)
 - [Come ottenere dati di mercato](#)
- WebSockets API
 - [Come connettersi all'API WebSocket](#)
 - [Come sottoscrivere gli adapter di streaming](#)

Esempio

Creare il client, impostare le credenziali, connettersi, sottoscrivere uno stream di prezzi di mercato e disconnettersi.

Forex | Connect REST API

Ogni chiamata all'API di trading Forex.com / StoneX / CityIndex richiede una sessione attiva. Lo stato della sessione viene ottenuto con **LogOn**, terminato con **LogOff** e mantenuto attivo con **Ping**.

Credenziali

Impostare le credenziali sul componente client principale. Vengono copiate nella sessione REST alla prima chiamata.

- **UserName:** nome utente dell'account Forex.com / StoneX / CityIndex.
- **Password:** password dell'account.
- **AppKey:** chiave dell'applicazione rilasciata dal venue.
- **AppVersion / AppComments:** stringhe opzionali registrate al login.

Metodi di sessione

Metodo	Parametri	Descrizione
LogOn	UserName, Password, App-Key	Aprire una sessione REST. Restituisce la risposta JSON completa e memorizza SessionToken, ClientAccountId e TradingAccountId nel client principale. Tutte le successive chiamate REST e streaming si autenticano con quel token.
LogOff		Invalida la sessione sul server e cancella il token.
Ping		Mantiene la sessione attiva. La sessione del server scade dopo un periodo di inattività; emettere Ping periodicamente quando non vengono effettuate altre chiamate REST.
GetServiceStatus		Riporta lo stato corrente del servizio (nessuna autenticazione richiesta).

Flusso token

Il token restituito da **LogOn** viene allegato a ogni successiva richiesta REST come intestazione HTTP *UserName*. Lo stesso token viene riutilizzato dal client di streaming come campo *LS_password* durante la creazione della sessione Lightstreamer. Non è necessario chiamare LogOn due volte: chiamando **Connect** sul client principale viene prima eseguito il LogOn REST e solo successivamente viene aperto il socket di streaming.

Esempio

Aprire una sessione REST e chiamare Ping

Forex | Ordini Trade e Stop-Limit

Le REST API espongono cinque endpoint per la gestione degli ordini. Ogni endpoint è disponibile in tre forme di overload in modo che il chiamante possa scegliere il livello di controllo richiesto.

Endpoint

Metodo	Descrizione
NewTradeOrder	Inserisce un ordine di trading a mercato / al prezzo migliore su un mercato CFD.
UpdateTradeOrder	Collega o modifica la protezione stop/limit su un trade aperto esistente.
NewStopLimitOrder	Inserisce un ordine stop o limit in sospeso (non ancora sul mercato).
UpdateStopLimitOrder	Modifica un ordine stop o limit in sospeso esistente.
CancelOrder	Annulla un ordine stop o limit in sospeso.

Livelli di Sovraccarico

I metodi Order sono disponibili in due overload:

1. **Overload con parametri scalari** (solo NewTradeOrder e CancelOrder) — accetta i campi più utilizzati (MarketId, TradingAccountId, Direction, Quantity, BidPrice / OfferPrice, AuditId per NewTradeOrder; OrderId, TradingAccountId, MarketId per CancelOrder). L'helper costruisce la richiesta internamente. Utile per ordini semplici senza protezione.
2. **Overload con oggetto ordine tipizzato** — accetta un **TsgcHTTPForexTradeOrder**, **TsgcHTTPForexStopLimitOrder** o **TsgcHTTPForexCancelOrder**. Questi oggetti espongono ogni campo documentato inclusi **IfDone** (stop/limit bracket allegati), array **Close** (semantica chiudi per id ordine), **PositionMethodId**, **Reference** e flag di riempimento parziale. Questo è il percorso consigliato per qualsiasi cosa oltre un semplice ordine market; la convenzione di costruzione degli oggetti è condivisa con altri venue come Kucoin.

Campi di TsgcHTTPForexTradeOrder

- **MarketId**: id numerico del mercato restituito da ListCfdMarkets o FullSearchWithTags.
- **TradingAccountId**: l'account di trading attivo. Leggerlo dal client principale dopo il LogOn.
- **Direction**: *acquisto* o *vendita*.
- **Quantity**: dimensione dell'operazione nell'unità nativa del mercato.
- **BidPrice / OfferPrice**: prezzi osservati dal client al momento dell'invio dell'ordine (Forex.com li utilizza per rilevare quotazioni obsolete).
- **AuditId**: id dell'audit del prezzo dall'ultimo tick di prezzo.
- **PositionMethodId**: comportamento netting / hedging (1 = *Netting*, 2 = *Hedging*).
- **IfDone**: elenco degli ordini allegati (stop, limit, stop garantito, trailing stop) che si attivano quando l'ordine padre viene eseguito.
- **Close**: array di id di ordini da chiudere quando questo ordine viene eseguito (chiusura parziale).
- **Reference**: riferimento client fornito dal chiamante, restituito negli aggiornamenti degli ordini.

Esempio

[Compra 1000 EUR/USD a mercato con take-profit e stop-loss associati](#)

Forex | Query account

Questi endpoint leggono lo stato dell'account e la cronologia degli ordini. Richiedono una sessione attiva (vedere [Connect REST API](#)).

Endpoint

Metodo	Parametri	Descrizione
GetClientAndTradingAccount		Restituisce l'account client e l'elenco degli account di trading collegati all'utente autenticato.
ListOpenPositions	TradingAccountId	Restituisce tutte le posizioni attualmente aperte su un conto di trading.
ListActiveStopLimitOrders	TradingAccountId	Restituisce tutti gli ordini stop e limite in sospeso.
GetOrder	OrderId	Restituisce un singolo ordine tramite id, inclusa la cronologia di esecuzione e gli ordini IfDone allegati.
ListTradeHistory	TradingAccountId, Max-Results	Restituisce lo storico degli ordini di trading per il conto, dal più recente al meno recente.
ListStopLimitOrderHistory	TradingAccountId, Max-Results	Restituisce la cronologia degli ordini stop-limit per l'account, a partire dal più recente.
SimulateTrade	JSON / scalare / oggetto-ordine	Esegue un ordine di trading attraverso il validatore del server senza inviarlo. Utile per verifiche preliminari di margine e tolleranza del prezzo. Accetta gli stessi tre livelli di overload di NewTradeOrder.

Endpoint ritirati

Il riferimento ufficiale di Forex.com / StoneX / CityIndex documenta ancora alcuni endpoint che il server live non gestisce più. Questi non vengono intenzionalmente esposti dal componente:

- **GetSystemLookup** — il riferimento descrive un endpoint di lookup generico, ma il server risponde con HTTP 404 per ogni chiave di lookup nota. Utilizzi invece **GetClientAndTradingAccount** e **ListCfdMarkets**.
- **/news/headlines/*** — la famiglia news-feed è ancora elencata nella documentazione di riferimento ma restituisce HTTP 500 sul server live. Non è disponibile alcuna sostituzione tramite l'API Trading; le notizie vengono servite solo tramite il front end web.

Esempio

Elenca le posizioni aperte e le ultime 50 transazioni

Forex | Dati di mercato

Metadati statici di mercato e prezzi storici. Questi endpoint vengono utilizzati per individuare i valori **MarketId** che vengono poi passati agli endpoint di trading e agli adattatori di streaming.

Endpoint

Metodo	Parametri	Descrizione
GetMarketInformation	MarketId	Restituisce il descrittore di mercato completo (simbolo, cifre decimali, dimensione del pip, fattore di margine, orari di negoziazione).
ListCfdMarkets	SearchByMarketName, ClientAccountId, MaxResults	Elenca i mercati CFD corrispondenti a un prefisso di nome.
FullSearchWithTags	Query, Tags, MaxResults	Ricerca testuale libera in tutti i mercati con filtri opzionali per tag (FX, Indici, Azioni, Materie prime).
GetPriceBars	MarketId, Interval, Span, PriceBars	Barre OHLC storiche. Interval è uno tra MINUTE, HOUR, DAY, WEEK; Span è il numero di intervalli per barra.
GetPriceTicks	MarketId, PriceTicks	Snapshot del tick più recente per un mercato (bid / ask / mid).

Esempio

Cercare EUR/USD e leggere le ultime 100 barre da 1 minuto

Forex | Connessione all'API WebSocket

Forex.com / StoneX / CityIndex trasmette dati di mercato e aggiornamenti dell'account tramite Lightstreamer. Il componente utilizza il protocollo wire nativo Lightstreamer TLCP 2.5 su WebSocket; non è necessaria alcuna libreria nativa Lightstreamer.

Trasporto

- **Sottoprotocollo WebSocket:** *TLCP-2.5.0.lightstreamer.com*. Comunicato nell'intestazione di richiesta *Sec-WebSocket-Protocol*.
- **Adapter set:** *STREAMINGALL*. Configurato sul client del sottoprotocollo interno **Lightstreamer**; tutti gli adattatori dati di Forex.com (PRICES, ORDERS, QUOTES, CLIENTACCOUNTMARGIN, TRADEMARGIN) risiedono sotto questo adapter set.
- **Authentication:** il token di sessione ottenuto dalla chiamata REST **LogOn** viene inviato come *LS_password* durante la creazione della sessione Lightstreamer, insieme a *LS_user* impostato sul nome utente dell'account.

Riconnessione

Quando **AutoReconnect** è abilitato (impostazione predefinita), il WatchDog del client WebSocket sottostante riapre il socket dopo errori di rete. Il protocollo Lightstreamer stesso segnala i frame *LOOP* quando il server vuole che il client passi a un URL diverso o effettui il rebind della sessione; il componente gestisce questi casi in modo trasparente e genera **OnForexDisconnect** / **OnForexConnect** solo in caso di interruzioni complete della sessione. Quando la sessione REST scade (nessun Ping per un lungo periodo) il server di streaming chiude la sessione. Il componente richiama **LogOn** nuovamente utilizzando le credenziali memorizzate, aggiorna **SessionToken** e riavvia la sessione Lightstreamer con il nuovo token.

Esempio

Apre la sessione di streaming

Forex | Sottoscrizione agli Adapter in Streaming

I dati in streaming vengono distribuiti tramite cinque adattatori dati Lightstreamer all'interno del set di adattatori *STREAMINGALL*. Ogni adattatore ha la propria convenzione di denominazione degli item e la propria modalità di sottoscrizione — **MERGE** per lo stato di tipo snapshot (prezzo corrente, margine corrente del conto) e **RAW** per i canali a flusso di eventi (aggiornamenti degli ordini, aggiornamenti delle posizioni, eventi di quotazione).

Adapters

DataAdapter	Item	Modalità	Event	Campi
PRICES	ID.{MarketId}	MERGE	OnForexPriceTick	MarketId, TickDate, Bid, OfferPrice, High, Low, Change, Direction, AuditId, StatusSummary
ORDERS	ORDERS.{ClientAccountId}	RAW	OnForexOrderUpdate	OrderId, MarketId, TradingAccountId, Direction, Quantity, Price, Status, OrderType, Currency, LastChangedDateTimeUTC
QUOTES	QUOTES.{ClientAccountId}	RAW	OnForexQuote	QuoteId, OrderId, RequestDateTimeUTC, ApprovalDateTimeUTC, BidPrice, OfferPrice, Status
CLIENTACCOUNT-MARGIN	ID.{ClientAccountId}	MERGE	OnForexAccountMargin	ClientAccountId, Cash, MarginMarginIndicator, NetEquity, OpenTradeEquity, TotalMarginRequirement, TradeableFunds, CurrencyId, CurrencySymbol, SOCode
TRADEMARGIN	TRADEMARGIN.{ClientAccountId}	RAW	OnForexPositionUpdate	OrderId, ClientAccountId, MarketId, MarketTypeId, DirectionId, Direction, Quantity, OpenPrice, CurrentPrice, Multiplier, MarginRequirement, Currency, OpenTradeEquity

Note:

- PRICES e CLIENTACCOUNTMARGIN utilizzano il formato *ID.{numericId}* per gli elementi e trasmettono un singolo record modificabile per elemento (MERGE).
- ORDERS, QUOTES e TRADEMARGIN utilizzano il prefisso legacy per-adapter (*ORDERS.{id}*, *QUOTES.{id}*, *TRADEMARGIN.{id}*) e trasmettono un flusso continuo di eventi (RAW). Un elemento contiene ogni ordine / quotazione / operazione per il **ClientAccountId** connesso; i record si identificano tramite **OrderId** / **QuoteId**.
- Il record TRADEMARGIN corrisponde al *TradeMarginDTO* ufficiale dell'SDK CityIndex legacy. **DirectionId** arriva come codice numerico (1 = acquisto, 2 = vendita); la libreria espone anche **Direction** come stringa "buy" / "sell". **OpenPrice** è il prezzo di ingresso della transazione (campo server *PriceTakenAt*), **CurrentPrice** il prezzo di valutazione più recente (*PriceCalculatedAt*), **OpenTradeEquity** il P&L flottante nella valuta del conto (*OTEConverted*).
- Il campo **Direction** degli ORDERS è il codice numerico pubblicato dal server (1 / 2); **Status** è una stringa (ad es. "Accepted", "Open", "Cancelled"); **OrderType** è una stringa ("Market" / "Stop" / "Limit").

- Il feed TRADEMARGIN non pubblica **TradingAccountId**, **MarketName** per singola operazione né un timestamp per operazione; un elemento trasporta tutte le posizioni per il **ClientAccountId** connesso. Chiamare **REST_API.ListOpenPositions** per quei campi. Il feed ORDERS non pubblica **TriggerPrice** né **Applicability**; chiamare **REST_API.GetOrder** quando necessario.

Metodi helper

Il client principale racchiude la logistica SUB / UNSUB di Lightstreamer. I più comuni sono:

- **WatchMarket(aMarketId)**: si sottoscrive ai PREZZI per un mercato. Gli aggiornamenti arrivano in **OnForexPriceTick**.
- **UnwatchMarket(aMarketId)**: rimuove la sottoscrizione.
- **WatchAccount**: sottoscrive a ORDERS, CLIENTACCOUNTMARGIN e TRADEMARGIN per gli account connessi. Gli aggiornamenti arrivano negli eventi corrispondenti.
- **WatchQuotes**: si sottoscrive all'adapter QUOTES per i flussi di lavoro di richiesta di quotazione.

Eventi del Ciclo di Vita della Sottoscrizione

Il client principale genera due eventi del ciclo di vita che rispecchiano i frame TLCP SUBOK / UNSUB sottostanti:

- **OnForexSubscribe(Sender, aAdapter, aGroup, altems, aFields)**: si attiva quando il server conferma qualsiasi sottoscrizione Lightstreamer. *aAdapter* è il nome dell'adattatore dati (PRICES, ORDERS, QUOTES, CLIENTACCOUNTMARGIN, TRADEMARGIN); *aGroup* è il gruppo di elementi (ad es. *ID.<ClientAccountId>* o *ORDERS.<ClientAccountId>*); *altems* e *aFields* sono i conteggi riportati dal server nel frame SUBOK / SUBCMD.
- **OnForexUnsubscribe(Sender, aAdapter, aGroup)**: generato al termine di un'iscrizione — tramite **UnwatchMarket**, alla disconnessione dallo streaming, o quando il server invia un frame UNSUB.

Esempio

Sottoscrivere i tick del prezzo EUR/USD

API Cex

Cex

La API WebSocket consente di ricevere notifiche in tempo reale senza inviare richieste aggiuntive, rendendola un metodo più rapido per ottenere dati dall'exchange

Il componente Cex ha una proprietà chiamata Cex dove è possibile inserire le chiavi API fornite da Cex per accedere ai dati del proprio account.

Codifica dei messaggi

- Tutti i messaggi sono codificati in formato JSON.
- I prezzi sono presentati come stringhe per evitare errori di arrotondamento durante il parsing JSON sul lato client.
- La compressione dei frame WebSocket non è supportata dal server.
- Il tempo è presentato come timestamp UNIX intero in secondi.

Autenticazione

Per accedere ai dati WebSocket di CEX.IO è necessario essere autorizzati.

- Accedi all'account CEX.IO.
- Andare alla pagina <https://cex.io/trade/profile#/api>.
- Selezionare il tipo di permessi richiesti.
- Faccia clic sul pulsante "Generate Key" e salvi la Sua secret key, poiché diventerà inaccessibile dopo l'attivazione.
- Attivare la propria chiave.

Connettività

- Se un socket connesso è inattivo per 15 secondi, il server CEX.IO invierà un messaggio PING.
- Solo il server può essere un iniziatore di una richiesta PING.
- Il server invia il ping solo all'utente autenticato.
- L'utente deve rispondere con un messaggio PONG. In caso contrario, il WebSocket verrà DISCONNESSO. Questo viene gestito automaticamente dalla libreria.
- Per l'utente autenticato, nel caso in cui non vi siano notifiche o ping dal server entro 15 secondi, sarebbe più sicuro inviare una richiesta come 'ticker' o 'get-balance' e ricevere una risposta, al fine di garantire la connettività e l'autenticazione.

Canali pubblici

Questi canali non richiedono autenticazione. Le risposte del server vengono ricevute tramite l'evento OnCexMessage.

- **SubscribeTickers:** Feed ticker con solo il prezzo delle transazioni effettuate su tutte le coppie (deprecato)

```
{
  "e": "tick",
  "data": {
    "symbol1": "BTC",
    "symbol2": "USD",
    "price": "428.0123"
```

```
}
}
```

- **SubscribeChart:** feed di grafici OHLCV con valori Open, High, Low, Close, Volume (deprecato)

```
{
  'e': 'ohlcv24',
  'pair': 'BTC:USD',
  'data': [
    '418.2936',
    '420.277',
    '412.09',
    '416.9778',
    '201451078368'
  ]
}
```

- **Subscribe Pair:** feed Market Depth (deprecato)

```
{
  'e': 'md_grouped',
  'data': {
    'pair': 'BTC:USD',
    'id': 11296131,
    'sell': {
      '427.5000': 1000000,
      '480.0000': 263544334,
      ...
    },
    'buy': {
      '385.0000': 3630000,
      '390.0000': 1452458642,
      ... 400+ pairs together with 'sell' pairs
    }
  }
}
```

- **Sottoscrivi coppia:** feed del book degli ordini (deprecato)

```
{
  'e': 'md',
  'data': {
    'pair': 'BTC:USD',
    'buy_total': 63221099,
    'sell_total': 112430315118,
    'id': 11296131,
    'sell': [
      [426.45, 10000000],
      [426.5, 66088429300],
      [427, 1000000],
      ... 50 pairs overaall
    ],
    'buy': [
      [423.3, 4130702],
      [423.2701, 10641168],
      [423.2671, 1000000],
      ... 50 pairs overaall
    ]
  }
}
```

Canali privati

Per accedere a questi canali, chiamare prima il metodo `Authenticate`. Le risposte dal server vengono ricevute tramite l'evento `OnCexMessage`.

GetTicker

```
{
  "e": "ticker",
  "data": {
    "timestamp": "1471427037",
    "low": "290",
    "high": "290",
    "last": "290",
    "volume": "0.02062068",
    "volume30d": "14.38062068",
    "bid": 240,
    "ask": 290,
    "pair": [
      "BTC",
      "USD"
    ]
  },
  "oid": "1471427036908_1_ticker",
  "ok": "ok"
}
```

GetBalance

```
{
  "e": "get-balance",
  "data": {
    "balance": {
      'LTC': '10.00000000',
      'USD': '1024.00',
      'RUB': '35087.98',
      'EUR': '217.53',
      'GHS': '10.00000000',
      'BTC': '9.00000000'
    },
    "obalance": {
      'BTC': '0.12000000',
      'USD': '512.00',
    },
  },
  "time": 1435927928597
  "oid": "1435927928274_2_get-balance",
  "ok": "ok"
}
```

SubscribeOrderBook

```
{
  "e": "order-book-subscribe",
  "data": {
    "timestamp": 1435927929,
    "bids": [
      [
        241.947,
        155.91626
      ],
      [
        241,
        981.1255
      ],
    ],
    "asks": [
      [
        241.95,
        15.4613
      ],
      [
        241.99,
        17.3303
      ],
    ],
    "pair": "BTC:USD",
    "id": 67809
  },
  "oid": "1435927928274_5_order-book-subscribe",
}
```



```
"ok": "ok"
}
```

UnSubscribeOrderBook

```
{
  "e": "order-book-unsubscribe",
  "data": {
    "pair": "BTC:USD"
  },
  "oid": "1435927928274_4_order-book-unsubscribe",
  "ok": "ok"
}
```

GetOpenOrders

```
{
  "e": "open-orders",
  "data": [
    {
      "id": "2477098",
      "time": "1435927928618",
      "type": "buy",
      "price": "241.9477",
      "amount": "0.02000000",
      "pending": "0.02000000"
    },
    {
      "id": "2477101",
      "time": "1435927928634",
      "type": "sell",
      "price": "241.9493",
      "amount": "0.02000000",
      "pending": "0.02000000"
    }
  ],
  "oid": "1435927928274_9_open-orders",
  "ok": "ok"
}
```

PlaceOrder

```
{
  "e": "place-order",
  "data": {
    "complete": false,
    "id": "2477098",
    "time": "1435927928618",
    "pending": "0.02000000",
    "amount": "0.02000000",
    "type": "buy",
    "price": "241.9477"
  },
  "oid": "1435927928274_7_place-order",
  "ok": "ok"
}
```

CancelReplaceOrder

```
{
  "e": "cancel-replace-order",
  "data": {
    "complete": false,
    "id": "2689009",

```

```

    "time": 1443464955904,
    "pending": "0.04000000",
    "amount": "0.04000000",
    "type": "buy",
    "price": "243.25"
  },
  "oid": "1443464955209_16_cancel-replace-order",
  "ok": "ok"
}

```

GetOrderRequest

Nel sistema CEX.IO, gli ordini possono essere presenti nel motore di trading o in un database di archivio. Possono esserci periodi di tempo (~2 secondi o più) in cui l'ordine è completato/annullato, ma non è ancora stato spostato nel database di archivio. Ciò significa che non è possibile vederlo tramite le chiamate: archived-orders/open-orders. Questa chiamata consente di ottenere le informazioni sull'ordine in qualsiasi caso. Le risposte possono avere formati diversi a seconda della posizione degli ordini.

```

{
  "e": "get-order",
  "data": {
    "user": "xxx",
    "type": "buy",
    "symbol1": "BTC",
    "symbol2": "USD",
    "amount": "0.02000000",
    "remains": "0.02000000",
    "price": "50.75",
    "time": 1450214742160,
    "tradingFeeStrategy": "fixedFee",
    "tradingFeeBuy": "5",
    "tradingFeeSell": "5",
    "tradingFeeUserVolumeAmount": "nil",
    "a:USD:c": "1.08",
    "a:USD:s": "1.08",
    "a:USD:d": "0.00",
    "status": "a",
    "orderId": "5582060"
  },
  "oid": "1450214742135_10_get-order",
  "ok": "ok"
}

```

CancelOrderRequest

```

{
  "e": "cancel-order",
  "data": {
    "order_id": "2477098"
    "time": 1443468122895
  },
  "oid": "1435927928274_12_cancel-order",
  "ok": "ok"
}

```

GetArchivedOrders

```

{
  "e": "archived-orders",
  "data": [
    {
      "type": "buy",
      "symbol1": "BTC",
      "symbol2": "USD",
      "amount": 0,
      "amount2": 5000,
      "remains": 0,
      "time": "2015-04-17T10:46:27.971Z",
      "tradingFeeBuy": "2",
      "tradingFeeSell": "2",

```

```

    "ta:USD": "49.00",
    "fa:USD": "0.98",
    "orderId": "2340298",
    "status": "d",
    "a:BTC:cds": "0.18151851",
    "a:USD:cds": "50.00",
    "f:USD:cds": "0.98"
  },
  {
    "type": "buy",
    "symbol1": "BTC",
    "symbol2": "USD",
    "amount": 0,
    "amount2": 10000,
    "remains": 0,
    "time": "2015-04-08T15:46:04.651Z",
    "tradingFeeBuy": "2.99",
    "tradingFeeSell": "2.99",
    "ta:USD": "97.08",
    "fa:USD": "2.91",
    "orderId": "2265315",
    "status": "d",
    "a:BTC:cds": "0.39869578",
    "a:USD:cds": "100.00",
    "f:USD:cds": "2.91"
  }
],
"oid": "1435927928274    15_archived-orders",
"ok": "ok"
}

```

OpenPosition

```

{
  "e": "open-position",
  "oid": "1435927928274_7_open-position",
  "data": {
    "amount": '1',
    "symbol": 'BTC',
    "pair": [
      "BTC",
      "USD"
    ],
    "leverage": '2',
    "ptype": 'long',
    "anySlippage": 'true',
    "eoprice": '650.3232',
    "stopLossPrice": '600.3232'
  }
}

```

GetPosition

```

{
  "e": "get_position",
  "ok": "ok",
  "data": {
    "user": "ud100036721",
    "pair": "BTC:USD",
    "amount": "1.00000000",
    "symbol": "BTC",
    "msymbol": "USD",
    "omamount": "1528.77",
    "lsymbol": "USD",
    "lamount": "3057.53",
    "slamount": "3380.11",
    "leverage": "3",
    "stopLossPrice": "3380.1031",
    "dfl": "3380.10310000",
    "flPrice": "3057.53333333",
    "otime": 1513002370342,
    "psymbol": "BTC",
    "ptype": "long",
    "ofee": "10",
    "pfee": "10",
    "cfee": "10",
    "tfeeAmount": "152.88",

```

```

    "rinterval": "14400000",
    "okind": "Manual",
    "a:BTC:c": "1.00000000",
    "a:BTC:s": "1.00000000",
    "oorder": "89101551",
    "pamount": "1.00000000",
    "lremains": "3057.53",
    "slremains": "3380.11",
    "oprice": "4586.3000",
    "status": "a",
    "id": "125531",
    "a:USD:cds": "4739.18"
  }
}

```

GetOpenPositions

```

{
  'e': 'open_positions',
  "oid": "1435927928256_7_open-positions",
  'ok': 'ok',
  'data': [
    {
      'user': 'ud100036721',
      'id': '104102',
      'otime': 1475602208467,
      'symbol': 'BTC',
      'amount': '1.00000000',
      'leverage': '2',
      'ptype': 'long',
      'psymbol': 'BTC',
      'msymbol': 'USD',
      'lsymbol': 'USD',
      'pair': 'BTC:USD',
      'oprice': '607.5000',
      'stopLossPrice': '520.3232',
      'ofee': '1',
      'pfee': '3',
      'cfee': '4',
      'tfeeAmount': '3.04',
      'pamount': '1.00000000',
      'omamount': '303.75',
      'lamount': '303.75',
      'oorder': '34106774',
      'rinterval': '14400000',
      'dfl': '520.32320000',
      'slamount': '520.33',
      'slremains': '520.33',
      'lremains': '303.75',
      'flPrice': '303.75000000',
      'a:BTC:c': '1.00000000',
      'a:BTC:s': '1.00000000',
      'a:USD:cds': '610.54',
    },
    ...
  ]
}

```

ClosePosition

```

{
  'e': 'close_position',
  "oid": "1435927928364_7_close-position",
  'ok': 'ok',
  'data': {
    'id': 104034,
    'ctime': 1475484981063,
    'ptype': 'long',
    'msymbol': 'USD',
    'pair': {
      'symbol1': 'BTC',
      'symbol2': 'USD'
    },
    'price': '607.1700',
    'profit': '-12.48',
  }
}

```


API Cex Plus

Cex Plus

API supportate

- **API WebSockets:** si connette a un server websocket pubblico e fornisce aggiornamenti in tempo reale dei dati di mercato.

WebSockets API

WebSocket è un protocollo di comunicazione full-duplex basato su TCP. Full-duplex significa che entrambe le parti possono inviarsi messaggi in modo asincrono utilizzando lo stesso canale di comunicazione. Questa sezione descrive quali messaggi Exchange Plus e Client devono inviarsi reciprocamente. Tutti i messaggi devono essere oggetti JSON validi.

L'API WebSocket è utilizzata principalmente per ottenere informazioni o eseguire azioni che non sono disponibili o non facili da eseguire utilizzando l'API REST. Tuttavia, alcune richieste o azioni sono possibili sia nell'API REST sia nell'API WebSocket. Exchange Plus invia messaggi al Client come risposta a una richiesta inviata in precedenza dal Client, o come notifica di un evento (senza una precedente richiesta del Client).

Chiamate API pubbliche

Il limite di frequenza delle richieste all'API pubblica è implicito per proteggere il sistema da attacchi DDoS e garantire che tutti i client abbiano lo stesso livello di accesso stabile agli endpoint dell'Exchange Plus API. Le richieste pubbliche sono limitate per indirizzo IP da cui vengono effettuate. I limiti di richiesta sono determinati dal costo associato a ciascuna chiamata API pubblica. Per impostazione predefinita, ogni richiesta pubblica ha un costo di 1 punto, ma per alcune richieste specifiche questo costo può essere superiore. Consultare le informazioni aggiornate sul costo del limite di frequenza nella specifica di ciascun metodo.

Exchange Plus limita le chiamate Public API a un massimo di 100 punti al minuto, considerando che ogni chiamata Public API ha il suo costo (veda di seguito). Se il limite della frequenza delle richieste viene raggiunto, Exchange Plus risponde con un errore, invia l'evento disconnected al client e successivamente chiude la connessione WS. Exchange Plus continuerà a servire il client a partire dal minuto di calendario successivo. Nell'esempio seguente, il contatore delle richieste verrà reimpostato alle 11:02:00.000.

Method	Description
GetTicker	Questo metodo è progettato per ottenere informazioni attuali sul Ticker, inclusi dati sui prezzi correnti, variazioni di prezzo e volume nelle ultime 24 ore, ultimo evento di scambio, ecc. di determinati asset.
GetOrderBook	Questo metodo consente al client di ricevere l'istantanea corrente del book degli ordini per una specifica coppia di trading.
GetCandles	Tramite il metodo Candles il client può ricevere candele OHLCV storiche di diverse risoluzioni e tipi di dati. Il client può indicare filtri aggiuntivi di intervallo temporale e limite per rendere la risposta più precisa alle esigenze del client.
GetTradeHistory	Questo metodo consente al Client di ottenere dati storici sulle operazioni avvenute sulla coppia di trading richiesta. Il Client può integrare la richiesta di storico operazioni con parametri di filtro aggiuntivi, come il periodo temporale, l'intervallo di trading, il lato, ecc., per ricevere le operazioni corrispondenti ai parametri della richiesta.
GetServerTime	Questo metodo viene utilizzato per ottenere l'ora corrente sul server Exchange Plus. Può essere utile per le applicazioni che devono essere sincronizzate con l'ora del server.

GetPairsInfo	Il metodo Pair Info consente al Client di ricevere i parametri per tutte le coppie di trading supportate.
GetCurrenciesInfo	Il metodo Currencies Info consente al Client di ricevere i parametri per tutte le valute configurate in Exchange Plus, nonché la disponibilità di deposito e prelievo tra Exchange Plus e CEX.IO Wallet.
GetProcessingInfo	Questa richiesta consente al Cliente di ricevere informazioni dettagliate sulle opzioni disponibili per effettuare depositi da portafogli esterni e prelievi verso portafogli esterni per ciascuna criptovaluta supportata, inclusi il nome della criptovaluta e le blockchain disponibili per depositi e prelievi. Inoltre, per ciascuna blockchain supportata sono indicati il tipo di criptovaluta sulla blockchain specificata, la disponibilità attuale di deposito e prelievo, gli importi minimi per depositi e prelievi e le commissioni di prelievo esterno. Le informazioni di elaborazione rendono il Cliente più flessibile nella scelta della blockchain desiderata per ricevere l'indirizzo di deposito e avviare prelievi esterni tramite una determinata blockchain, consentendo così al Cliente di utilizzare il metodo più comodo per trasferire i propri asset crypto da o verso l'ecosistema CEX.IO.
SubscribeOrder-Book	Il client, iscrivendosi tramite WebSocket, può sottoscrivere il feed del libro degli ordini per la coppia di trading richiesta. In risposta alla richiesta di sottoscrizione al Libro degli Ordini, il client riceverà l'istantanea corrente (iniziale) del libro degli ordini per la coppia richiesta con il numero seqId indicato. Per tracciare i successivi aggiornamenti al Libro degli Ordini, il client deve sottoscrivere tramite WebSocket i messaggi "order_book_increment", che conterranno il nome della coppia di trading, il numero seqId e i delta dei livelli di prezzo Bids e Asks.
UnSubscribeOrder-Book	Cancellazione dell'iscrizione al canale del libro degli ordini.
SubscribeTrade	Utilizzando il metodo Trade Subscribe, il client può sottoscrivere tramite WebSocket al flusso live degli eventi di trade che si verificano sulla coppia di trading richiesta. In risposta alla richiesta Trade Subscribe, il client riceverà un identificatore univoco della sottoscrizione di trade da utilizzare successivamente per l'annullamento della sottoscrizione quando non è più necessaria. Il client deve sottoscrivere tramite WebSocket ai messaggi "tradeHistory-Snapshot" e "tradeUpdate" per ricevere snapshot iniziali e periodici della cronologia dei trade, nonché eventi di trade live per la coppia di trading richiesta.
UnSubscribeTrade	Annullamento dell'iscrizione al canale di trading.

Esempio: ottenere l'ultimo ticker della coppia BTC-USD

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_CexPlus oCexPlus = new TsgcWSAPI_CexPlus();
oCexPlus.Client = oClient;
oCexPlus.OnCexPlusConnect += OnCexPlusConnectEvent;
oCexPlus.OnCexPlusMessage += OnCexPlusMessageEvent;
oClient.Active = true;

void OnCexPlusConnectEvent(object Sender)
{
    oCexPlus.GetTicker("BTC-USD");
}

void OnCexPlusMessageEvent(object Sender, string Event, string Msg)
{
    MessageBox.Show("Ticker data: " + Msg);
}
```

Chiamate API private

Exchange Plus utilizza chiavi API per consentire l'accesso alle API private.

Il client può generare, configurare e gestire le chiavi API, impostare i livelli di autorizzazione, gli IP nella whitelist per le chiavi API, ecc. tramite Exchange Plus Web Terminal nella sezione Gestione Chiavi API del Profilo.

Limite API Keys: Per impostazione predefinita il Client può avere fino a 5 API Keys.

Per limitare l'accesso a determinate funzionalità durante l'utilizzo delle API Key, è necessario definire un insieme specifico di permessi per ogni API Key. L'insieme di permessi definito può essere modificato successivamente se necessario.

Per le API Key sono disponibili i seguenti livelli di autorizzazione:

- **Lettura:** livello di autorizzazione per la visualizzazione dei dati relativi all'account, la ricezione di report, la sottoscrizione ai dati di mercato ecc.
- **Trade:** livello di autorizzazione che consente di piazzare e annullare ordini per conto del conto.
- **Funds Internal:** livello di autorizzazione che consente il trasferimento di fondi tra account (tra sub-account o account principale e sub-account) del Portfolio CEX.IO Exchange Plus.
- **Funds Wallet:** livello di autorizzazione che consente il trasferimento di fondi dai conti Portfolio CEX.IO Exchange Plus (conto principale e sotto-conti) verso CEX.IO Wallet e viceversa.

Method	Description
GetCurrentFee	Questo metodo indica le commissioni correnti in un momento specifico, tenendo conto del volume aggiornato a 30 giorni del Client e del giorno della settimana (le commissioni possono essere diverse, ad esempio, nei fine settimana).
GetFeeStrategy	La strategia delle commissioni restituisce tutte le opzioni di commissione applicabili al Cliente, tenendo conto del volume di trading del Cliente, del giorno della settimana, delle coppie, del gruppo di coppie, ecc. Questo metodo fornisce informazioni sulla strategia generale delle commissioni, che include tutti i possibili valori delle commissioni di trading applicabili al Cliente. Per ricevere le commissioni di trading correnti in base al volume di trading degli ultimi 30 giorni del Cliente, il Cliente deve utilizzare il metodo [Commissione corrente]. Per ricevere il volume di trading corrente degli ultimi 30 giorni, il Cliente deve utilizzare il metodo [Volume].
GetVolume	Questa richiesta consente al cliente di ricevere il proprio volume di trading degli ultimi 30 giorni in equivalente USD.
CreateAccount	Questa richiesta consente al cliente di creare un nuovo sotto-account. Per impostazione predefinita, il cliente può avere fino a 5 sotto-account, incluso il conto principale.
GetAccountStatus	Tramite il metodo Account Status V3, il Client può visualizzare il saldo attuale e il suo equivalente indicativo nella valuta di conversione (per impostazione predefinita "USD"), gli importi bloccati negli ordini aperti (attivi) per ciascun sotto-account e valuta. Se è disponibile un saldo commissioni di trading per il Client, la risposta conterrà anche i dati generali del saldo commissioni di trading come nome promo, nome valuta, saldo totale e data di scadenza della promo sul saldo commissioni di trading. È responsabilità del Client monitorare il saldo di trading disponibile del proprio sotto-account come saldo attuale del sotto-account ridotto dell'importo del saldo bloccato negli ordini aperti (attivi) del sotto-account.
GetOrders	Questa richiesta consente al Client di scoprire informazioni sui propri ordini.
NewOrder	Il client può inserire nuovi ordini tramite l'API WebSocket utilizzando Do My New Order Request. Insieme a una risposta a questa richiesta, Exchange Plus invia messaggi Account Event e Execution Report al client se la richiesta ha esito positivo. Il messaggio di risposta indica l'ultimo stato aggiornato dell'ordine disponibile nel sistema al momento dell'invio della risposta. Se il client non ha ricevuto un messaggio di risposta a Do My New Order Request, può interrogare lo stato corrente dell'ordine utilizzando Get My Orders Request con il parametro clientId. Quando si invia una richiesta per un nuovo ordine, si raccomanda vivamente di utilizzare il parametro clientId che corrisponde alla specifica richiesta di nuovo ordine sul lato del client. Exchange Plus evita di inserire più ordini con lo stesso clientId. Se vengono identificati più di un nuovo ordine con lo stesso clientId e gli stessi altri parametri dell'ordine, Exchange Plus inserisce solo il primo ordine e restituisce lo stato

	di tale ordine al client in risposta alle seconde e successive richieste di nuovo ordine con gli stessi parametri. Se vengono identificati più di un nuovo ordine con lo stesso clientOrderId ma con diversi altri parametri dell'ordine, Exchange Plus elabora solo il primo ordine e rifiuta le seconde e successive richieste di nuovo ordine con lo stesso clientOrderId ma con diversi altri parametri dell'ordine.
NewMarketOrder	Inserisce un nuovo ordine a mercato.
NewLimitOrder	Inserisce un nuovo ordine limite.
CancelOrder	Il client può annullare gli ordini. Insieme alla risposta a questa richiesta, Exchange Plus invia al client messaggi di Account Event e Execution Report se la richiesta ha successo. Inoltre, se la richiesta di annullamento di un ordine viene rifiutata, Exchange Plus invia un messaggio di Order Cancellation Rejection.
CancelAllOrders	Il client può annullare tutti gli ordini aperti tramite WebSocket API. Insieme alla risposta a questa richiesta, Exchange Plus avvierà il processo di cancellazione per tutti gli ordini aperti e invierà al client i messaggi Account Event e Execution Report corrispondenti.
GetTransactionHistory	Questa richiesta consente al Client di conoscere le proprie transazioni finanziarie (depositi, prelievi, trasferimenti interni, commissioni o operazioni di trading).
GetFundingHistory	Questa richiesta consente al client di trovare le proprie transazioni di deposito e prelievo.
InternalTransfer	Il client può richiedere il trasferimento di fondi tra i propri sub-account o tra il conto principale e un sub-account. Exchange Plus non addebita alcuna commissione al client per il trasferimento di fondi tra i propri account. Insieme alla risposta a questa richiesta, Exchange Plus invia messaggi di evento account al client se la richiesta ha esito positivo.
GetDepositAddress	Questo metodo può essere utilizzato dal Client per ricevere un indirizzo crypto in cui depositare criptovalute. L'indirizzo di deposito può essere generato per account principali e secondari. L'elenco delle blockchain disponibili per la generazione dell'indirizzo di deposito può essere ottenuto dal Client tramite la richiesta Get Processing Info.
FundsDeposit-FromWallet	Il client può depositare fondi dal Wallet CEX.IO all'account Exchange Plus. Il sistema evita l'elaborazione di più richieste di deposito con lo stesso clientTxId. Se vengono ricevute più richieste di deposito con lo stesso clientTxId, il sistema elabora solo la prima richiesta di deposito e rifiuta la seconda e le successive richieste di deposito con lo stesso clientTxId.
FundsWithdrawal-ToWallet	Il client può prelevare fondi dall'account Exchange Plus verso il Portafoglio CEX.IO. Il sistema evita richieste di prelievo multiple con lo stesso clientTxId. Se vengono ricevute più richieste di prelievo con lo stesso clientTxId, il sistema elabora solo la prima e rifiuta la seconda e le successive con lo stesso clientTxId.
GetWalletBalance	Recupera le informazioni sul saldo del Wallet CEX.IO.
SubscribeAccountEvents	Sottoscrive le notifiche di eventi dell'account in tempo reale (variazioni di saldo, esecuzioni di ordini).
UnSubscribeAccountEvents	Annulla l'iscrizione alle notifiche degli eventi dell'account.

Esempio: ottenere gli ordini.

```

TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_CexPlus oCexPlus = new TsgcWSAPI_CexPlus();
oCexPlus.Client = oClient;
oCexPlus.CexPlus.ApiKey = "your-api-key";
oCexPlus.CexPlus.ApiSecret = "your-api-secret";
oCexPlus.OnCexPlusAuthenticated += OnCexPlusAuthenticatedEvent;
oCexPlus.OnCexPlusMessage += OnCexPlusMessageEvent;
oClient.Active = true;

void OnCexPlusAuthenticatedEvent(object Sender)
{
    oCexPlus.GetOrders();
}

```

```
void OnCexPlusMessageEvent(object Sender, string Event, string Msg)
{
    MessageBox.Show("Orders: " + Msg);
}
```

API Discord

Discord

Nota: A partire dall'API v10, il dominio delle API Discord è cambiato da `discordapp.com` a `discord.com`. Il componente è stato aggiornato per utilizzare la versione 10 dell'API e il nuovo dominio.

I gateway sono la forma di comunicazione in tempo reale di Discord tramite WebSocket sicuri. I client ricevono eventi e dati tramite il gateway a cui sono connessi e inviano dati tramite l'API REST.

Authorization

Prima di tutto è necessario generare un nuovo Bot e copiare il Bot Token che verrà utilizzato per autenticarsi tramite API. Quindi impostare questo token nel Componente API.

```
TsgcWSAPI_Discord1.DiscordOptions.BotOptions.Token = "...bot token here...";
```

Intents

Mantenere un'applicazione con stato può essere difficile quando si tratta della quantità di dati che ci si aspetta di elaborare, specialmente su larga scala. I Gateway Intents sono un sistema che aiuta a ridurre tale onere computazionale.

Durante l'identificazione presso il gateway, è possibile specificare un parametro intents che consente di sottoscrivere in modo condizionale a "intenti" predefiniti, gruppi di eventi definiti da Discord. Se non si specifica un determinato intento, non si riceverà nessuno degli eventi gateway raggruppati in quel gruppo. Gli intents validi sono (il valore zero indica che tutti gli eventi vengono ricevuti):

```
GUILDS (1 << 0) = Integer (1)
- GUILD_CREATE
- GUILD_DELETE
- GUILD_ROLE_CREATE
- GUILD_ROLE_UPDATE
- GUILD_ROLE_DELETE
- CHANNEL_CREATE
- CHANNEL_UPDATE
- CHANNEL_DELETE
- CHANNEL_PINS_UPDATE
GUILD_MEMBERS (1 << 1) = Integer (2)
- GUILD_MEMBER_ADD
- GUILD_MEMBER_UPDATE
- GUILD_MEMBER_REMOVE
GUILD_BANS (1 << 2) = Integer (4)
- GUILD_BAN_ADD
- GUILD_BAN_REMOVE
GUILD_EMOJIS (1 << 3) = Integer (8)
- GUILD_EMOJIS_UPDATE
GUILD_INTEGRATIONS (1 << 4) = Integer (16)
- GUILD_INTEGRATIONS_UPDATE
GUILD_WEBHOOKS (1 << 5) = Integer (32)
- WEBHOOKS_UPDATE
GUILD_INVITES (1 << 6) = Integer (64)
- INVITE_CREATE
- INVITE_DELETE
GUILD_VOICE_STATES (1 << 7) = Integer (128)
- VOICE_STATE_UPDATE
GUILD_PRESENCES (1 << 8) = Integer (256)
```

- PRESENCE_UPDATE
- GUILD_MESSAGES (1 << 9) = Integer (512)
- MESSAGE_CREATE
- MESSAGE_UPDATE
- MESSAGE_DELETE
- GUILD_MESSAGE_REACTIONS (1 << 10) = Integer (1024)
- MESSAGE_REACTION_ADD
- MESSAGE_REACTION_REMOVE
- MESSAGE_REACTION_REMOVE_ALL
- MESSAGE_REACTION_REMOVE_EMOJI
- GUILD_MESSAGE_TYPING (1 << 11) = Integer (2048)
- TYPING_START
- DIRECT_MESSAGES (1 << 12) = Integer (4096)
- CHANNEL_CREATE
- MESSAGE_CREATE
- MESSAGE_UPDATE
- MESSAGE_DELETE
- CHANNEL_PINS_UPDATE
- DIRECT_MESSAGE_REACTIONS (1 << 13) = Integer (8192)
- MESSAGE_REACTION_ADD
- MESSAGE_REACTION_REMOVE
- MESSAGE_REACTION_REMOVE_ALL
- MESSAGE_REACTION_REMOVE_EMOJI
- DIRECT_MESSAGE_TYPING (1 << 14) = Integer (16384)
- TYPING_START

Heartbeat

Gli heartbeat sono gestiti automaticamente dal componente, quindi non deve preoccuparsene. Quando il client si connette al server, il server invia una risposta HELLO con un intervallo di heartbeat, e il componente legge la risposta e regola automaticamente l'heartbeat per inviare un ping ogni x secondi. A volte il server può inviare un ping al client; anche questo viene gestito automaticamente dal client.

Connessione Pronta

Quando la connessione è pronta, dopo un accesso e un'autorizzazione riusciti da parte del server, viene sollevato l'evento **OnDiscordReady** e quindi è possibile iniziare a ricevere aggiornamenti dal server.

Ripresa della connessione

Se la connessione si chiude in modo imprevisto, quando il client tenta di riconnettersi, chiama l'evento **OnDiscordBeforeReconnect**. Il componente salva automaticamente tutti i dati necessari per effettuare una ripresa riuscita, ma i parametri possono essere modificati se necessario. Se non desidera riconnettersi e avviare una nuova sessione pulita, imposti semplicemente Reconnect su False.

Se la sessione viene ripristinata, viene generato l'evento **OnDiscordResumed**. Se si tratta di una nuova sessione, verrà generato **OnDiscordReady**.

Dispatch Events

Gli eventi vengono distribuiti tramite **OnDiscordDispatch**, quindi qui è possibile leggere gli eventi inviati dal server al client.

```
void OnDiscordDispatch(object Sender, string aEvent, string RawData)
{
    DoLog("#discord dispatch: " + aEvent + " " + RawData);
}
```

aEvent il parametro contiene il nome dell'evento.

RawData contiene il messaggio JSON completo.

Richieste HTTP

Per richiedere informazioni su guild, utenti o per aggiornare dati... invece di utilizzare i messaggi del gateway websocket, Discord richiede l'uso di richieste HTTP. Di seguito sono elencati tutti i metodi disponibili per effettuare una richiesta HTTP:

```
function GET_Request(const aPath: String): string;
function POST_Request(const aPath, aMessage: String): string;
function PUT_Request(const aPath, aMessage: String): string;
function PATCH_Request(const aPath, aMessage: String): string;
function DELETE_Request(const aPath: String): string;
```

Esempio: ottieni le informazioni sull'utente corrente

```
result = GET_Request("/users/@me");
```

risposta di esempio dal server:

```
{
  "id": "637423922035480852",
  "username": "test",
  "avatar": null,
  "discriminator": "5125",
  "bot": true,
  "email": null,
  "verified": true,
  "locale": "en-US",
  "mfa_enabled": false,
  "flags": 0
}
```

API | OpenAI

L'API OpenAI Realtime abilita interazioni multimodali a bassa latenza, incluse esperienze conversazionali speech-to-speech e trascrizione in tempo reale.

Il componente **TsgcWSAPI_OpenAI** implementa la RealTime OpenAI API.

Configurazione

Usare la proprietà **method** per selezionare **Conversation** o **Transcription**; attualmente è supportata solo la modalità Transcription.

La proprietà **InputAudio** consente di personalizzare i seguenti dati:

- **Language:** esempio inglese (value = 'en').
- **Model:** quale modello verrà utilizzato, esempio: whisper-1
- **Prompt:** prompt facoltativo per fornire istruzioni al modello.

OpenAI

L'API OpenAI utilizza chiavi API per l'autenticazione. Visitare la pagina delle [Chiavi API](#) per recuperare la chiave API da utilizzare nelle richieste.

Si ricordi che la Sua chiave API è un segreto! Non la condivida con altri e non la esponga in alcun codice lato client (browser, app). Le richieste di produzione devono essere instradate attraverso il Suo server backend, dove la chiave API può essere caricata in modo sicuro da una variabile d'ambiente o da un servizio di gestione delle chiavi.

Questa **API Key** deve essere configurata nella proprietà **OpenAIOptions.ApiKey** del componente. Facoltativamente, per gli utenti che appartengono a più organizzazioni, è possibile impostare la propria organizzazione nella proprietà **OpenAIOptions.Organization** se il proprio account appartiene a un'organizzazione.

Una volta configurata la chiave API, di seguito è riportato un elenco delle funzioni disponibili per interagire con l'API OpenAI.

Azure

Il client supporta Microsoft Azure OpenAI Services, pertanto è possibile utilizzare il proprio account Azure per interagire con l'API Azure OpenAI. Per configurare il client affinché funzioni con Azure, seguire i passaggi successivi:

1. Configuri la proprietà **OpenAIOptions.Provider** = oapvAzure
2. Impostare i valori di **ResourceName** e **DeploymentId** (questi valori si trovano nel proprio account Azure)
 1. **OpenAIOptions.AzureOptions.ResourceName** = <nome della risorsa>.
 2. **OpenAIOptions.AzureOptions.DeploymentId** = <id del deployment>.
3. Impostare la API Key del proprio Account Azure
 1. **OpenAIOptions.ApiKey** = <azure api key>.

Esempio di Trascrizione

Di seguito è riportato un esempio di trascrizione in tempo reale tramite l'API openAI

```
var wsClient = new TsgcWebSocketClient(null);
var audioRecorder = new TsgcAudioRecorderWave(null);
var openAI = new TsgcWSAPI_OpenAI(null);

openAI.Client = wsClient;
openAI.AudioRecorder = audioRecorder;
openAI.OpenAIOptions.ApiKey = "your-api-key-here";
openAI.OpenAIOptions.Method = OpenAIMethod.Transcription;
openAI.OpenAIOptions.Provider = OpenAIProvider.OpenAI;
```

```
openAI.InputAudio.Language = "en";  
openAI.InputAudio.Model = "whisper-1";  
openAI.AudioTranscriptionCompleted += OnAudioTranscriptionCompleted;  
  
private void OnAudioTranscriptionCompleted(object sender, TsgcWSOpenAIConversation_Item_Completed aItem)  
{  
    Console.WriteLine("#transcription_completed: " + aItem.Transcript);  
}
```

API MEXC

MEXC è un exchange di criptovalute globale che espone dati di mercato e di account in streaming tramite connessioni WebSocket sicure.

L'API Spot WebSocket segue le specifiche pubblicate nella documentazione [Market Streams](#) e [User Data Streams](#). I canali di mercato vengono consegnati come frame Protocol Buffers mentre i flussi privati usano JSON. Il componente **TsgcWSAPI_MEXC** gestisce il ciclo di vita della connessione, l'autenticazione e la manutenzione della chiave di ascolto in modo che le applicazioni possano concentrarsi sull'elaborazione dei dati inviati.

Oltre allo streaming tramite WebSocket, il toolkit espone anche l'API HTTP Spot completa. Il componente **TsgcHTTP_API_MEXC_Spot** (unità **sgcHTTP_API_MEXC**) incapsula ogni endpoint REST ufficiale in modo che le applicazioni possano interrogare i dati di mercato o operare con richieste firmate.

Configurazione del componente

L'endpoint WebSocket di base è **wss://wbs-api.mexc.com/ws**. Quando **MEXCUserDataStreams.UserStream** è abilitato e viene fornita una chiave API valida, il componente richiede automaticamente una chiave di ascolto e la aggiunge all'URL di connessione.

Inserire un componente **TsgcWebSocketClient** e un componente **TsgcWSAPI_MEXC** nel form (o crearli nel codice), assegnare l'istanza client al componente API e attivare il client per avviare la sessione.

```
TsgcWebSocketClient wsClient = new TsgcWebSocketClient();
TsgcWSAPI_MEXC mexc = new TsgcWSAPI_MEXC();
mexc.Client = wsClient;
mexc.MEXCAPI.ApiKey = "YOUR_KEY";
mexc.MEXCAPI.ApiSecret = "YOUR_SECRET";
mexc.MEXCUserDataStreams.UserStream = true;
if (wsClient.Connect()) {
    mexc.SubscribeTrade("BTCUSDT");
}
```

Proprietà

- **Client:** riferimento al trasporto **TsgcWebSocketClient**. Il componente abilita l'heartbeat del client (15 secondi) per mantenere viva la sessione.
- **MEXCAPI.ApiKey / ApiSecret:** Credenziali API Spot. La chiave è richiesta per i canali privati; il segreto è necessario per firmare le richieste di listen key.
- **MEXCUserDataStreams.UserStream:** quando è **True** (impostazione predefinita), il componente recupera una listen key prima della connessione e la aggiorna ogni 10 minuti. Disabilitarlo per lavorare solo con feed pubblici.
- **MEXCUserDataStreams.ListenKeyOnDisconnect:** azione eseguita quando il WebSocket si disconnette mentre una listen key è attiva:
 - **blkmxodDeleteListenKey:** revoca la listen key tramite REST (predefinito).
 - **blkmxodClearListenKey:** mantiene la listen key lato server e cancella il valore memorizzato nella cache.
 - **blkmxodDoNothing:** riutilizza la stessa listen key alla connessione successiva.
- **ListenKey:** proprietà di sola lettura che espone la listen key attiva restituita dall'API REST.
- **RawMessages:** se abilitato, i frame di testo in arrivo non vengono analizzati e vengono inoltrati come JSON semplice tramite gli eventi WebSocket ereditati.

Eventi

- **OnConnect / OnDisconnect:** attivati quando la sessione WebSocket sottostante viene stabilita o terminata.

- **OnMEXCMarketStream**: attivato per ogni frame di mercato Protobuf. Il gestore riceve un **TsgcMEXCSpotProtoMessage** la cui proprietà **MessageType** identifica il payload (Trade, KLine, DiffDepth, LimitDepth, BookTicker, BookTickerBatch, MiniTicker o MiniTickers). Utilizzare le classi helper in **sgcWebSocket_API_MEXC_Proto** per esaminare la struttura decodificata.
- **OnMEXCResponse**: riceve conferme di iscrizione e risposte ping/pong in formato JSON (**Id**, **Code**, **Message**).
- **OnMEXCUserDataStream**: viene attivato quando arrivano messaggi privati dell'account (ordini, operazioni, aggiornamenti del saldo). Vengono forniti il nome del canale e il payload JSON grezzo.
- **OnMEXCException**: acquisisce le eccezioni REST o WebSocket (errori di listen key, errori HTTP, problemi di analisi, ecc.).

L'esempio seguente mostra come decodificare un frame di trade:

Metodi di sottoscrizione WebSocket

Il componente espone helper per ogni canale Spot pubblico. Richiamare il metodo **Subscribe*** per avviare lo streaming dei dati e il corrispondente **UnSubscribe*** per interromperlo.

Canali pubblici

Metodo	Parametri	Descrizione
SubscribeTrade	Symbol, Interval (ms)	Esecuzioni di scambi aggregati distribuite come Protobuf public-deals .
SubscribeKline	Symbol, Interval	Aggiornamenti candlestick per intervalli configurabili (da 1 minuto a 1 mese).
SubscribeDiffDepth	Symbol, Interval (ms)	Delta di profondità incrementale del book ordini per la gestione del book locale.
SubscribeBookDepth	Symbol, Levels	Istantanee periodiche del libro degli ordini limite con la profondità richiesta (predefinita 5).
SubscribeBookTicker	Symbol, Interval (ms)	Aggiornamenti migliore offerta/domanda per una singola coppia di trading.
SubscribeBookTicker-Batch	flag UTC	Aggiornamenti aggregati del book ticker per più simboli.
SubscribeMiniTickers	flag UTC	Statistiche mini ticker rolling delle ultime 24 ore per tutti gli strumenti attivi.
SubscribeMiniTicker	Symbol, flag UTC	Mini ticker per un singolo simbolo.

Canali privati

I topic privati richiedono una chiave API valida. Il componente effettua la sottoscrizione utilizzando la listen key attiva e la rinnova periodicamente.

Metodo	Descrizione
SubscribeAccountUpdate	Variazioni del saldo e della posizione dell'account.
SubscribeAccountDeals	Report di esecuzione per gli ordini eseguiti.
SubscribeAccountOrders	Aggiornamenti del ciclo di vita degli ordini (nuovo, annullato, rifiutato, ecc.).

Utilizzare i metodi **UnSubscribe*** corrispondenti per terminare uno stream. Il componente invia automaticamente comandi **PING** e convalida i **PONG** restituiti per monitorare la connettività.

Endpoint REST pubblici

Metodi REST disponibili pubblicamente.

Metodo	Descrizione
Ping	Verifica di connettività che chiama /api/v3/ping e restituisce True quando l'exchange risponde con un oggetto vuoto.
GetServerTime	Recupera il timestamp corrente dell'exchange da /api/v3/time .
GetExchangeInformation	Restituisce le regole di trading e i metadati dei simboli da /api/v3/exchangeInfo .
GetOrderBook	Scarica snapshot del libro degli ordini da /api/v3/depth ; impostare il parametro opzionale limit per controllare il numero di livelli (predefinito 100).
GetTrades	Recupera le transazioni pubbliche recenti da /api/v3/trades con un limit opzionale (predefinito 100).
GetHistoricalTrades	Fornisce dati storici sugli scambi con filtri opzionali limit e fromId .
GetAggregateTrades	Restituisce operazioni compresse/aggregate tramite /api/v3/aggTrades supportando filtri opzionali per limit , fromId , startTime e endTime .
GetKlines	Scarica i candlestick da /api/v3/klines supportando i filtri di intervallo temporale e di limite.
GetAveragePrice	Restituisce il prezzo medio ponderato corrente da /api/v3/avgPrice .
Get24hrTicker	Recupera le statistiche ticker delle ultime 24 ore per uno o tutti i simboli tramite /api/v3/ticker/24hr .
GetPriceTicker	Recupera il prezzo più recente tramite /api/v3/ticker/price ; passare un simbolo vuoto per recuperare i prezzi di tutte le coppie di trading.
GetBookTicker	Ottiene le migliori quotazioni bid/ask da /api/v3/ticker/bookTicker con supporto per richieste su singolo simbolo o sull'intero exchange.

Endpoint REST privati

Gli endpoint privati richiedono chiavi API valide e una stringa di query firmata. Il componente aggiunge automaticamente il timestamp, **recvWindow** e la firma **HMAC**.

Metodo	Descrizione
GetAccountInformation	Restituisce saldi e permessi da /api/v3/account .
GetOpenOrders	Elenca gli ordini aperti correnti (facoltativamente filtrati per simbolo).
GetAllOrders	Recupera gli ordini storici con filtri opzionali di tempo e limite.
GetOrder	Interroga lo stato di un ordine specifico fornendo orderId o origClientId .
GetMyTrades	Elenca le esecuzioni di scambi private con filtri opzionali limit , fromId , startTime e endTime .
NewOrder	Inserisce un ordine live su /api/v3/order (market, limit, stop, ecc.) con parametri opzionali timeInForce , quantity , quoteOrderQty , price , newClientOrderId , stopPrice , icebergQty e parametri aggiuntivi.
TestNewOrder	Invia una richiesta di sola validazione a /api/v3/order/test accettando lo stesso insieme di parametri di NewOrder .
CancelOrder	Annulla un ordine specifico su /api/v3/order utilizzando orderId o origClientId .

CancelAllOrders	Annulla in blocco tutti gli ordini aperti per un simbolo tramite /api/v3/openOrders .
GetSubAccounts	Elenca i sotto-account gestiti (/api/v3/sub-account/list).
GetSubAccountAssets	Restituisce i saldi di un sotto-account specifico.
TransferSubAccount	Trasferisce asset tra sub-account tramite /api/v3/sub-account/transfer ; fornire l'importo e opzionalmente il tipo di trasferimento.
GetDepositAddress	Richiede gli indirizzi di deposito con un filtro opzionale per network .
GetDepositHistory	Recupera i record di deposito con filtri opzionali per coin , status , startTime e endTime .
GetWithdrawHistory	Restituisce la cronologia dei prelievi filtrata per coin , status e intervallo di tempo opzionale.
Withdraw	Invia una richiesta di prelievo tramite /api/v3/capital/withdraw/apply includendo i parametri obbligatori coin , address e amount oltre ai parametri opzionali network , addressTag , withdrawOrderId e parametri aggiuntivi.
BatchOrders	Invia ordini batch per più simboli in una singola richiesta.
GetTradeFee	Ottieni la commissione di trading per un simbolo specifico.
GetDepositHistory	Recupera i record di deposito con filtri opzionali per coin , status , startTime e endTime .
CancelWithdraw	Annulla una richiesta di prelievo in sospeso tramite l'ID del prelievo.
CreateInternalTransfer	Trasferisce internamente asset tra account.
GetTransferHistory	Ottieni la cronologia dei trasferimenti interni tra account.

API MEXC Futures

I contratti perpetui e a consegna di [MEXC](#) espongono una API di streaming dedicata, documentata nella [specifica ufficiale Futures WebSocket](#). Il componente **TsgcWSAPI_MEXC_Futures** incapsula la connessione, l'handshake di login e la gestione dei topic necessari per consumare dati sui derivati in tempo reale.

MEXC mantiene anche un'interfaccia REST per il trading di derivati. Il componente **TsgcHTTP_API_MEXC_Futures** contenuto in **sgcHTTP_API_MEXC** integra i feed WebSocket con helper HTTP che corrispondono uno a uno agli endpoint ufficiali.

Configurazione del componente

L'endpoint WebSocket Futures è **wss://contract.mexc.com/edge**. I messaggi sono codificati come oggetti JSON e la piattaforma richiede frame **ping/pong** periodici che vengono gestiti automaticamente dal componente.

Per ricevere notifiche private (dati di ordini e account) è necessario autenticarsi. Impostare **MEXCAPI.ApiKey** e **MEXCAPI.ApiSecret** prima di attivare il client WebSocket. Quando le credenziali sono presenti, il componente firma una richiesta di accesso (HMAC SHA256) non appena il socket si connette.

```
TsgcWebSocketClient wsClient = new TsgcWebSocketClient();
TsgcWSAPI_MEXC_Futures futures = new TsgcWSAPI_MEXC_Futures();
futures.Client = wsClient;
futures.MEXCAPI.ApiKey = "YOUR_KEY";
futures.MEXCAPI.ApiSecret = "YOUR_SECRET";
if (wsClient.Connect()) {
    futures.SubscribeDepth("BTC_USDT", true);
}
```

Proprietà

- **Client:** trasporto WebSocket utilizzato dal componente API. Il supporto heartbeat è abilitato per mantenere attivo il socket.
- **MEXCAPI.ApiKey / ApiSecret:** necessari per firmare la richiesta di login. Senza credenziali sono accessibili solo i canali pubblici.
- **RawMessages:** quando abilitato, bypassa il parser JSON in modo che i frame di testo non elaborati siano esposti tramite gli eventi base.

Eventi

- **OnConnect / OnDisconnect:** Notifiche del ciclo di vita WebSocket.
- **OnMEXCLogin:** generato dopo un accesso autenticato con successo.
- **OnMEXCSubscribed / OnMEXCUnsubscribed:** conferma delle modifiche alla sottoscrizione. L'evento espone il nome del canale restituito da MEXC.
- **OnMEXCMessage:** recapita ogni aggiornamento di mercato o di account insieme all'identificatore del canale.
- **OnMEXCError:** viene generato quando il server risponde con un payload **rs.error** (parametri non validi, errore di autenticazione, ecc.).
- **OnMEXCException:** segnala le eccezioni locali generate durante l'elaborazione dei dati JSON o l'esecuzione di operazioni di rete.

Il seguente gestore registra le conferme di iscrizione e stampa gli snapshot di profondità in arrivo:

Metodi di sottoscrizione WebSocket

Ogni helper racchiude una richiesta **sub/unsub** come descritto dall'API ufficiale. Usare il metodo **UnSubscribe*** corrispondente per annullare lo stream.

Metodo	Parametri	Descrizione
SubscribeDeal	Symbol	Operazioni eseguite sul contratto (canale: deal).
SubscribeTickers	-	Statistiche globali del ticker per tutti i contratti.
SubscribeTicker	Symbol	Riepilogo ticker per un singolo strumento.
SubscribeDepth	Symbol, Compress	Aggiornamenti incrementali del libro degli ordini
SubscribeDepthFull	Symbol, Level	Snapshot di profondità completa con numero di livelli configurabile (predefinito 20).
SubscribeKline	Symbol, Interval	Dati candlestick per i timeframe supportati (Min1, Min5, Min15, Min30, Min60, Hour4, Hour12, Day1, Week1, Month1).
SubscribeFundingRate	Symbol	Ultimi annunci sul tasso di finanziamento.
SubscribeIndexPrice	Symbol	Flusso del prezzo dell'indice sottostante.
SubscribeFairPrice	Symbol	Aggiornamenti del prezzo Mark (equo) inviati dall'exchange.

Quando il server riconosce una richiesta, viene attivato l'evento **OnMEXCSubscribed** con il nome del canale (ad esempio **rs.sub.depth.BTC_USDT**). Gli errori vengono inoltrati tramite **OnMEXCError**, incluso il messaggio del server per la risoluzione dei problemi.

Canali privati

I canali privati richiedono una chiave API valida. Il componente esegue automaticamente l'autenticazione quando vengono fornite le credenziali.

Metodo	Descrizione
SubscribePersonalOrder	Aggiornamenti personali sugli ordini.
SubscribePersonalOrderDeal	Aggiornamenti delle operazioni (fill) degli ordini personali.
SubscribePersonalPosition	Aggiornamenti della posizione personale.
SubscribePersonalPlanOrder	Aggiornamenti degli ordini del piano personale (trigger).
SubscribePersonalStopOrder	Aggiornamenti degli ordini stop personali.
SubscribePersonalStopPlanOrder	Aggiornamenti degli ordini stop personali.
SubscribePersonalRiskLimit	Aggiornamenti del limite di rischio personale.
SubscribePersonalADLLevel	Aggiornamenti del livello ADL (auto-deleveraging) personale.
SubscribePersonalAsset	Aggiornamenti degli asset personali.

Endpoint REST pubblici

Metodi REST disponibili pubblicamente.

Metodo	Descrizione
GetPing	Verifica la raggiungibilità dell'API tramite /api/v1/ping .
GetServerTime	Restituisce il timestamp del server da /api/v1/time .

GetContracts	Fornisce l'elenco dei contratti disponibili (/api/v1/contract/detail).
GetDepth	Scarica la profondità del book degli ordini da /api/v1/contract/depth ; utilizzare il parametro opzionale limit (predefinito 50) per selezionare il numero di livelli.
GetDeals	Recupera le transazioni recenti tramite /api/v1/contract/deals con un limit facoltativo (predefinito 100).
GetKlines	Restituisce dati candlestick tramite /api/v1/contract/kline con filtri facoltativi startTime , endTime e limit (default 200).
GetIndexPrice	Recupera il prezzo dell'indice sottostante per il simbolo richiesto (/api/v1/contract/indexPrice).
GetFairPrice	Restituisce il prezzo equo (mark) per un contratto da /api/v1/contract/fairPrice .
GetFundingRate	Riporta l'ultimo tasso di finanziamento per un simbolo utilizzando /api/v1/contract/fundingRate .
GetAllTickers	Ottieni tutti i ticker dei contratti futures con riepiloghi dei prezzi in tempo reale.
GetFundingRateHistory	Recupera i record storici del tasso di finanziamento per un contratto.
GetFairPriceKline	Recupera i dati kline/candlestick del prezzo equo (mark price) per un contratto.
GetIndexPriceKline	Recupera i dati kline/candlestick del prezzo indice per un contratto.

Endpoint REST privati

Gli endpoint derivati privati richiedono credenziali API. Il componente firma automaticamente ogni richiesta con il timestamp e il recvWindow.

Metodo	Descrizione
GetAccountAssets	Restituisce i saldi del margine da /api/v1/private/account/assets .
GetPositionList	Elenca le posizioni correnti tramite /api/v1/private/position/list , opzionalmente filtrate per simbolo.
SetPositionLeverage	Aggiorna la leva per un simbolo e, se fornita, la modalità di margine.
PlaceOrder	Inserisce un ordine futures su /api/v1/private/order fornendo symbol , side , positionSide , type e quantity più i parametri opzionali price , clientOrderId e altri parametri aggiuntivi.
CancelOrder	Annulla un ordine specifico utilizzando /api/v1/private/order/cancel tramite orderId o clientOrderId .
CancelAllOrders	Annulla tutti gli ordini aperti per un symbol tramite /api/v1/private/order/cancel-all .
GetOpenOrders	Elenca gli ordini aperti correnti tramite /api/v1/private/order/list/open con un filtro symbol opzionale.
GetOrderHistory	Recupera la cronologia degli ordini (/api/v1/private/order/list/history) con parametri facoltativi di symbol, intervallo temporale e limit.
GetFundingHistory	Restituisce i pagamenti di funding passati da /api/v1/private/account/funding con filtri opzionali startTime , endTime e limit .
GetOpenPositions	Ottieni tutte le posizioni attualmente aperte.
ChangeMargin	Modifica l'importo del margine per una posizione.
GetPositionMode	Ottenere la modalità di posizione corrente (modalità unidirezionale o hedge).

ChangePositionMode	Cambia la modalità di posizione tra modalità unidirezionale e modalità hedge.
PlaceBatchOrder	Invia più ordini futures in un'unica richiesta batch.
GetOrderDetail	Ottiene informazioni dettagliate per un ordine specifico.
GetOrderDealDetails	Ottieni i dettagli di esecuzione/operazione per un ordine specifico.
PlaceTriggerOrder	Inserire un ordine trigger (piano) che viene eseguito quando le condizioni sono soddisfatte.
CancelTriggerOrder	Annulla uno specifico ordine trigger.
CancelAllTriggerOrders	Annulla tutti gli ordini trigger aperti.
GetTriggerOrders	Ottieni l'elenco degli ordini trigger (plan).
GetStopOrders	Ottieni l'elenco degli ordini stop.
CancelStopOrder	Annullare uno specifico ordine stop.
CancelAllStopOrders	Annulla tutti gli ordini stop aperti.

API Bitget

Bitget

API supportate

- **WebSocket API:** connette a un server websocket e fornisce aggiornamenti in tempo reale dei dati di mercato, variazioni dell'account e altro.
- **REST API:** invia richieste HTTP per ottenere dati di mercato, piazzare ordini, dati dell'account...

Proprietà

Utilizzare la proprietà **BitgetChannel** per selezionare il tipo di canale:

- **bgcSpot:** connette all'API WebSocket Spot.
- **bgcFutures:** connessi all'API WebSocket Futures.

Può configurare le seguenti proprietà nella proprietà Bitget.

- **ApiKey:** può richiedere una nuova chiave api nel suo account Bitget, basta copiare il valore in questa proprietà. Se l'ApiKey è impostata, il client si connetterà al server privato websocket. Se è vuota, si connetterà al server pubblico WebSocket.
- **ApiSecret:** è il valore segreto dell'api.
- **Passphrase:** è la stringa personalizzata definita durante la creazione di una nuova chiave API.

Connessione

Quando il client si connette con successo ai server Bitget, viene generato l'evento **OnConnect**. Dopo che l'evento **OnConnect** è stato generato, è possibile iniziare a **inviare e ricevere messaggi** da/verso i server Bitget. Se ci si connette al canale WebSocket privato, è necessario attendere che l'evento **OnBitgetAuthentication** venga generato e verificare che il parametro success sia true, prima di sottoscrivere qualsiasi canale.

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWS_API_Bitget oBitget = new TsgcWS_API_Bitget();
oBitget.Client = oClient;
oBitget.Bitget.ApiKey = "your_api_key";
oBitget.Bitget.ApiSecret = "your_api_secret";
oBitget.Bitget.Passphrase = "your_passphrase";
oBitget.BitgetChannel = bgcSpot;
oClient.Active = true;
void OnConnect(TsgcWSConnection Connection)
{
    DoLog("#Bitget Connected");
}
```

Eventi

Il client Bitget implementa i seguenti eventi per controllare il flusso di connessione e ottenere i dati inviati dal server WebSocket:

- **OnConnect:** attivato quando il client websocket si connette al server WebSocket.
- **OnDisconnect:** attivato quando il client websocket si disconnette dal WebSocket Server.
- **OnBitgetAuthentication:** attivato quando il client si autentica sul Private WebSocket Server.
- **OnBitgetSubscribe:** quando il client si sottoscrive a un canale websocket.
- **OnBitgetUnSubscribe:** quando il client annulla la sottoscrizione a un canale websocket.
- **OnBitgetData:** quando il client riceve dati dal server.
- **OnBitgetError:** quando si verifica un errore durante la connessione websocket Bitget.
- **OnBitgetHTTPException:** viene generato quando si verifica un errore durante la richiesta REST.

WebSocket API

Il feed websocket fornisce aggiornamenti in tempo reale sui dati di mercato per ordini e transazioni. Può sottoscrivere i seguenti **canali**:

Method	Pubblico o Privato	Description
SubscribeTicker	Public	Sottoscrive lo stream ticker per uno strumento.
SubscribeTrade	Public	Sottoscrivi allo stream degli scambi recenti.
SubscribeCandle	Public	Sottoscrivi al flusso candlestick. Supporta più intervalli (es. candle1m, candle5m, candle1H).
SubscribeOrderBook	Public	Sottoscrivi al flusso dell'order book. Supporta diversi livelli di profondità.
SubscribeOrders	Privato	Sottoscrivi agli aggiornamenti degli ordini. Richiede autenticazione.
SubscribePositions	Privato	Sottoscrivi agli aggiornamenti delle posizioni. Richiede autenticazione.
SubscribeAccount	Privato	Sottoscrive gli aggiornamenti del saldo dell'account. Richiede l'autenticazione.

Di seguito è riportato un esempio di sottoscrizione ai canali WebSocket privati dopo un'autenticazione riuscita.

```
void OnBitgetAuthentication(TObject Sender, bool aSuccess, string aError, string aRawMessage)
{
    if (aSuccess == true)
    {
        oBitget.SubscribeOrders("BTCUSDT");
        oBitget.SubscribeAccount();
    }
}
```

REST API

La REST API fornisce metodi pubblici e privati per richiedere dati dai mercati e dagli account privati. Acceda alla REST API tramite la proprietà **REST_API** del componente.

Method	Public / Private	Description
GetServerTime	Public	Ottiene l'ora del server.
GetTickers	Public	Ottieni informazioni sul ticker per uno o tutti i simboli.
GetOrderBook	Public	Ottieni il book degli ordini per un simbolo.
GetCandles	Public	Recupera i dati candlestick/kline per un simbolo.
GetRecentTrades	Public	Ottiene le operazioni più recenti per un symbol.
PlaceOrder	Privato	Inserire un nuovo ordine.
CancelOrder	Privato	Annula un ordine esistente.
GetOpenOrders	Privato	Ottenere l'elenco degli ordini aperti.
GetOrderDetail	Privato	Ottiene i dettagli di un ordine specifico.
GetOrderHistory	Privato	Recupera la cronologia degli ordini.
GetAccountAssets	Privato	Ottieni informazioni sulle risorse dell'account.
GetAccountInfo	Privato	Recupera le informazioni sull'account.

Di seguito è riportato un esempio di utilizzo della REST API.

```
TsgcWS_API_Bitget oBitget = new TsgcWS_API_Bitget();
oBitget.Bitget.ApiKey = "your_api_key";
oBitget.Bitget.ApiSecret = "your_api_secret";
oBitget.Bitget.Passphrase = "your_passphrase";
// Get tickers
oBitget.REST_API.GetTickers("SPOT");
// Place an order
oBitget.REST_API.PlaceOrder("BTCUSDT", "buy", "market", "normal", "0.01");
```

API GateIO

[Gate.io](https://gate.io)

API supportate

- **WebSocket API:** connette a un server websocket e fornisce aggiornamenti in tempo reale dei dati di mercato, variazioni dell'account e altro.
- **REST API:** invia richieste HTTP per ottenere dati di mercato, piazzare ordini, dati dell'account...

Proprietà

Utilizzare la proprietà **GateIOChannel** per selezionare il tipo di canale:

- **giocSpot:** connette all'API WebSocket Spot.
- **giocFutures:** connettersi all'API WebSocket Futures.

È possibile configurare le seguenti proprietà nella proprietà GateIO.

- **ApiKey:** è possibile richiedere una nuova chiave API nel proprio account Gate.io; copiare il valore in questa proprietà. Se ApiKey è impostato, il client si autenticherà sul server privato. Se è vuoto, saranno disponibili solo i canali pubblici.
- **ApiSecret:** è il valore segreto dell'api.

Connessione

Quando il client si connette con successo ai server Gate.io, viene generato l'evento **OnConnect**. Dopo che l'evento **OnConnect** è stato generato, è possibile iniziare a **inviare e ricevere messaggi** da/verso i server Gate.io. Se ci si connette al canale WebSocket privato, è necessario attendere che l'evento **OnGateIOAuthentication** venga generato e verificare che il parametro success sia true, prima di sottoscrivere qualsiasi canale privato.

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWS_API_GateIO oGateIO = new TsgcWS_API_GateIO();
oGateIO.Client = oClient;
oGateIO.GateIO.ApiKey = "your_api_key";
oGateIO.GateIO.ApiSecret = "your_api_secret";
oGateIO.GateIOChannel = giocSpot;
oClient.Active = true;
void OnConnect(TsgcWSConnection Connection)
{
    DoLog("#GateIO Connected");
}
```

Eventi

Il client GateIO implementa i seguenti eventi per controllare il flusso di connessione e ottenere i dati inviati dal server WebSocket:

- **OnConnect:** attivato quando il client websocket si connette al server WebSocket.
- **OnDisconnect:** attivato quando il client websocket si disconnette dal WebSocket Server.
- **OnGateIOAuthentication:** attivato quando il client si autentica con il Private WebSocket Server.
- **OnGateIOSubscribe:** quando il client si sottoscrive a un canale websocket.
- **OnGateIOUnSubscribe:** quando il client annulla la sottoscrizione a un canale websocket.
- **OnGateIOData:** quando il client riceve dati dal server.
- **OnGateIOError:** quando si verifica un errore durante la connessione websocket a GateIO.
- **OnGateIOHTTPException:** in caso di errore durante la richiesta REST.

WebSocket API

Il feed websocket fornisce aggiornamenti in tempo reale sui dati di mercato per ordini e transazioni. Può sottoscrivere i seguenti **canali**:

Method	Pubblico o Privato	Description
SubscribeTicker	Public	Iscrizione allo stream ticker per un simbolo.
SubscribeTrades	Public	Sottoscrivi allo stream degli scambi recenti.
SubscribeCandlesticks	Public	Si iscriva allo stream candlestick. Supporta più intervalli (per esempio 1m, 5m, 15m, 30m, 1h).
SubscribeOrderBook	Public	Abbonati al flusso del libro degli ordini. Supporta livelli di profondità e intervalli di aggiornamento configurabili.
SubscribeOrders	Privato	Sottoscrivi agli aggiornamenti degli ordini. Richiede autenticazione.
SubscribeBalances	Privato	Sottoscrivere gli aggiornamenti del saldo. Richiede l'autenticazione.

Di seguito è riportato un esempio di sottoscrizione a canali WebSocket pubblici.

```
void OnConnect(TsgcWSConnection Connection)
{
    oGateIO.SubscribeTicker("BTC_USDT");
    oGateIO.SubscribeOrderBook("BTC_USDT", "20", "100ms");
}
```

REST API

La REST API fornisce metodi pubblici e privati per richiedere dati dai mercati e dagli account privati. Acceda alla REST API tramite la proprietà **REST_API** del componente.

Method	Public / Private	Description
GetCurrencyPairs	Public	Ottiene informazioni sulle coppie di valute disponibili.
GetTickers	Public	Ottieni informazioni sul ticker per una o tutte le coppie di valute.
GetOrderBook	Public	Ottiene il book degli ordini per una coppia di valute.
GetCandlesticks	Public	Ottenere i dati candlestick/kline per una coppia di valute.
GetTrades	Public	Ottieni gli scambi più recenti per una coppia di valute.
PlaceOrder	Privato	Inserisce un nuovo ordine. Supporta ordini limite e di mercato.
CancelOrder	Privato	Annulla un ordine esistente.
GetOrder	Privato	Ottiene i dettagli di un ordine specifico.
GetOpenOrders	Privato	Ottenere l'elenco degli ordini aperti.
GetSpotAccounts	Privato	Ottiene le informazioni sull'account spot.

Di seguito è riportato un esempio di utilizzo della REST API.

```
TsgcWS_API_GateIO oGateIO = new TsgcWS_API_GateIO();
oGateIO.GateIO.ApiKey = "your_api_key";
oGateIO.GateIO.ApiSecret = "your_api_secret";
// Get tickers
oGateIO.REST_API.GetTickers("BTC_USDT");
// Place an order
oGateIO.REST_API.PlaceOrder("BTC_USDT", "buy", "0.001", "50000", "limit");
```

API Deribit

Deribit

Deribit è una piattaforma di scambio di derivati in criptovaluta che offre il trading di futures e opzioni su Bitcoin ed Ethereum.

API supportate

- **WebSocket API:** connette a un server websocket e fornisce aggiornamenti in tempo reale dei dati di mercato, variazioni dell'account e altro.
- **REST API:** invia richieste HTTP per ottenere dati di mercato, piazzare ordini, dati dell'account...

Proprietà

È possibile configurare le seguenti proprietà nella proprietà Deribit.

- **ApiKey:** è possibile richiedere una nuova chiave API nel proprio account Deribit; copiare il valore in questa proprietà. Se APIKey è impostata, il client si autenticcherà contro il server privato. Se è vuota, saranno disponibili solo i canali pubblici.
- **ApiSecret:** è il valore segreto dell'api.
- **TestNet:** se abilitato, si conatterà all'ambiente Deribit TestNet (disabilitato per impostazione predefinita). Utile per i test senza rischiare fondi reali.

Connessione

Quando il client si connette correttamente ai server Deribit, viene generato l'evento **OnConnect**. Dopo che l'evento **OnConnect** è stato generato, è possibile iniziare a **inviare e ricevere messaggi** da/verso i server Deribit. Se si connette al canale websocket privato, deve attendere finché non viene generato l'evento **OnDeribitAuthentication** e verificare se il parametro success è true, prima di sottoscrivere qualsiasi canale privato.

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWS_API_Deribit oDeribit = new TsgcWS_API_Deribit();
oDeribit.Client = oClient;
oDeribit.Deribit.ApiKey = "your_api_key";
oDeribit.Deribit.ApiSecret = "your_api_secret";
oDeribit.Deribit.TestNet = false;
oClient.Active = true;
void OnConnect(TsgcWSConnection Connection)
{
    DoLog("#Deribit Connected");
}
```

Eventi

Il client Deribit implementa i seguenti eventi per controllare il flusso di connessione e ottenere i dati inviati dal server WebSocket:

- **OnConnect:** attivato quando il client websocket si connette al server WebSocket.
- **OnDisconnect:** attivato quando il client websocket si disconnette dal WebSocket Server.
- **OnDeribitAuthentication:** si attiva quando il client si autentica contro il Server WebSocket Privato.
- **OnDeribitSubscribe:** quando il client si iscrive a un canale websocket.
- **OnDeribitUnSubscribe:** quando il client annulla la sottoscrizione a un canale WebSocket.
- **OnDeribitData:** quando il client riceve dati dal server.
- **OnDeribitError:** quando si verifica un errore durante la connessione WebSocket a Deribit.
- **OnDeribitHTTPException:** quando si verifica un errore durante la richiesta REST.

WebSocket API

Il feed websocket fornisce aggiornamenti in tempo reale sui dati di mercato per ordini e transazioni.

È possibile sottoscrivere ai seguenti **canali pubblici**:

Method	Description
SubscribeTicker	Iscrizione al flusso ticker per uno strumento. Fornisce dati in tempo reale su prezzi e volumi.
SubscribeTrades	Si iscrive allo stream dei trade recenti per uno strumento.
SubscribeOrderBook	Iscrizione allo stream order book. Supporta raggruppamento configurabile, livelli di profondità e intervalli di aggiornamento.
SubscribeCandle	Sottoscrivi allo stream candlestick. Supporta più risoluzioni (ad es. 1, 3, 5, 10, 15, 30, 60, 120, 180, 360, 720, 1D).
SubscribePerpetual	Iscriviti ai dati del contratto perpetuo per uno strumento. Fornisce il tasso di finanziamento e altre informazioni specifiche dei contratti perpetui.
SubscribeBookChange	Sottoscrivi alle notifiche di variazione del book ordini per uno strumento.

È possibile sottoscrivere i seguenti **canali privati** (richiedono autenticazione):

Method	Description
SubscribeOrders	Sottoscrizione agli aggiornamenti degli ordini per uno strumento.
SubscribePositions	Sottoscrive gli aggiornamenti delle posizioni. Supporta il filtraggio per valuta e tipo di strumento.
SubscribeAccount-Changes	Sottoscrivi alle variazioni del saldo del conto per una valuta.
SubscribeUserTrades	Abbonati agli aggiornamenti sulle operazioni dell'utente per uno strumento.

Di seguito un esempio di sottoscrizione ai canali websocket.

```
void OnDeribitAuthentication(TObject Sender, bool aSuccess, string aError, string aRawMessage)
{
    if (aSuccess == true)
    {
        oDeribit.SubscribeOrders("BTC-PERPETUAL");
        oDeribit.SubscribePositions("BTC");
    }
}
```

REST API

La REST API fornisce metodi pubblici e privati per richiedere dati da mercati, trading, account e portafoglio. Accedere alla REST API tramite la proprietà **REST_API** del componente.

Dati di mercato (Pubblici)

Method	Description
GetInstruments	Ottieni gli strumenti disponibili per una valuta.
GetTicker	Ottiene le informazioni sul ticker per uno strumento.
GetOrderBook	Ottiene l'order book per uno strumento.
GetTrades	Recupera le operazioni più recenti per uno strumento.
GetCurrencies	Ottieni l'elenco delle valute supportate.
GetIndexPrice	Ottiene il prezzo dell'indice corrente per un nome di indice.
GetFundingRateHistory	Recupera la cronologia del tasso di finanziamento per uno strumento.
GetFundingRateValue	Ottenere il valore del tasso di finanziamento corrente per uno strumento.
GetBookSummaryBy-Currency	Ottieni il riepilogo del book per tutti gli strumenti di una valuta.
GetBookSummaryBy-Instrument	Recupera il riepilogo del libro degli ordini per uno strumento specifico.

Trading (Privato)

Method	Description
Acquisto	Effettua un ordine di acquisto. Supporta i tipi di ordine a mercato e a limite.
Vendita	Inserisce un ordine di vendita. Supporta i tipi di ordine market e limit.
CancelOrder	Annulla un ordine esistente tramite l'ID ordine.
CancelAllOrders	Annulla tutti gli ordini aperti.
CancelAllByInstrument	Annulla tutti gli ordini aperti per uno strumento specifico.
EditOrder	Modifica un ordine esistente (cambia importo o prezzo).
GetOpenOrders	Ottieni l'elenco degli ordini aperti per una valuta.
GetOrderState	Ottieni lo stato di un ordine specifico.
GetOrderHistory	Ottenere la cronologia degli ordini per una valuta.

Account (Privato)

Method	Description
GetPositions	Ottieni le posizioni per una valuta.
GetAccountSummary	Recupera il riepilogo del conto per una valuta.
GetSubAccounts	Ottiene l'elenco dei sotto-account.
GetTransactionLog	Recupera il registro delle transazioni per una valuta.

Wallet (Private)

Method	Description
GetDeposits	Ottieni la cronologia dei depositi per una valuta.
GetWithdrawals	Ottiene lo storico dei prelievi per una valuta.
GetTransfers	Ottiene la cronologia dei trasferimenti per una valuta.

Di seguito è riportato un esempio di utilizzo della REST API.

```
TsgcWS_API_Deribit oDeribit = new TsgcWS_API_Deribit();
oDeribit.Deribit.ApiKey = "your_api_key";
oDeribit.Deribit.ApiSecret = "your_api_secret";
// Get ticker
oDeribit.REST_API.GetTicker("BTC-PERPETUAL");
// Place a buy order
oDeribit.REST_API.Buy("BTC-PERPETUAL", 10, "market");
// Get account summary
oDeribit.REST_API.GetAccountSummary("BTC");
```

API Crypto.com

[Crypto.com](#)

API supportate

- **WebSocket API:** connette a un server websocket e fornisce aggiornamenti in tempo reale dei dati di mercato, variazioni dell'account e altro.
- **REST API:** invia richieste HTTP per ottenere dati di mercato, piazzare ordini, dati dell'account...

Proprietà

Utilizzare la proprietà **Channel** per selezionare il tipo di canale:

- **cccMarket:** connette alla Market WebSocket API per i dati di mercato pubblici.
- **cccUser:** si connette all'API WebSocket User per i dati privati dell'account (richiede autenticazione).

È possibile configurare le seguenti proprietà nella proprietà CryptoCom.

- **ApiKey:** può richiedere una nuova chiave api nel Suo account Crypto.com, copi semplicemente il valore in questa proprietà. Richiesta per i canali privati e le chiamate REST API autenticate.
- **ApiSecret:** è il valore segreto dell'api.

Connessione

Quando il client si connette con successo ai server Crypto.com, viene attivato l'evento **OnConnect**. Dopo che l'evento **OnConnect** è stato attivato, è possibile iniziare a **inviare e ricevere messaggi** da/verso i server Crypto.com. Se ci si connette al canale Utente, è necessario attendere che l'evento **OnCryptoComAuthentication** venga attivato e verificare che il parametro success sia true, prima di iscriversi a qualsiasi canale privato.

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWS_API_CryptoCom oCryptoCom = new TsgcWS_API_CryptoCom();
oCryptoCom.Client = oClient;
oCryptoCom.CryptoCom.ApiKey = "your_api_key";
oCryptoCom.CryptoCom.ApiSecret = "your_api_secret";
oCryptoCom.Channel = cccMarket;
oClient.Active = true;
void OnConnect(TsgcWSConnection Connection)
{
    DoLog("#CryptoCom Connected");
}
```

Eventi

Il client Crypto.com implementa i seguenti eventi per controllare il flusso di connessione e ottenere i dati inviati dal server WebSocket:

- **OnConnect:** attivato quando il client websocket si connette al server WebSocket.
- **OnDisconnect:** attivato quando il client websocket si disconnette dal WebSocket Server.
- **OnCryptoComAuthentication:** generato quando il client si autentica con il Server WebSocket Privato.
- **OnCryptoComSubscribe:** quando il client si iscrive a un canale websocket.
- **OnCryptoComUnSubscribe:** quando il client annulla la sottoscrizione a un canale websocket.
- **OnCryptoComData:** quando il client riceve dati dal server.
- **OnCryptoComError:** quando si verifica un errore durante la connessione websocket a Crypto.com.
- **OnCryptoComHTTPException:** quando si verifica un errore durante la richiesta REST.

WebSocket API

Il feed websocket fornisce aggiornamenti in tempo reale sui dati di mercato per ordini e transazioni.

È possibile sottoscrivere ai seguenti **canali di mercato** (pubblici):

Method	Description

SubscribeTicker	Iscrizione al flusso ticker per uno strumento. Fornisce dati in tempo reale su prezzi e volumi.
SubscribeTrade	Si iscrive allo stream dei trade recenti per uno strumento.
SubscribeCandlestick	Sottoscrivi al flusso di candele. Supporta più intervalli (ad es. 5m, 15m, 30m, 1h, 4h, 1D).
SubscribeBook	Sottoscrivi al flusso dell'order book. Supporta livelli di profondità configurabili (ad es. 10, 50).

È possibile sottoscrivere ai seguenti **canali utente** (privati, richiedono autenticazione):

Method	Description
SubscribeOrders	Iscriviti agli aggiornamenti degli ordini. Filtrare facoltativamente per nome strumento.
SubscribeUserTrades	Sottoscrivere gli aggiornamenti sulle operazioni dell'utente. Facoltativamente filtrare per nome strumento.
SubscribeBalance	Sottoscrivi gli aggiornamenti del saldo per il tuo account.

Di seguito è riportato un esempio di iscrizione ai canali di mercato.

```
void OnConnect(TsgcWSConnection Connection)
{
    oCryptoCom.SubscribeTicker("BTC_USDT");
    oCryptoCom.SubscribeBook("BTC_USDT", "10");
}
```

REST API

La REST API fornisce metodi pubblici e privati per richiedere dati dai mercati e dagli account privati. Acceda alla REST API tramite la proprietà **REST_API** del componente.

Method	Public / Private	Description
GetInstruments	Public	Recupera l'elenco degli strumenti disponibili.
GetTicker	Public	Ottieni informazioni sul ticker per uno o tutti gli strumenti.
GetOrderBook	Public	Ottiene l'order book per uno strumento.
GetTrades	Public	Recupera le operazioni più recenti per uno strumento.
GetCandlestick	Public	Recupera i dati candlestick/kline per uno strumento.
CreateOrder	Privato	Crea un nuovo ordine. Supporta gli ordini limit e market.
CancelOrder	Privato	Annula un ordine esistente.
CancelAllOrders	Privato	Annula tutti gli ordini aperti per uno strumento.
GetOpenOrders	Privato	Ottenere l'elenco degli ordini aperti.
GetOrderDetail	Privato	Ottiene i dettagli di un ordine specifico.
GetOrderHistory	Privato	Recupera la cronologia degli ordini.
GetAccountSummary	Privato	Ottiene le informazioni di riepilogo del conto.

Di seguito è riportato un esempio di utilizzo della REST API.

```
TsgcWS_API_CryptoCom oCryptoCom = new TsgcWS_API_CryptoCom();
oCryptoCom.CryptoCom.ApiKey = "your_api_key";
oCryptoCom.CryptoCom.ApiSecret = "your_api_secret";
// Get instruments
oCryptoCom.REST_API.GetInstruments();
// Create an order
oCryptoCom.REST_API.CreateOrder("BTC_USDT", "BUY", "MARKET", "", "0.001");
// Get account summary
oCryptoCom.REST_API.GetAccountSummary();
```


API HTX

HTX

HTX (precedentemente Huobi) è un exchange di criptovalute internazionale e multilingue. Il componente API HTX fornisce una REST API HTTP a complemento dell'esistente [Huobi WebSocket API](#).

API supportate

- **WebSocket API:** connette a un server websocket e fornisce aggiornamenti in tempo reale sui dati di mercato, variazioni dell'account e altro ancora. Consultare la documentazione [API Huobi](#) per i metodi di sottoscrizione WebSocket.
- **REST API:** invia richieste HTTP per ottenere dati di mercato, piazzare ordini, dati dell'account...

Proprietà

È possibile configurare le seguenti proprietà nella proprietà Huobi (per WebSocket) o nella proprietà HTXOptions (per REST API).

- **ApiKey:** è possibile richiedere una nuova chiave API nel proprio account HTX; è sufficiente copiare il valore in questa proprietà. Se ApiKey è impostato, il client si conatterà al server websocket privato. Se è vuoto, si conatterà al server WebSocket pubblico.
- **ApiSecret:** è il valore segreto dell'api.

Connessione

Il componente HTX estende il client WebSocket API Huobi. Quando il client si connette con successo ai server HTX, viene generato l'evento **OnConnect**. Se ApiKey non è vuoto, il client tenterà di connettersi al server websocket privato, quindi saranno disponibili solo i metodi privati. Se ApiKey è vuoto, il client si conatterà al server websocket pubblico e saranno disponibili solo i metodi pubblici.

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWS_API_Huobi oHTX = new TsgcWS_API_Huobi();
oHTX.Client = oClient;
oHTX.Huobi.ApiKey = "your_api_key";
oHTX.Huobi.ApiSecret = "your_api_secret";
oClient.Active = true;
void OnConnect(TsgcWSConnection Connection)
{
    DoLog("#HTX Connected");
}
```

Eventi

Il client HTX implementa i seguenti eventi per controllare il flusso di connessione e ottenere i dati inviati dal server WebSocket:

- **OnConnect:** attivato quando il client websocket si connette al server WebSocket.
- **OnDisconnect:** attivato quando il client websocket si disconnette dal WebSocket Server.
- **OnHuobiSubscribed:** evento chiamato dopo una sottoscrizione riuscita.
- **OnHuobiUnSubscribed:** evento chiamato dopo un annullamento della sottoscrizione riuscito.
- **OnHuobiUpdate:** ogni volta che si verifica un aggiornamento nei dati (kline, profondità di mercato...) questo evento viene chiamato.
- **OnHuobiError:** in caso di errore nell'API HTX, questo evento fornirà informazioni sull'errore.

WebSocket API

L'API WebSocket fornisce aggiornamenti di dati di mercato in tempo reale. Consultare la documentazione [Huobi API](#) per l'elenco completo dei metodi di sottoscrizione WebSocket, inclusi i canali pubblici (SubscribeKLine, SubscribeMarketDepth, SubscribeTradeDetail, ecc.) e i canali privati (SubscribeOrderUpdates, SubscribeTradeClearing, SubscribeAccountChange).

REST API

La REST API fornisce metodi pubblici e privati per richiedere dati dai mercati e dagli account privati. La REST API è accessibile tramite il componente **TsgcHTTP_API_HTX**.

Dati di mercato (Pubblici)

Method	Description
GetServerTime	Ottiene l'ora del server.
GetSymbols	Recupera l'elenco dei simboli di trading disponibili.
GetCurrencys	Ottieni l'elenco delle valute supportate.
GetMarketTickers	Ottiene i ticker per tutti i simboli di trading.
GetMarketDetail	Recupera il dettaglio di mercato delle ultime 24 ore per un simbolo.
GetMarketDepth	Recupera la profondità del book degli ordini per un simbolo. Supporta tipo di profondità e livelli di profondità configurabili.
GetMarketTrade	Recupera l'operazione più recente per un simbolo.
GetMarketHistoryTrade	Ottieni la cronologia delle transazioni più recenti per un simbolo.
GetMarketHistoryKline	Ottiene i dati candlestick/kline. Supporta più periodi (1min, 5min, 15min, 30min, 60min, 4hour, 1day, 1mon, 1week, 1year).

Trading (Privato)

Method	Description
PlaceOrder	Inserisce un nuovo ordine. Supporta buy-market, sell-market, buy-limit, sell-limit e altri tipi di ordine.
CancelOrder	Annula un ordine esistente tramite l'ID ordine.
GetOrder	Ottiene i dettagli di un ordine specifico.
GetOpenOrders	Ottiene l'elenco degli ordini aperti per un account.
GetOrderHistory	Ottieni la cronologia degli ordini per un simbolo.

Account (Privato)

Method	Description
GetAccounts	Ottieni l'elenco degli account.
GetAccountBalance	Recupera il saldo di un account specifico.
GetAccountAssetValuation	Ottiene la valutazione totale degli asset per un tipo di conto (ad es. spot).

Di seguito è riportato un esempio di utilizzo della REST API.

```
TsgcHTTP_API_HTX oHTX = new TsgcHTTP_API_HTX();
oHTX.HTXOptions.ApiKey = "your_api_key";
oHTX.HTXOptions.ApiSecret = "your_api_secret";
// Get market tickers
oHTX.GetMarketTickers();
// Get kline data
oHTX.GetMarketHistoryKline("btcusdt", "1min", 150);
// Place an order
oHTX.PlaceOrder("12345678", "btcusdt", "buy-limit", "0.001", "50000");
// Get account balance
oHTX.GetAccountBalance("12345678");
```

WhatsApp Cloud API

Whatsapp

Invia e ricevi messaggi utilizzando una versione cloud-hosted della **WhatsApp Business Platform**. La **Cloud API** consente di implementare le WhatsApp Business API senza il costo dell'hosting di server propri e consente inoltre di scalare più facilmente la messaggistica aziendale. La Cloud API supporta fino a 80 messaggi al secondo di invio e ricezione combinati (inclusi i messaggi di testo e multimediali).

L'API WhatsApp Business consente alle medie e grandi imprese di comunicare con i propri clienti su larga scala. Tramite l'API, le aziende possono costruire sistemi che collegano migliaia di clienti ad agenti o bot, abilitando sia la comunicazione programmatica sia quella manuale. Inoltre, è possibile integrare l'API con numerosi sistemi backend, come CRM e piattaforme di marketing.

Funzionalità

Le aziende otterranno tutte le nuove funzionalità più rapidamente tramite Cloud API. Attualmente, WhatsApp Business Cloud API include tutte le funzionalità disponibili con WhatsApp Business API.

Funzionalità utili di WhatsApp Cloud API:

- **Integra** la messaggistica WhatsApp con strumenti come **CRM**, **analisi** e **app di terze parti**
- **Segno di spunta verde**, profilo WhatsApp Business verificato
- WhatsApp **Broadcast & Bulk Messaging**
- Nessuna app o interfaccia, utilizzo tramite BSP o CRM
- **WhatsApp Chatbot & automazione** chat tramite app di terze parti
- **Pianificare** i messaggi WhatsApp su larga scala
- Le funzionalità di **messaggistica interattiva** includono messaggi List, pulsanti di risposta, messaggi CTA

Usi più comuni

- **Configurazione**
 - [Creare un'App WhatsApp](#)
 - [WhatsApp Phone Number Id](#)
 - [WhatsApp Token](#)
 - [WhatsApp Webhook](#)
 - [WhatsApp Security](#)
- **Messaggi**
 - [WhatsApp Invia Messaggi](#)
 - [WhatsApp Invia Messaggi Interattivi](#)
 - [WhatsApp Invia Messaggi Template](#)
 - [WhatsApp Ricevi Messaggi e Notifiche di Stato](#)
 - [WhatsApp Invia File](#)
 - [WhatsApp Scarica Media](#)

Introduzione

Per inviare e ricevere un primo messaggio utilizzando un numero di test, completare i seguenti passaggi:

1. Configurare gli asset per sviluppatori e l'accesso alla piattaforma

- [Registrarsi come Meta Developer](#)
- [Abiliti l'autenticazione a due fattori per il Suo account](#)

- **Crea un'app Meta:** Accedere a **developers.facebook.com > My Apps > Create App**. Selezionare il tipo "Business" e seguire le istruzioni sullo schermo.

Dalla App Dashboard, faccia clic sull'app che desidera connettere a WhatsApp. Scorra verso il basso per trovare il prodotto "WhatsApp" e faccia clic su **Set up**.

Successivamente, verrà visualizzata l'opzione per selezionare un Business Manager esistente (se disponibile) o, se lo si desidera, il processo di onboarding può crearne uno automaticamente (il profilo aziendale potrà essere personalizzato in seguito, se necessario). Effettuare una selezione e fare clic su **Continua**.

Quando si fa clic su **Continua**, il processo di configurazione iniziale esegue le seguenti azioni:

- La propria App è associata al Business Manager scelto o creato automaticamente.
- Un numero di telefono di test WhatsApp viene aggiunto alla propria attività. È possibile utilizzare questo numero di telefono di test per esplorare la piattaforma WhatsApp Business senza registrare o migrare un numero di telefono reale. I numeri di telefono di test possono inviare messaggi illimitati a un massimo di 5 destinatari (che possono trovarsi ovunque nel mondo).

2. Inviare un messaggio di test

Ora è possibile aprire l'IDE e creare un nuovo progetto. Inserire un componente `TsgcWhatsapp_Client` e compilare le seguenti proprietà:

- **WhatsappOptions.PhoneNumberId:** è l'ID del numero di telefono utilizzato per inviare messaggi.
- **WhatsappOptions.Token:** è il Token di Accesso Temporaneo valido per 24 ore.

Una volta configurate correttamente queste 2 proprietà, chiamare il metodo **SendTest** per inviare il **Primo messaggio** a un numero di telefono tramite la **Whatsapp Business Platform**.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient.SendTest("34605889421");
```

3. Configurare un Webhook

Per ricevere notifiche quando si riceve un messaggio o quando lo stato di un messaggio è cambiato, è necessario configurare un endpoint Webhooks per la propria applicazione. La configurazione dei Webhooks non influisce sullo stato del numero di telefono e non interferisce con l'invio o la ricezione di messaggi.

Per iniziare, deve innanzitutto creare l'endpoint, quindi per prima cosa configuri la proprietà **ServerOptions** del componente WhatsApp Client e configuri le seguenti proprietà:

- **ServerOptions:** qui è possibile configurare l'indirizzo IP da associare, la porta di ascolto e se si utilizza SSL (il WebHook deve essere eseguito su un server sicuro; è possibile configurare il server per utilizzare SSL oppure effettuare il proxy delle richieste WebHook verso un server non HTTPS). Il server è basato su [Tsgc-WebSocketHTTPServer](#).
 - **WebhookOptions:** questa proprietà consente di impostare le proprietà del Webhook che verranno successivamente configurate nel proprio account Facebook per sviluppatori.
 - **Endpoint:** è il nome dell'endpoint; per impostazione predefinita è `/webhook`. Esempio: se il server è in ascolto su `https://www.esegece.com`, l'endpoint sarà `"https://www.esegece.com/webhook"`
 - **Token:** è una stringa di sicurezza che può contenere qualsiasi valore definito dall'utente. Viene utilizzata per verificare che la registrazione del Webhook sia corretta.

Dopo aver configurato il server, è possibile utilizzare il metodo **StartServer** per avviare il server e accettare le richieste in arrivo.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.ServerOptions.WebhookOptions.PhoneNumberId = "/webhook";
oClient.ServerOptions.WebhookOptions.Token = "MySecretToken";
```

```
oClient.StartServer();
```

Una volta che il proprio endpoint è pronto, accedere alla Dashboard dell'App.

Nel pannello di controllo dell'app, trovi il prodotto WhatsApp e clicca su **Configurazione**. Quindi, trova la sezione webhook e clicca su **Configura un webhook**. Dopo il clic, sullo schermo appare una finestra di dialogo che richiede due informazioni:

- Callback URL: questo è l'URL a cui Meta invierà gli eventi.
- Verify Token: questa stringa viene impostata da Lei quando crea il proprio endpoint webhook.

Dopo aver aggiunto le informazioni, fare clic su **Verifica e salva**.

Tornando alla App Dashboard, fare clic su **WhatsApp > Configuration** nel pannello laterale sinistro. In Webhooks, fare clic su **Manage**. Si aprirà una finestra di dialogo con tutti gli oggetti di cui si possono ricevere notifiche. Per ricevere messaggi dagli utenti, fare clic su **Subscribe per messages**.

4. Ricevere un messaggio di prova

Ogni volta che viene ricevuto un nuovo messaggio, verrà chiamato l'evento client **OnMessageReceived**.

```
void OnMessageReceived(TsgcWhatsApp_Client Sender, TsgcWhatsApp_Receive_Message Message, ref bool MarkAsRead)
{
    DoLog("Received: " + Message.Text);
}
```

Ora che il Webhook è configurato, inviare un messaggio al numero di test utilizzato. Si dovrebbe ricevere immediatamente una notifica Webhook con il contenuto del messaggio!

Le API WhatsApp non consentono l'invio di messaggi di testo libero a telefoni che non hanno contattato l'utente in precedenza (nelle ultime 24 ore). L'unico modo per inviare un messaggio di testo a un telefono che non ha mai contattato il numero dell'account sviluppatore è inviare un Template (precedentemente approvato da Meta). Per superare questa limitazione durante il test dei messaggi di testo libero, inviare prima un messaggio WhatsApp dal numero di destinazione al numero dell'account sviluppatore; successivamente sarà possibile inviare messaggi di testo libero per 24 ore.

Eventi

OnBeforeSendMessage

Questo evento viene chiamato prima che il messaggio venga inviato ai server WhatsApp. È possibile accedere al messaggio interno tramite il parametro RawMessage.

OnBeforeSubscribe

Questo evento viene chiamato prima che il server si iscriva a un topic. Utilizzi il parametro Accept per consentire o negare l'iscrizione. Per impostazione predefinita, il server si iscriverà a tutti gli eventi richiesti.

OnRawMessage

Questo evento viene chiamato quando il server riceve un nuovo messaggio che non è ancora stato analizzato, consentendo l'accesso al messaggio grezzo.

OnMessageReceived

Questo evento viene chiamato dopo che il server riceve e analizza un nuovo messaggio. Se si imposta il parametro MarkAsRead su True, il mittente riceverà una doppia spunta.

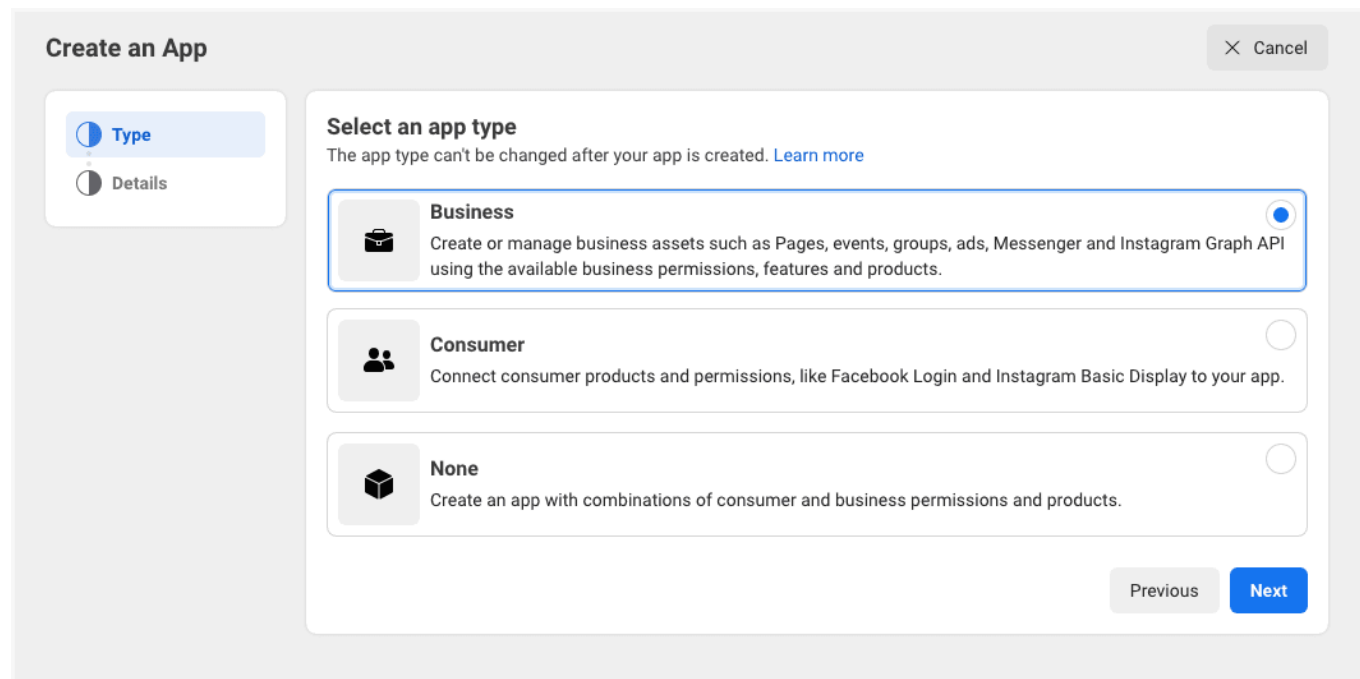
OnMessageSent

Questo evento viene richiamato ogni volta che il server riceve un nuovo messaggio di stato relativo al messaggio precedentemente inviato. Leggere la proprietà Status per sapere se il messaggio è stato inviato, consegnato o letto.

Creazione App WhatsApp

Andare su developers.facebook.com e **Creare App**.

Selezioni **Business Type** come tipo di app e prosegua.



Create an App ✕ Cancel

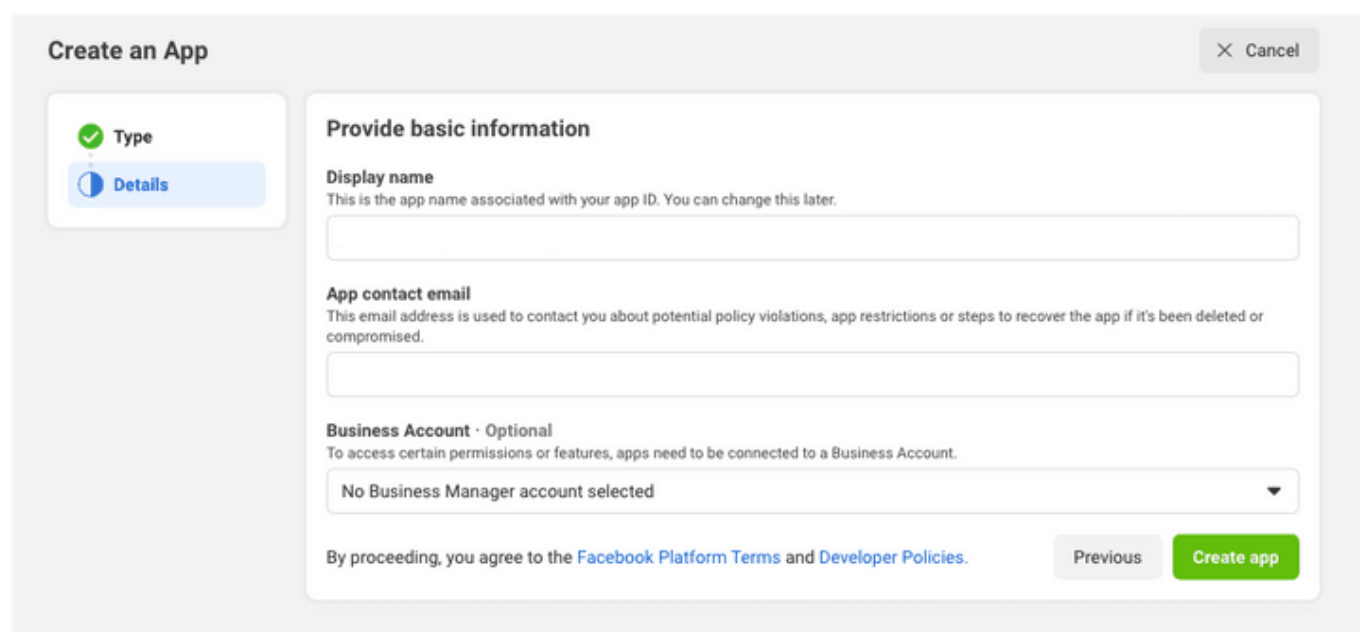
Type Details

Select an app type
The app type can't be changed after your app is created. [Learn more](#)

- Business** ☒
Create or manage business assets such as Pages, events, groups, ads, Messenger and Instagram Graph API using the available business permissions, features and products.
- Consumer** ☐
Connect consumer products and permissions, like Facebook Login and Instagram Basic Display to your app.
- None** ☐
Create an app with combinations of consumer and business permissions and products.

Previous Next

Fornisca un nome per la sua app (eviti di utilizzare nomi registrati come marchi quali “WhatsApp” o “Facebook”).



Create an App ✕ Cancel

✓ Type Details

Provide basic information

Display name
This is the app name associated with your app ID. You can change this later.

App contact email
This email address is used to contact you about potential policy violations, app restrictions or steps to recover the app if it's been deleted or compromised.

Business Account · Optional
To access certain permissions or features, apps need to be connected to a Business Account.
No Business Manager account selected ▼

By proceeding, you agree to the [Facebook Platform Terms](#) and [Developer Policies](#). Previous Create app

Una volta creata l'app, fare clic sul **pulsante WhatsApp** nella schermata successiva per aggiungere le funzionalità di invio WhatsApp alla propria app.



Nella schermata successiva, sarà necessario collegare la propria app WhatsApp al proprio account aziendale Facebook. Sarà inoltre disponibile l'opzione per creare un nuovo account aziendale se non se ne possiede ancora uno.

WhatsApp Phone Number Id

Quando ci si registra con WhatsApp Cloud API, Facebook fornisce un numero di telefono WhatsApp di prova che sarà l'indirizzo di invio predefinito dell'applicazione. Per i destinatari, si ha la possibilità di aggiungere un massimo di 5 numeri di telefono durante la fase di sviluppo senza dover effettuare alcun pagamento.

In seguito potrà registrare il Suo numero di telefono per evitare la limitazione di 5 numeri di telefono.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();  
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
```

Token WhatsApp

WhatsApp Cloud API richiede un token valido per inviare qualsiasi messaggio utilizzando la Cloud API.

Facebook fornisce un numero di telefono WhatsApp di test che consente di inviare messaggi a un massimo di 5 numeri di telefono. È possibile superare in seguito questa limitazione registrando il proprio numero di telefono.

L'API WhatsApp fornisce un **Token di Accesso Temporaneo** che sarà valido per 23 ore. Questo token deve essere configurato nel componente `TsgcWhatsApp_Client` per consentire l'invio di messaggi.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();  
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
```

Se è necessario un token a lungo termine, è possibile creare (o aggiornare) un utente di sistema e generare un nuovo token con il permesso **whatsapp_business_messaging**. Ciò consentirà di inviare e ricevere messaggi WhatsApp senza aggiornare il token ogni 23 ore.

WhatsApp Webhook

Effettui la sottoscrizione ai Webhook per ricevere notifiche relative ai messaggi ricevuti dalla Sua azienda e agli aggiornamenti dei profili dei clienti.

Crea Endpoint

Prima di poter iniziare a ricevere notifiche, è necessario creare un endpoint sul proprio server per riceverle.

Il Suo endpoint deve essere in grado di elaborare due tipi di richieste HTTPS: Richieste di verifica e Notifiche di eventi. Poiché entrambe le richieste utilizzano HTTPS, il Suo server deve avere un certificato TLS o SSL valido, correttamente configurato e installato. I certificati autofirmati non sono supportati.

Quando si configura il Webhook nelle impostazioni di WhatsApp, è necessario definire l'endpoint su cui è in ascolto il server e un Token che può avere qualsiasi valore; questo token viene utilizzato durante la registrazione dell'endpoint webhook per verificare che il sottoscrittore sia valido.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.ServerOptions.WebhookOptions.PhoneNumberId = "/webhook";
oClient.ServerOptions.WebhookOptions.Token = "MySecretToken";
oClient.StartServer();
```

Una volta configurato il Webhook, si iscriva al campo Webhook **Messages** per ricevere una notifica ogni volta che viene ricevuto un nuovo messaggio.

Per ulteriori informazioni sulla configurazione del [Server SSL](#).

Sicurezza WhatsApp

Ogni volta che viene ricevuto un nuovo messaggio o si verifica un nuovo stato di un messaggio, il server riceve una notifica nell'endpoint configurato nel [Webhook](#). Per verificare che la richiesta provenga dai server dell'API Cloud di WhatsApp, la richiesta contiene un'intestazione con una firma; è possibile configurare il client WhatsApp per verificare le firme prima di elaborare il messaggio.

Per fare ciò, è necessario innanzitutto impostare il Segreto Applicazione nella proprietà **ServerOptions.Application.Secret** e abilitare la proprietà **VerifySignature**.

Una volta configurato, ogni volta che viene ricevuto un nuovo messaggio, la firma viene prima verificata e, se non è corretta, viene restituito un errore 500 e il messaggio non viene elaborato.

WhatsApp Invia Messaggi

Tutte le chiamate API devono essere autenticate con un Access Token. Gli sviluppatori possono autenticare le proprie chiamate API con il token di accesso generato in **App Dashboard > WhatsApp > Getting Started**

Le chiamate API restituiscono l'ID del messaggio come stringa.

Messaggi di Testo

Chiamare il metodo **SendMessageText** e passare i seguenti parametri:

- **aTo:** numero di telefono
- **aText:** testo del messaggio.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmlB2LRXJkte2Y5PMNh2";
oClient.SendMessageText("34605889421", "Hello from sgcWebSockets!!!");
```

Messaggi immagine

Chiamare il metodo **SendMessageImage** e passare i seguenti parametri:

- **aTo:** numero di telefono
- **aLink:** URL dove si trova l'immagine da inviare
- **aCaption:** titolo dell'immagine (opzionale).

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmlB2LRXJkte2Y5PMNh2";
oClient.SendMessageImage("34605889421", "Hello from sgcWebSockets!!!", "logo");
```

Messaggi Documento

Chiamare il metodo **SendMessageDocument** e passare i seguenti parametri:

- **aTo:** numero di telefono
- **aLink:** url in cui si trova il documento da inviare
- **aCaption:** titolo del documento (opzionale).
- **aFileName:** nome del file (facoltativo).

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmlB2LRXJkte2Y5PMNh2";
oClient.SendMessageDocument("34605889421", "https://www.documents.com/file.txt", "Document", "file.txt");
```

Messaggi audio

Chiami il metodo **SendMessageAudio** e passi i seguenti parametri:

- **aTo:** numero di telefono
- **aLink:** url in cui si trova l'audio da inviare

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient.SendMessageAudio("34605889421", "https://www.audio.com/audio.mp3");
```

Messaggi Video

Chiamare il metodo **SendMessageVideo** e passare i seguenti parametri:

- **aTo:** numero di telefono
- **aLink:** url in cui si trova il video da inviare

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient.SendMessageVideo("34605889421", "https://www.video.com/audio.mp4");
```

Sticker Messages

Chiamare il metodo **SendMessageSticker** e passare i seguenti parametri:

- **aTo:** numero di telefono
- **aLink:** url in cui si trova l'adesivo da inviare

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient.SendMessageSticker("34605889421", "https://www.stickers.com/sticker");
```

Location Messages

Chiamare il metodo **SendMessageLocation** e passare i seguenti parametri:

- **aTo:** numero di telefono
- **aLongitude:** Longitudine della posizione.
- **aLatitude:** Latitudine della posizione.
- **aName:** Nome della posizione.
- **aAddress:** Indirizzo della posizione. Visualizzato solo se aName è impostato.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient.SendMessageLocation("34605889421", "50.159305", "9.762686", "My Location", "My Address");
```

Messaggi di contatto

Chiamare il metodo **SendMessageContact** e passare i seguenti parametri:

- **aTo**: numero di telefono
- **aName**: Nome completo, come appare normalmente (obbligatorio).
- **aPhone**: il numero di telefono (opzionale).
- **aEmail**: l'indirizzo email (facoltativo).

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();  
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";  
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmlB2LRXJkte2Y5PMNh2";  
oClient.SendMessageLocation("34605889421", "John Smith", "15550386570", "john@mail.com");
```

WhatsApp - Invio di messaggi interattivi

I messaggi interattivi offrono ai suoi utenti un modo più semplice per trovare e selezionare ciò che desiderano dalla sua attività su WhatsApp. Durante i test, i chatbot che utilizzano funzionalità di messaggistica interattiva hanno ottenuto tassi di risposta e conversioni significativamente più elevati rispetto a quelli basati su testo.

I seguenti messaggi sono considerati interattivi:

- **Elenco messaggi:** Messaggi che includono un menu con un massimo di 10 opzioni. Questo tipo di messaggio offre un modo più semplice e coerente per consentire agli utenti di effettuare una selezione quando interagiscono con un'azienda.
- **Pulsanti di risposta:** Messaggi con fino a 3 opzioni — ogni opzione è un pulsante. Questo tipo di messaggio offre agli utenti un modo più rapido per effettuare una selezione da un menu durante l'interazione con un'azienda. I pulsanti di risposta hanno la stessa esperienza utente dei template interattivi con pulsanti.

Specifiche dei messaggi interattivi

- I messaggi interattivi possono essere combinati insieme nello stesso flusso.
- Gli utenti non possono selezionare più di un'opzione contemporaneamente da un elenco o da un messaggio con pulsanti, ma possono tornare indietro e riaprire un messaggio precedente.
- I messaggi con pulsanti di lista o di risposta non possono essere utilizzati come notifiche. Al momento, possono essere inviati solo entro 24 ore dall'ultimo messaggio inviato dall'utente. Se si tenta di inviare un messaggio al di fuori della finestra di 24 ore, si riceve un messaggio di errore.

Quando utilizzarlo

I List Messages sono ideali per presentare diverse opzioni, come ad esempio:

- Un menu di assistenza clienti o FAQ
- Un menu da asporto
- Selezione di negozi o punti vendita nelle vicinanze
- Orari di prenotazione disponibili
- Scelta di un ordine recente da ripetere

I Reply Buttons sono ideali per offrire risposte rapide a partire da un insieme limitato di opzioni, come ad esempio:

- Ricarica del credito telefonico
- Modifica dei dati personali
- Riordinamento di un ordine precedente
- Richiesta di un reso
- Aggiunta di extra opzionali a un ordine alimentare
- Scelta di un metodo di pagamento

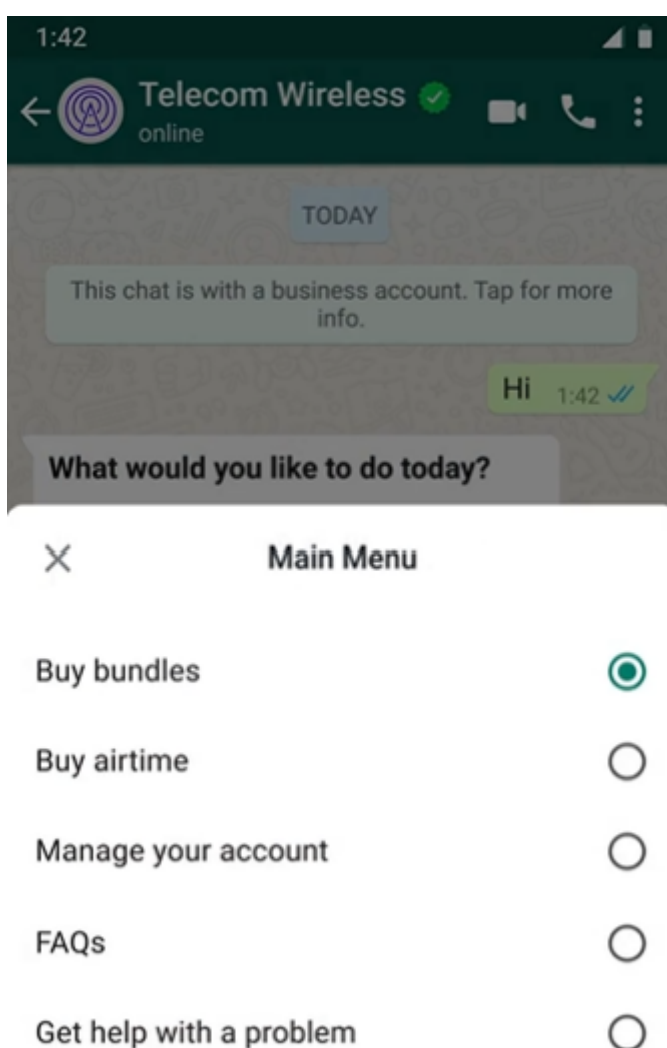
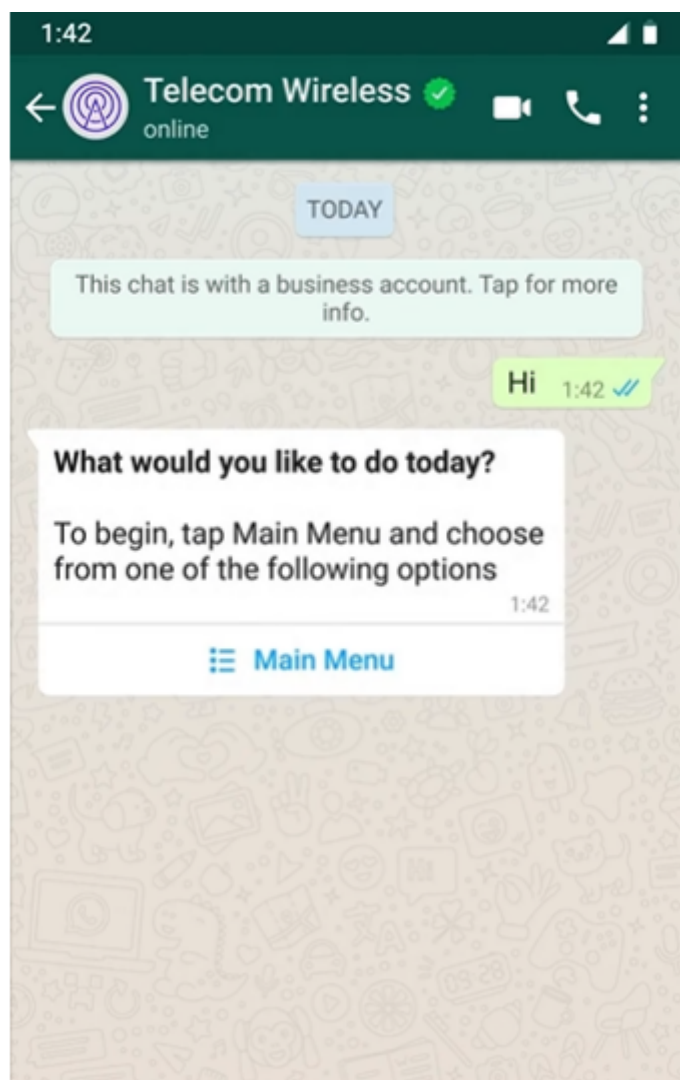
I pulsanti di risposta sono particolarmente utili per i casi d'uso "personalizzati" in cui una risposta generica non è adeguata.

Elenco Interattivo

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmlB2LRXJkte2Y5PMNh2";
oClient.SendMessageInteractiveList("
```

34605889421

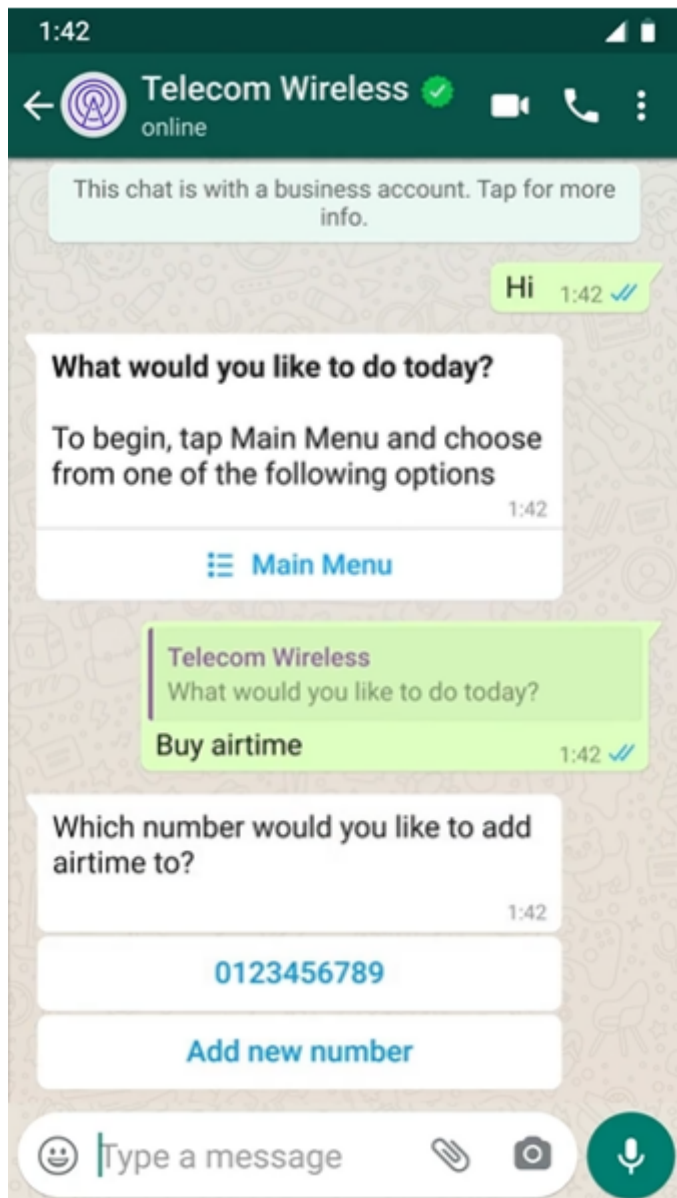
```
", "What Would you like to do today?", "To begin, Tap Main Menu and choose from of the following options", "", "M
```

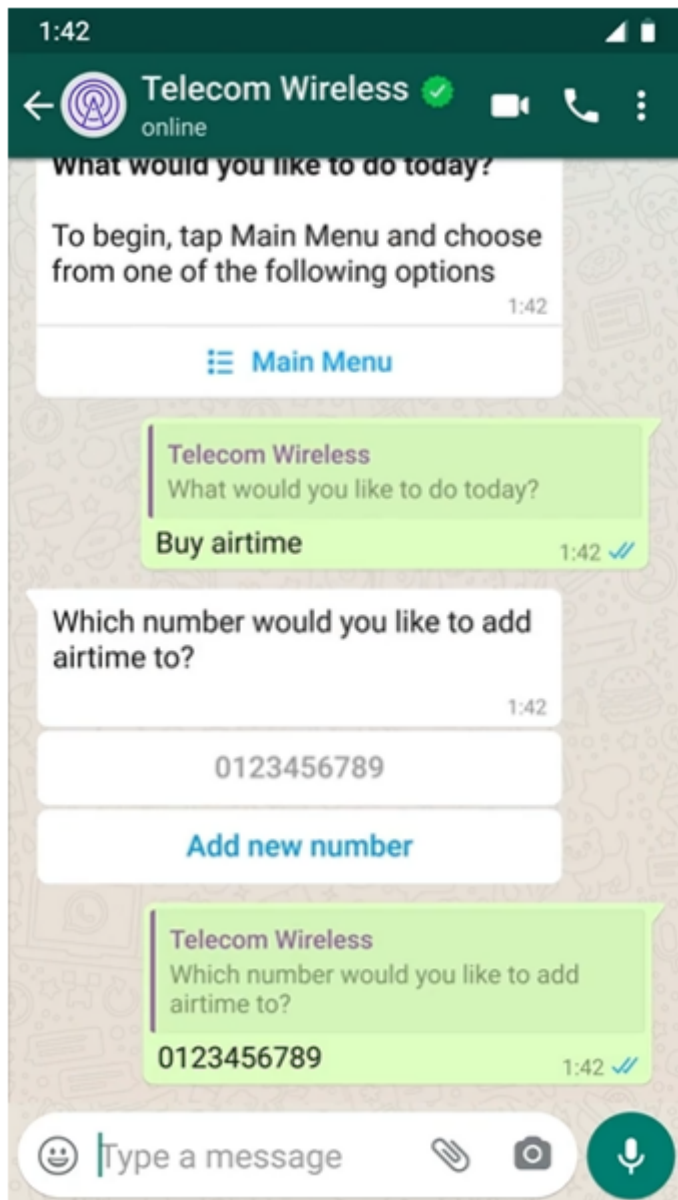


Pulsanti di risposta

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmlB2LRXJkte2Y5PMNh2";
```

```
oClient.SendMessageInteractiveButtons("34605889421", "Select an option", "Which number would you like to add airt
```





WhatsApp Invio Messaggi Template

Chiami il metodo **SendMessageTemplate** e passi i seguenti parametri:

- **aTo:** numero di telefono
- **aTemplate:** identificatore del template.
- **aLanguageCode:** lingua del template.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmlB2LRXJkte2Y5PMNh2";
oClient.SendMessageTemplate("34605889421", "hello_world", "en_US");
```

Parametri del messaggio template

I template possono includere parametri; vedere di seguito un esempio di template predefinito con parametri

```
void SendSamplePurchaseFeedbackTemplate(string aName)
{
    TsgcWhatsApp_Send_Message_Template oTemplate = new TsgcWhatsApp_Send_Message_Template();
    oTemplate.Language.Code = "en_US";
    oTemplate.TemplateName = "sample_purchase_feedback";
    // ... header
    TsgcWhatsApp_Send_Message_Template_Component oComponent = new TsgcWhatsApp_Send_Message_Template_Component();
    oComponent._Type = wapctHeader;
    oTemplate.Components.Add(oComponent);
    TsgcWhatsApp_Send_Message_Template_Parameter oParameter = new TsgcWhatsApp_Send_Message_Template_Parameter();
    oParameter.Image.Link = "https://www.esegece.com/images/esegece.png";
    oParameter._Type = wapptImage;
    oComponent.Parameters.Add(oParameter);
    // ... body
    TsgcWhatsApp_Send_Message_Template_Component oComponent2 = new TsgcWhatsApp_Send_Message_Template_Component();
    oComponent2._Type = wapctBody;
    oTemplate.Components.Add(oComponent2);
    TsgcWhatsApp_Send_Message_Template_Parameter oParameter2 = new TsgcWhatsApp_Send_Message_Template_Parameter();
    oParameter2.Text = aName;
    oParameter2._Type = wapptText;
    oComponent2.Parameters.Add(oParameter2);
    whatsapp.SendMessageTemplate("107809351952205", oTemplate);
}
```

Immagine caricata nel messaggio del template

Di seguito è riportato un esempio di template in cui, invece di utilizzare un collegamento a un'immagine, si carica prima l'immagine sul server e poi si imposta l'id del documento.

```
void SendSamplePurchaseFeedbackTemplate(string aName)
{
    // ... first upload the file
    string vId = whatsapp.UploadMedia("c:\\images\\file.png", "image/png");

    // ... send message
    TsgcWhatsApp_Send_Message_Template oTemplate = new TsgcWhatsApp_Send_Message_Template();
    oTemplate.Language.Code = "en_US";
    oTemplate.TemplateName = "sample_purchase_feedback";
    // ... header
    TsgcWhatsApp_Send_Message_Template_Component oComponent = new TsgcWhatsApp_Send_Message_Template_Component();
    oComponent._Type = wapctHeader;
    oTemplate.Components.Add(oComponent);
    TsgcWhatsApp_Send_Message_Template_Parameter oParameter = new TsgcWhatsApp_Send_Message_Template_Parameter();
    oParameter.Image.id = vId;
    oParameter._Type = wapptImage;
    oComponent.Parameters.Add(oParameter);
}
```

```
// ... body
TsgcWhatsApp_Send_Message_Template_Component oComponent2 = new TsgcWhatsApp_Send_Message_Template_Component();
oComponent2._Type = wapctBody;
oTemplate.Components.Add(oComponent2);
TsgcWhatsApp_Send_Message_Template_Parameter oParameter2 = new TsgcWhatsApp_Send_Message_Template_Parameter();
oParameter2.Text = aName;
oParameter2._Type = wapptText;
oComponent2.Parameters.Add(oParameter2);
whatsapp.SendMessageTemplate("107809351952205", oTemplate);
}
```

WhatsApp Ricezione Messaggi e Notifiche di Stato

Si iscriva ai [Webhooks](#) per ricevere notifiche sui messaggi che la Sua attività riceve e sugli aggiornamenti dei profili dei clienti.

Ogni volta che si verifica un evento trigger, la piattaforma WhatsApp Business rileva l'evento e invia una notifica a un URL Webhook precedentemente specificato. È possibile ricevere due tipi di notifiche:

- **Messaggi ricevuti:** Questo avviso notifica la ricezione di un messaggio.
- **Notifiche di stato e prezzo dei messaggi:** Questo avviso informa quando lo stato di un messaggio è cambiato — ad esempio, il messaggio è stato letto o consegnato.

Messaggi ricevuti

Ogni volta che viene ricevuto un nuovo messaggio, viene chiamato l'evento **OnMessageReceived**, tramite il quale è possibile accedere al contenuto del messaggio e contrassegnarlo come letto.

Di seguito è riportato un esempio in cui, quando viene ricevuto un nuovo messaggio di testo, viene rimandato all'utente che lo ha inviato.

```
void OnWhatsAppMessageReceived(TsgcWhatsApp_Client Sender, TsgcWhatsApp_Receive_Message Message, ref bool MarkAsRead)
{
    DoLog("Message Received: [" + Message.From + "] " + Message.Text);
    MarkAsRead = true;
}
```

Messaggi inviati

La piattaforma WhatsApp Business invia notifiche per informare dello stato dei messaggi scambiati tra Lei e gli utenti. Quando un messaggio viene inviato con successo, riceve una notifica quando il messaggio viene inviato, consegnato e letto. L'ordine di queste notifiche nell'applicazione potrebbe non riflettere l'effettiva tempistica dello stato del messaggio. Visualizzare il timestamp per determinare la tempistica, se necessario.

- **sent:** La seguente notifica viene ricevuta quando un'azienda invia un messaggio nell'ambito di una conversazione avviata dall'utente (se tale conversazione non ha avuto origine da un punto di ingresso gratuito):
- **delivered:** La seguente notifica viene ricevuta quando il messaggio di un'azienda viene consegnato e quel messaggio fa parte di una conversazione avviata dall'utente (se quella conversazione non ha avuto origine da un punto di ingresso gratuito):
- **read:** La seguente notifica viene ricevuta quando l'utente legge il messaggio.

Ogni volta che viene ricevuto un nuovo stato, viene chiamato l'evento **OnMessageSent**.

```
void OnWhatsAppMessageSent(TsgcWhatsApp_Client Sender, TsgcWhatsApp_Receive_Message Message, TsgcWhatsAppSendMessageStatusType Status)
{
    string status = "unknown";
    if (Status == TsgcWhatsAppSendMessageStatusType.wapsmstRead)
    {
        status = "read";
    }
    else if (Status == TsgcWhatsAppSendMessageStatusType.wapsmstSent)
    {
        status = "sent";
    }
}
```

```
    status = "sent";  
  }  
  else if (Status == TsgcWhatsAppSendMessageStatusType.wapsmstDelivered)  
  {  
    status = "delivered";  
  }  
  DoLog("Message Sent: " + Message.Id + " [" + status + "]);  
}
```


Invio di file tramite WhatsApp

Tutte le chiamate API devono essere autenticate con un Access Token. Gli sviluppatori possono autenticare le proprie chiamate API con il token di accesso generato in **App Dashboard > WhatsApp > Getting Started**

Le chiamate API restituiscono l'ID del messaggio come stringa.

Quando si invia un file tramite l'API WhatsApp, il messaggio viene prima caricato sui server WhatsApp e poi viene inviato un nuovo messaggio con l'ID oggetto restituito dopo il caricamento del file.

Messaggi immagine

Chiamare il metodo **SendMessageImage** e passare i seguenti parametri:

- **aTo:** numero di telefono
- **aFileName:** nome file completo (con percorso) del file immagine da inviare.
- **aFileType:**
 - image/jpeg
 - image/png
- **aCaption:** titolo dell'immagine (opzionale).

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient.SendFileImage("34605889421", "c:\\images\\image.png", "image/png");
```

Messaggi Documento

Chiamare il metodo **SendMessageDocument** e passare i seguenti parametri:

- **aTo:** numero di telefono
- **aFileName:** nome file completo (con percorso) del documento da inviare.
- **aFileType:**
 - text/plain
 - application/pdf
 - application/vnd.ms-powerpoint
 - application/msword
 - application/vnd.ms-excel
 - application/vnd.openxmlformats-officedocument.wordprocessingml.document
 - application/vnd.openxmlformats-officedocument.presentationml.presentation
 - application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
- **aCaption:** titolo del documento (opzionale).

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient.SendFileDocument("34605889421", "c:\\MyDocuments\\invoice.pdf", "application/pdf");
```

Messaggi audio

Chiami il metodo **SendMessageAudio** e passi i seguenti parametri:

- **aTo:** numero di telefono

- **aFileName:** nome file completo (con percorso) del file audio da inviare.
- **aFileType:**
 - audio/aac
 - audio/mp4
 - audio/mpeg
 - audio/amr
 - audio/ogg

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient.SendFileAudio("34605889421", "c:\\Music\\audio.mp3", "audio/mp4");
```

Messaggi Video

Chiamare il metodo **SendMessageVideo** e passare i seguenti parametri:

- **aTo:** numero di telefono
- **aFileName:** nome file completo (con percorso) del file video da inviare.
- **aFileType:**
 - video/mp4
 - video/3gp

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient.SendFileVideo("34605889421", "c:\\Videos\\video.mp4", "video/mp4");
```

Sticker Messages

Chiamare il metodo **SendMessageSticker** e passare i seguenti parametri:

- **aTo:** numero di telefono
- **aFileName:** nome file completo (con percorso) del file sticker da inviare.
- **aFileType:**
 - image/webp

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient.SendFileSticker("34605889421", "c:\\Stickers\\MySicker.webp", "image/webp");
```

WhatsApp Download Media

Se riceve un messaggio con un link a un file multimediale, può scaricare il file multimediale utilizzando il metodo **DownloadMedia**.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient.WhatsappOptions.PhoneNumberId = "107809351952205";
oClient.WhatsappOptions.Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmlB2LRXJkte2Y5PMNh2";
oClient.DownloadMedia("38923878928822", "c:\\whatsapp\\media\\image.png");
```

Per eliminare un file multimediale precedentemente caricato, è sufficiente chiamare **DeleteMedia** e passare l'id dell'oggetto come argomento.

API Telegram

Telegram

Telegram offre due tipi di API: la **Bot API**, che consente di creare programmi che utilizzano bot e HTTPS come protocollo, e la **Telegram API e TDLib**, che permettono di creare client Telegram personalizzati ed è molto più potente della Bot API.

sgcWebSockets **supporta TDLib tramite la libreria tdjson**, il che significa che è possibile costruire il proprio client Telegram. TDLib si occupa di tutti i dettagli di implementazione di rete, crittografia e archiviazione dati locale. TDLib supporta tutte le funzionalità di Telegram.

Vantaggi di TDLib (Telegram Database Library)

- **Multipiattaforma:** può essere utilizzato su Windows, Android, iOS, MacOS, Linux...
- **Facile da usare:** usa messaggi json per comunicare tra l'applicazione e telegram.
- **High-performance:** Nell'API Telegram Bot, ogni istanza TDLib gestisce più di 24000 bot.
- **Consistent:** TDLib garantisce che tutti gli aggiornamenti verranno consegnati nell'ordine corretto.
- **Reliable:** TDLib rimane stabile su connessioni internet lente e inaffidabili.
- **Secure:** Tutti i dati locali vengono crittografati utilizzando una chiave di crittografia fornita dall'utente.
- **Completamente asincrono:** Le richieste a TDLib non si bloccano a vicenda. Le risposte verranno inviate quando sono disponibili.

Configurazione

Windows

TDLib richiede altre librerie di terze parti: OpenSSL e ZLib. Queste librerie devono essere distribuite insieme alla libreria tdjson.

* Le versioni Windows richiedono VCRuntime, scaricabile da Microsoft: <https://www.microsoft.com/en-us/download/details.aspx?id=52685>. Se il problema persiste dopo l'installazione, provare a copiare le seguenti dll nella stessa cartella dell'applicazione: VCRUNTIME140.dll e VCRUNTIME140_1.dll.

Copiare le seguenti librerie nella stessa directory in cui si trova l'applicazione:

Windows 32	Windows 64
tdjson.dll	tdjson.dll
libcrypto-1_1.dll	libcrypto-1_1-x64.dll
libssl-1_1.dll	libssl-1_1-x64.dll
zlib1.dll	zlib1.dll

Creazione dell'applicazione Telegram

Per ottenere un ID API e sviluppare la propria applicazione utilizzando l'API Telegram, è necessario seguire questi passaggi:

- Registrarsi su Telegram usando qualsiasi applicazione.
- Acceda al suo Telegram core: <https://my.telegram.org>.
- Andare su **Strumenti di sviluppo API** e compilare il modulo.
- Si otterranno gli indirizzi di base nonché i parametri **api_id** e **api_hash** necessari per l'autorizzazione dell'utente.

- Al momento ogni numero può avere un solo **api_id** collegato.

Questi valori devono essere impostati nella proprietà **Telegram.API** del componente Telegram. Per autenticarsi, è possibile farlo come utente o come bot; ci sono 2 proprietà che è possibile impostare per accedere a Telegram:

- **PhoneNumber:** se si effettua il login come utente, è necessario impostare il proprio **numero di telefono** (con prefisso internazionale), esempio: +34699123456
- **BotToken:** se si effettua il login come bot, impostare il proprio token in questa proprietà (come fornito da telegram).
- **DatabaseDirectory:** consente di specificare dove si trova il database tdlb. Lasci vuoto per utilizzare la configurazione predefinita.

È possibile configurare i seguenti parametri:

- **ApplicationVersion:** versione dell'applicazione, esempio: 1.0
- **DeviceModel:** modello del dispositivo, esempio: desktop
- **LanguageCode:** codice lingua dell'utente, esempio: en.
- **SystemVersion:** versione del sistema operativo, esempio: windows.

Facoltativamente, è possibile configurare il percorso in cui si trova la libreria tdjson utilizzando il metodo **SetTDJSONPath**. Basta passare il percorso prima di avviare una nuova sessione Telegram.

Una volta configurato il componente Telegram, è possibile impostare la proprietà Active su true e il programma tenterà di connettersi a Telegram.

Codice di esempio

```
TsgcTDLib_Telegram oTelegram = new TsgcTDLib_Telegram();
oTelegram.Telegram.API.ApiHash = "your api hash";
oTelegram.Telegram.API.ApiId = "your api id";
oTelegram.PhoneNumber = "your phone number";
oTelegram.ApplicationVersion = "1.0";
oTelegram.DeviceModel = "Desktop";
oTelegram.LanguageCode = "en";
oTelegram.SystemVersion = "Windows";
oTelegram.Active = true;
```

Authorization

Ci sono due eventi che possono essere chiamati dalla libreria per ottenere un Codice di Autenticazione (consegnato nell'applicazione Telegram, non via SMS) o per fornire una password.

OnAuthenticationCode

Questo evento viene chiamato quando Telegram invia un Codice di Autorizzazione all'applicazione Telegram e l'utente deve copiare questo codice e impostarlo nell'argomento Code di questo evento.

```
void OnAuthenticationCode(TObject Sender, ref string Code)
{
    Code = "telegram code here";
}
```

OnAuthenticationPassword

Questo evento viene chiamato quando Telegram richiede all'utente di impostare una password.

Stato dell'autorizzazione

Una volta avviata l'autorizzazione, è possibile verificare lo stato tramite l'evento **OnAuthorizationStatus**; questo evento viene chiamato ogni volta che si verifica un cambiamento di stato nell'autorizzazione. Alcuni valori di Status sono:

- authorizationStateWaitTdlbParameters
- authorizationStateWaitEncryptionKey
- authorizationStateWaitPhoneNumber
- authorizationStateWaitCode
- authorizationStateLoggingOut
- authorizationStateClosed
- authorizationStateReady

Stato della Connessione

Una volta avviata la connessione, è possibile verificare lo stato della connessione tramite l'evento **OnConnectionStatus**; questo evento viene chiamato ogni volta che cambia lo stato della connessione. Alcuni valori di Status sono:

- connectionStateConnecting
- connectionStateUpdating
- connectionStateReady

Metodi principali

Il componente API TsgcTDLib_Telegram supporta diversi metodi Telegram; di seguito sono elencati i più utilizzati.

Method	Parametri	Descrizione
SendText-Message	aChatId : Id della Chat a cui verrà inviato il messaggio aText : Testo del messaggio. InlineKeyboard : Pulsanti facoltativi (solo bot).	Invia un messaggio di testo a una chat
SendRich-TextMessage	aChatId : Id della chat a cui verrà inviato il messaggio aText : Testo del messaggio. InlineKeyboard : Pulsanti facoltativi (solo bot).	Invia un messaggio in formato Rich Text a una Chat. Sintassi Markdown: • Grassetto: **bold** • Corsivo: <i>__italic__</i> • Barrato: ~~--strike--~~ • Sottolineato: <u>~~underline~~</u> • Codice: <code>##code##</code>
SendDocumentMessage	aChatId : Id della chat a cui verrà inviato il messaggio aFilePath : percorso completo del file del documento aInlineKeyboard : Pulsanti opzionali (solo bot).	Invia un documento a una chat.
SendPhoto-Message	aChatId : Id della chat a cui verrà inviato il messaggio aFilePath : percorso completo del file della foto Width : larghezza della foto. Height : larghezza della foto. InlineKeyboard : Pulsanti facoltativi (solo bot).	Invia una foto a una chat.
SendVideo-Message	aChatId : Id della Chat a cui verrà inviato il messaggio aFilePath : percorso completo del file del video aWidth : larghezza del video.	Invia un video a una chat.

	Height: larghezza del video. aDuration: durata del video in secondi. ainlineKeyboard: Pulsanti opzionali (solo bot).	
SendInvoiceMessage	aChatId: Id della chat a cui verrà inviato il messaggio ainvoice: Testo del messaggio. ainlineKeyboard: Pulsanti opzionali (solo bot).	Invia una Fattura (disponibile solo quando si è un Bot e nei Canali Privati).
EditTextMessage	aChatId: ID della chat a cui verrà inviato il messaggio aMessageId: Id del messaggio da modificare Text: Testo del messaggio. InlineKeyboard: Pulsanti facoltativi (solo bot). ShowKeyboard: pulsanti opzionali (solo bot).	Modifica il testo di un messaggio (o il testo di un messaggio di gioco)
AddChatMember	aChatId: Id della chat a cui verrà inviato il messaggio aUserId: Identificatore dell'utente. aForwardLimit: Il numero di messaggi precedenti della chat da inoltrare al nuovo membro; fino a 100. Ignorato per supergruppi e canali.	Aggiunge un nuovo membro a una chat. I membri non possono essere aggiunti a chat private o segrete. I membri non verranno aggiunti fino a quando lo stato della chat non sarà sincronizzato con il server.
AddChatMembers	aChatId: Id della chat a cui verrà inviato il messaggio aUserIds: Identificatori degli utenti da aggiungere alla chat.	Aggiunge più nuovi membri a una chat. Attualmente questa opzione è disponibile solo per supergruppi e canali. Questa opzione non può essere utilizzata per partecipare a una chat. I membri non possono essere aggiunti a un canale se ha più di 200 membri. I membri non verranno aggiunti fino a quando lo stato della chat non sarà sincronizzato con il server.
GetChatMember	aChatId: Identificatore Chat. aUserId: Identificatore Utente.	Restituisce informazioni su un singolo membro di una chat.
GetBasicGroupFullInfo	aGroupId: Identificatore del gruppo di base	Restituisce le informazioni complete su un gruppo di base tramite il suo identificatore.
GetSuperGroupMembers	aSuperGroupId: Identificatore del supergruppo o del canale. aSupergroupMembersFilter: Il tipo di utenti da restituire. Per impostazione predefinita null aOffset: Numero di utenti da saltare. aLimit: Il numero massimo di utenti da restituire; fino a 200.	Restituisce informazioni sui membri o sugli utenti bannati in un supergruppo o canale.
JoinChatByInviteLink	aLink: Link di invito da importare;	Utilizza un invite link per aggiungere l'utente corrente alla chat se possibile. Il nuovo membro non verrà aggiunto finché lo stato della chat non sarà stato sincronizzato con il server.
CreateNewSecretChat	aUserId: Identificatore dell'utente.	Crea una nuova chat segreta.

CreateNew-BasicGroup-Chat	aUserIds: Identificatori degli utenti da aggiungere alla chat. aTitle: Titolo del nuovo gruppo di base	Crea un nuovo gruppo base
CreateNew-SupergroupChat	aTitle: Titolo del nuovo SuperGroup alsChannel: True, se deve essere creata una chat di canale. aDescription: Descrizione della chat.	Crea un nuovo supergruppo o canale.
CreatePrivateChat	aUserId: Identificatore dell'utente. aForce: Se true, la chat verrà creata senza richiesta di rete. In questo caso tutte le informazioni sulla chat ad eccezione del tipo, del titolo e della foto potrebbero non essere corrette	Restituisce una chat esistente corrispondente a un determinato utente
GetChats	aOffsetOrder: Ordine della chat da cui restituire le chat aOffsetChatId: Identificatore della chat da cui restituire le chat aLimit: Il numero massimo di chat da restituire.	Restituisce un elenco ordinato di chat. Le chat sono ordinate per la coppia (order, chat_id) in ordine decrescente (non può essere utilizzato se registrato come Bot)
GetChat	aChatId: Identificatore della chat	Restituisce informazioni su una chat tramite il suo identificatore
GetChatHistory	aChatId: Identificatore della chat aFromMessageId: Identificatore del messaggio a partire dal quale deve essere recuperata la cronologia; utilizzare 0 per ottenere i risultati dall'ultimo messaggio. aOffset: Specificare 0 per ottenere risultati esattamente dal from_message_id oppure un offset negativo fino a 99 per ottenere anche alcuni messaggi più recenti. aLimit: Il numero massimo di messaggi da restituire	Restituisce i messaggi di una chat. I messaggi vengono restituiti in ordine cronologico inverso
GetUser	aUserId: Identificatore utente	Restituisce informazioni su un utente tramite il suo identificatore.
AddProxy-HTTP	aServer: Nome del server proxy. aPort: Numero della porta proxy. aUserName: Nome utente per l'accesso; può essere vuoto. aPassword: Password per l'accesso; può essere vuota. aHTTPOnly: Passare true se il proxy supporta solo richieste HTTP e non supporta connessioni TCP trasparenti tramite il metodo HTTP CONNECT.	Aggiunge un server proxy HTTP per le richieste di rete. Può essere chiamato prima dell'autorizzazione.
AddProxy-MTProto	aServer: Nome del server proxy. aPort: Numero della porta proxy. aSecret: Il segreto del proxy in codifica esadecimale.	Aggiunge un server proxy MTProto per le richieste di rete. Può essere chiamato prima dell'autorizzazione.
AddProxy-Socks5	aServer: Nome del server proxy. aPort: Numero della porta proxy. aUserName: Nome utente per l'accesso; può essere vuoto. aPassword: Password per l'accesso; può essere vuota.	Aggiunge un server proxy Socks5 per le richieste di rete. Può essere chiamato prima dell'autorizzazione.
EnableProxy	aid: ID del proxy	Abilita un proxy. È possibile abilitare un solo proxy alla volta. Può essere chiamato prima dell'autorizzazione.

DisableProxy		Disabilita il proxy attualmente abilitato. Può essere chiamato prima dell'autorizzazione.
RemoveProxy	aid: ID del proxy	Rimuove un server proxy. Può essere chiamato prima dell'autorizzazione.
GetProxies		Restituisce l'elenco dei proxy attualmente configurati. Può essere chiamato prima dell'autorizzazione.
getChatSponsoredMessage	aChatId: ID della chat	Restituisce il messaggio sponsorizzato da mostrare in una chat; solo per le chat di canale. Restituisce un errore 404 se non è presente alcun messaggio sponsorizzato nella chat.
ViewMessage	aSponsorChatId: ID della Chat sponsor aMessageId: ID del messaggio	Informa TDLib che i messaggi vengono visualizzati dall'utente. Molte attività utili dipendono dal fatto che i messaggi vengano attualmente visualizzati o meno
Logout		Disconnessione da Telegram.
TDLibSend	aRequest: Richiesta JSON.	Invia qualsiasi richiesta nel protocollo JSON.

Esempio di come inviare un messaggio di testo

```
TsgcTDLib_Telegram oTelegram = new TsgcTDLib_Telegram();
oTelegram.Telegram.API.ApiHash = "your api hash";
oTelegram.Telegram.API.ApiId = "your api id";
oTelegram.PhoneNumber = "your phone number";
oTelegram.Active = true;
...
oTelegram.SendTextMessage("1234", "My First Message from sgcWebSockets");
```

Esempio: come inviare un metodo non implementato

È possibile inviare qualsiasi messaggio JSON utilizzando il metodo TDLibSend, ad esempio: unirsi a una chat di Telegram.

```
TsgcTDLib_Telegram oTelegram = new TsgcTDLib_Telegram();
oTelegram.Telegram.API.ApiHash = "your api hash";
oTelegram.Telegram.API.ApiId = "your api id";
oTelegram.PhoneNumber = "your phone number";
oTelegram.Active = true;
...
oTelegram.TDLibSend("{\"@type\": \"joinChat\", \"chat_id\": \"1234\"}");
```

Eventi

OnBeforeReadEvent

Questo evento viene chiamato quando un messaggio JSON viene ricevuto dal componente API Telegram e non è ancora stato elaborato. Impostare la proprietà Handled su True se si elabora questo evento manualmente o se non si desidera che l'evento venga elaborato dal componente. È possibile utilizzare questo evento anche per registrare tutti i messaggi.

OnMessageText

Questo evento viene chiamato quando è stato ricevuto un nuovo testo di messaggio; legga il parametro MessageText per accedere alle proprietà del testo del messaggio.

- **ChatId:** Identificatore della chat.
- **MessageId:** Identificatore Messaggio.
- **SenderId:** Identificatore del mittente.
- **Text:** Testo del messaggio.

OnMessageDocument

Questo evento viene chiamato quando viene ricevuto un nuovo messaggio documento. Accedere a MessageDocument per ottenere accesso alle proprietà del documento.

- **ChatId:** Identificatore della chat.
- **MessageId:** Identificatore Messaggio.
- **SenderId:** Identificatore del mittente (leggere SenderChat e SenderUser da tdlb 1.7.+).
- **FileName:** Nome del documento.
- **DocumentId:** Identificatore del documento.
- **LocalPath:** percorso completo al file locale se esiste.
- **MimeType:** Mime-type del documento.
- **Size:** Dimensione del documento.
- **RemoteDocumentId:** Identificatore del documento remoto.

OnMessagePhoto

Questo evento viene generato quando viene ricevuto un nuovo messaggio fotografico. Accedere a MessagePhoto per ottenere le proprietà della foto.

- **ChatId:** Identificatore della chat.
- **MessageId:** Identificatore Messaggio.
- **SenderId:** Identificatore del mittente (leggere SenderChat e SenderUser da tdlb 1.7.+).
- **PhotoId:** Identificatore della foto.
- **LocalPath:** percorso completo al file locale se esiste.
- **Size:** Dimensione della foto.
- **RemotePhotoId:** Identificatore della foto remota.

OnVideoPhoto

Questo evento viene chiamato quando viene ricevuto un nuovo messaggio video. Accedere a MessageVideo per ottenere accesso alle proprietà del video.

- **ChatId:** Identificatore della chat.
- **MessageId:** Identificatore Messaggio.
- **SenderId:** Identificatore del mittente (leggere SenderChat e SenderUser da tdlb 1.7.+).
- **VideoId:** Identificatore della foto.
- **LocalPath:** percorso completo al file locale se esiste.
- **Width:** larghezza del video.
- **Height:** altezza del video.
- **Duration:** durata in secondi del video.
- **Dimensione:** Dimensione del video.
- **RemoteVideoId:** Identificatore della foto remota.

OnMessageSponsored

Questo evento viene chiamato quando viene ricevuto un nuovo messaggio sponsorizzato (dopo aver chiamato il metodo getChatSponsoredMessage)

- **SponsorChatId:** Identificatore della Chat dello Sponsor.
- **MessageId:** Identificatore Messaggio.
- **Text:** Testo del messaggio.

OnNewChat

Questo evento viene chiamato quando viene ricevuta una nuova chat.

- **ChatId:** Identificatore della chat.
- **ChatType:** Tipo di chat (chatTypeSupergroup, chatTypePrivate...)
- **Title:** Nome della chat.

- **SuperGroupId**: ID del gruppo se è un SuperGroup.
- **IsChannel**: restituisce se è un canale o meno.

OnNewCallbackQuery

Questo evento viene chiamato quando viene ricevuta una nuova callback query in entrata; solo per i bot.

- **Id**: Identificatore univoco della query.
- **SenderId**: Identificatore dell'utente che ha inviato la query.
- **ChatId**: Identificatore della chat in cui è stata inviata la query.
- **MessageId**: Identificatore del messaggio da cui ha avuto origine la query.
- **ChatInstance**: Identificatore che corrisponde in modo univoco alla chat a cui è stato inviato il messaggio.
- **PayloadData**: il payload da un pulsante di callback generico.
 - **Data**: Dati allegati al pulsante di callback.

OnEvent

Questo evento viene chiamato quando un nuovo Evento viene ricevuto dal componente API. Può essere utilizzato per elaborare alcuni eventi non implementati dal componente API.

- **Evento**: Nome dell'evento (eventi come: updateOption, updateUser...)
- **Text**: messaggio JSON completo

OnException

Questo evento viene chiamato se si verifica un'eccezione durante l'elaborazione dei dati dell'API Telegram.

Proprietà

MyId: restituisce l'identificatore dell'utente corrente.

Esempio di codice completo

Verificare il seguente esempio di codice che mostra come connettersi alla API di Telegram, chiedere all'utente di inserire un Codice (se richiesto dalla API di Telegram), inviare un messaggio quando la connessione è pronta e registrare i messaggi di testo ricevuti.

```
TsgcTDLib_Telegram oTelegram = new TsgcTDLib_Telegram();
oTelegram.Telegram.API.ApiHash = "your api hash";
oTelegram.Telegram.API.ApiId = "your api id";
oTelegram.PhoneNumber = "your phone number";
oTelegram.ApplicationVersion = "1.0";
oTelegram.DeviceModel = "Desktop";
oTelegram.LanguageCode = "en";
oTelegram.SystemVersion = "Windows";
oTelegram.Active = true;

void OnAuthenticationCode(TObject Sender, ref string Code)
{
    Code = InputBox("Telegram Code", "Introduce code", "");
}

void OnMessageText(TObject Sender, TsgcTelegramMessageText MessageText)
{
    Log("Message Received: " + MessageText.Text);
}

void OnConnectionStatus(TObject Sender, const string Status)
{
    if (Status == "connectionStateReady")
    {
        oTelegram.SendTextMessage("1234", "Hello Telegram!");
    }
}
```

Telegram | Inviare un messaggio Telegram con pulsanti inline

Le API Telegram consentono di inviare messaggi con pulsanti inline per selezionare opzioni come risposta (questa opzione è disponibile solo per i bot).

Prima di inviare un messaggio, creare un'istanza della classe **TsgcTelegramReplyMarkupInlineKeyboard** e chiamare il metodo **AddButtonTypeCallback** o **AddButtonTypeUrl** per ogni pulsante che si desidera creare.

Esempio

Creare un nuovo messaggio che chieda all'utente se gli piace o no il messaggio e un link per rispondere a un sondaggio. Elaborare la risposta utilizzando l'evento **OnNewCallbackQuery**.

```
TsgcTelegramReplyMarkupInlineKeyboard oReplyMarkup = new TsgcTelegramReplyMarkupInlineKeyboard();
oReplyMarkup.AddButtonTypeCallback("Yes", "I like it");
oReplyMarkup.AddButtonTypeCallback("No", "I hate it");
oReplyMarkup.AddButtonTypeUrl("Poll", "https://www.yoursite.com/telegram/poll");
sgcTelegram.SendTextMessage("123456", "Do you like the message?", oReplyMarkup);

void OnNewCallbackQuery(TObject Sender, TsgcTelegramCallbackQuery CallbackQuery)
{
    if (CallbackQuery.PayloadData.Data == "I like it") then
    {
        MessageBox.Show("yes")
    }
    else
    {
        MessageBox.Show("no");
    }
}
```

Telegram | Invia messaggio bot con pulsanti

L'API Telegram consente di inviare messaggi con pulsanti per richiedere dati all'utente (questa opzione è disponibile solo per i bot).

Prima di inviare un messaggio, creare un'istanza della classe **TsgcTelegramReplyMarkupShowKeyboard** e chiamare il metodo **AddButtonTypeRequestLocation**, **AddButtonTypeRequestPhoneNumber** o **AddButtonTypeText** per ogni pulsante che si desidera creare.

Esempio

Creare un nuovo messaggio che chieda all'utente di fornire il PhoneNumber

```
oReplyMarkup = new TsgcTelegramReplyMarkupShowKeyboard();
oReplyMarkup.AddButtonTypeRequestPhoneNumber("Give me your phone");
sgcTelegram.SendTextMessage("123456", "Please provide the information below", null, oReplyMarkup);
```

Telegram | Send Telegram Message Bold

È possibile evidenziare i messaggi di testo utilizzando grassetto, corsivo e altri stili. Utilizzare il metodo **SendRich-TextMessage**, per inviare un messaggio di testo con capacità di stile; questo metodo analizza il messaggio di testo e aggiunge automaticamente le entità richieste all'API Telegram.

Sintassi Markdown

- Grassetto [*]

```
**This is Bold**
```

- Corsivo [_]

```
_This is Italic_
```

- Strike [-]

```
--This is Strike--
```

- Sottolineato [~]

```
~~This is Underline~~
```

- Codice [#]

```
##This is Monospace##
```

Telegram | Chat non trovata come Bot

Quando si **accede come bot**, il metodo GetChats non può essere utilizzato, quindi non si ottengono tutte le chat disponibili. Se è la **prima volta che si accede come Bot** e si tenta di **inviare un messaggio** a una **Chat nota**, si riceverà questo **errore**:

```
{"@type":"error","code":5,"message":"Chat not found"}
```

La soluzione consiste nel chiamare il metodo **GetChat** prima di inviare un messaggio Telegram e passare il **ChatId** come parametro. Una volta ottenuti i dati della chat, può inviare messaggi Telegram come al solito.

Si noti che è necessario **solo** chiamare **GetChat** la **PRIMA VOLTA** prima di inviare un messaggio se non si è mai ricevuto alcun messaggio bot da questa chat. Se si chiude l'applicazione e la si riavvia, non è necessario chiamare prima GetChat perché la Chat è già salvata nel database di telegram.

Telegram | Messaggi Sponsorizzati

Ogni volta che l'utente apre un canale, è necessario chiamare `channels.getSponsoredMessages` per ricevere i messaggi sponsorizzati disponibili per quel canale. Il risultato deve essere memorizzato nella cache per 5 minuti.

Visualizzazione di messaggi sponsorizzati

I messaggi sponsorizzati devono essere visualizzati sotto tutti gli altri post nel canale, dopo che l'utente scorre ulteriormente verso il basso, oltre l'ultimo messaggio. Il canale o bot promosso specificato nel campo `from_id` deve essere visualizzato come autore del messaggio. Il messaggio dovrebbe inoltre contenere uno dei seguenti pulsanti in basso:

- **View Bot:** se un bot viene promosso. Toccando il pulsante si deve aprire la chat con il bot. Se `start_param` è specificato, l'app deve utilizzare il meccanismo di deep linking per aprire il bot.
- **Visualizza canale:** se un canale è in promozione. Toccando il pulsante si deve aprire il canale.
- **Visualizza post:** se viene promosso un canale e viene specificato `channel_post`. Toccando il pulsante si apre il particolare post del canale.

Una volta che l'intero testo è visualizzato sullo schermo (escluso il pulsante), il metodo **ViewMessage** deve essere chiamato con il `random_id` di questo messaggio sponsorizzato.

Ottieni messaggi sponsorizzati

Invia una richiesta al canale chiedendo se sono disponibili messaggi sponsorizzati; chiamare il metodo **GetChatSponsoredMessage**.

```
TsgcTDLib_Telegram oTelegram = new TsgcTDLib_Telegram();
oTelegram.Telegram.API.ApiHash = "ABCDEFGHJKLMNOP";
oTelegram.Telegram.API.ApiId = "1234";
oTelegram.PhoneNumber = "008745744155";
oTelegram.Active = true;
oTelegram->getChatSponsoredMessage("100");
```

Se la chat contiene messaggi sponsorizzati, viene chiamato l'evento **OnMessageSponsored** con il contenuto del messaggio sponsorizzato. Se non ci sono messaggi, viene restituito un errore 404.

```
private void(TObject Sender, TsgcTelegramMessageSponsored MessageSponsored)
{
    DoLog(MessageSponsored.Text);
}
```

Chiami il metodo **ViewMethod** dopo che i Messaggi Sponsorizzati sono stati mostrati all'utente.

```
oTelegram.ViewMessage("100", "54653256245");
```


Telegram | Invia messaggio fattura Telegram

Se il bot supporta la modalità inline, gli utenti possono anche inviare fatture ad altre chat tramite il bot, incluse le chat individuali con altri utenti.

I messaggi di fattura includono una foto e la descrizione del prodotto insieme a un prominente pulsante Paga. Toccando questo pulsante si apre un'interfaccia di pagamento speciale nell'app Telegram

I bot possono inviare fatture come messaggio utilizzando il metodo **SendInvoiceMessage**.

```
private void SendInvoice()
{
    TsgcTelegramSendInvoice oInvoice = new TsgcTelegramSendInvoice();
    oInvoice.Title = 'Invoice Title Test';
    oInvoice.Description = 'Description Invoice Test';
    oInvoice.Invoice.Currency = 'EUR';
    oInvoice.Invoice.Total = 800;
    oInvoice.Invoice.IsTest = True;
    oInvoice.Invoice.Payload := "payload";
    oInvoice.Invoice.ProviderToken := "provider_token";
    oInvoice.Invoice.ProviderData := "provider_data";

    sgcTelegram.SendInvoiceMessage("3284239872", oInvoice);
}
```

Telegram | Ottieni membri SuperGroup

L'API Telegram consente di ottenere informazioni sui membri di un SuperGroup. Utilizzare il metodo **GetSuperGroupMembers** per ottenere informazioni sui membri o sugli utenti bannati in un supergruppo o canale. Può essere utilizzato solo se `SupergroupFullInfo.can_get_members` è true; inoltre, potrebbero essere richiesti privilegi di amministratore per alcuni filtri.

Per impostazione predefinita il metodo restituisce tutti i membri del gruppo, ma è possibile filtrare i membri restituiti utilizzando il parametro `Filter`. Sono disponibili i seguenti parametri:

tsgmFilterNone

Valore predefinito, indica che i membri non sono filtrati.

tsgmFilterAdministrators

Restituisce il creatore e gli amministratori.

tsgmFilterBanned

Restituisce gli utenti bannati dal supergruppo o canale; può essere utilizzato solo dagli amministratori. È possibile utilizzare l'argomento `aSuperGroupMembersQuery` per la ricerca tramite una query.

tsgmFilterBots

Restituisce i membri bot del supergruppo o del canale.

tsgmFilterContacts

Restituisce i contatti dell'utente, che sono membri del supergruppo o del canale. È possibile utilizzare l'argomento `aSuperGroupMembersQuery` per la ricerca tramite una query.

tsgmFilterMention

Restituisce gli utenti che possono essere menzionati nel supergruppo.

tsgmFilterRecent

Restituisce gli utenti recentemente attivi in ordine cronologico inverso.

tsgmFilterRestricted

Restituisce i membri limitati del supergruppo; può essere utilizzato solo dagli amministratori. È possibile utilizzare l'argomento `aSuperGroupMembersQuery` per la ricerca tramite una query.

tsgmFilterSearch

Utilizzato per cercare i membri di un supergruppo o di un canale tramite una query (stringa). È possibile utilizzare l'argomento `aSuperGroupMembersQuery` per la ricerca tramite una query.

È possibile leggere il risultato utilizzando il callback `OnEvent` e filtrando per `event = "chatMembers"`.

```
Telegram.GetSupergroupMembers(1452979380);
```

```
private void OnTelegramEvent(TObject Sender, const string Event, const string Text)
{
    if (Event == "chatMembers")
    {
        ReadJSON(Text);
    }
}
```

Telegram | Aggiungi proxy Telegram

Il Client Telegram può essere configurato per utilizzare un proxy. Attualmente, Telegram supporta 3 tipi di proxy:

1. HTTP
2. MTProto
3. Socks5

Aggiungi Proxy

Per configurare un proxy HTTP, è necessario innanzitutto aggiungere il proxy alla configurazione di Telegram; per farlo, chiamare semplicemente **AddProxyHTTP** e, se l'operazione riesce, verrà restituito un messaggio con il nuovo proxy aggiunto. Una volta aggiunto il proxy all'elenco, chiamare semplicemente **EnableProxy** e passare l'**ID del proxy** ricevuto nel messaggio di conferma.

```
Telegram.AddProxyHTTP("8.8.8.8", 8080, "", "", true);  
// ... read the confirmation message and save the ID of the proxy.  
Telegram.EnableProxy(2);
```

Rimuovi proxy

Chiamare il metodo **RemoveProxy** e passare l'ID del proxy che si desidera rimuovere.

Telegram | Registra Utente Telegram

Il processo di registrazione di un nuovo utente in Telegram è molto semplice: sono necessari l'API Id e l'API Hash, e il numero di telefono del nuovo account.

Configurare il client telegram:

- API Id
- Hash API
- Numero di telefono del nuovo account Telegram.

Avviare il client; un nuovo codice verrà inviato al telefono, il client chiederà il codice Telegram e, se corretto, verrà chiamato l'evento `OnRegisterUser`. In questo evento impostare il Nome e il Cognome dell'utente per completare la registrazione.

```
TsgcTDLib_Telegram oTelegram = new TsgcTDLib_Telegram();
oTelegram.Telegram.API.ApiHash = "ABCDEFGHJKLMN";
oTelegram.Telegram.API.ApiId = "1234";
oTelegram.PhoneNumber = "008745744155";
oTelegram.Active = true;

void OnTelegramAuthenticationCode(TObject Sender, ref string Code)
{
    Code = "code sent to phone";
}

void OnTelegramRegisterUser(TObject Sender, ref string FirstName, ref string LastName)
{
    FirstName = "first name";
    LastName = "last name";
}
```

RCON

RCON

Il Source RCON Protocol è un protocollo di comunicazione basato su TCP/IP utilizzato dal Source Dedicated Server, che consente di inviare comandi di console al server tramite una "console remota", o RCON. L'uso più comune di RCON è consentire ai proprietari del server di controllare i propri game server senza accesso diretto alla macchina su cui è in esecuzione il server.

Configurazione

Il **RCON_Options** consente di configurare le seguenti proprietà:

- **Host:** indirizzo remoto del server.
- **Port:** porta di ascolto del server.
- **Password:** è la stringa segreta utilizzata per autenticarsi presso il server

Connetti

Utilizzare la proprietà **Active** per connettersi / disconnettersi dal server.

Quando Active viene impostato su True, il client tenta di connettersi al server; se la connessione riesce, cercherà di autenticarsi utilizzando la password fornita.

Il server invierà una risposta a una richiesta di autenticazione. Verrà chiamato l'evento **OnAuthenticate** e sarà possibile verificare se l'autenticazione è riuscita o meno tramite il parametro **Authenticate**.

Invio di comandi

Utilizzare il metodo **ExecCommand** per inviare comandi al server. Le risposte saranno disponibili nell'evento **OnResponse**.

```
TsgcLib_RCON oRCON = new TsgcLib_RCON();
oRCON.RCON_Options.Host = "127.0.0.1";
oRCON.RCON_Options.Port = 25575;
oRCON.RCON_Options.Password = "test";
oRCON.Active = true;

void OnAuthenticate(TObject Sender, bool Authenticated, const TsgcRCON_Packet aPacket)
{
    if (Authenticated == true)
    {
        DoLog("#authenticated");
    }
    else
    {
        DoLog("#not authenticated");
    }
}

void OnResponse(Object Sender, const string aResponse, const TsgcRCON_Packet aPacket)
{
    DoLog(aResponse);
}
```

CryptoHopper

CryptoHopper

CryptoHopper è un bot di trading automatizzato di criptovalute che consente di automatizzare il trading e la gestione del portafoglio per Bitcoin, Ethereum, Litecoin e altro.

Configurazione

Richiede un **Account sviluppatore** e una volta approvato è possibile iniziare a creare una nuova App. L'API utilizza OAuth2 per l'autenticazione, quindi è possibile recuperare **client_id** e **client_secret** dalla propria App.

```
TsgcHTTP_Cryptohopper oCryptoHopper = new TsgcHTTP_Cryptohopper();
oCryptoHopper.CryptoHopperOptions.OAuth2.ClientId = "client_id";
oCryptoHopper.CryptoHopperOptions.OAuth2.ClientSecret = "client_secret";
oCryptoHopper.CryptoHopperOptions.OAuth2.LocalIP = "127.0.0.1";
oCryptoHopper.CryptoHopperOptions.OAuth2.LocalPort = 8080;
oCryptoHopper.CryptoHopperOptions.OAuth2.Scope = "read,notifications,manage,trade";
```

Metodi

CryptoHopper utilizza HTTPs come protocollo per inviare richieste all'API. Alcuni metodi richiedono autenticazione (effettuare ordini, recuperare dati utente...) mentre altri sono pubblici (ad esempio ottenere dati di exchange).

Le funzioni restituiscono la risposta di CryptoHopper e, in caso di errore, verrà sollevata un'eccezione.

Hoppers

Gestire le operazioni di base dell'Hopper.

Method	Argomenti	Description
GetHoppers		Ottieni i Hopper degli utenti.
Create-Hopper	aBody: testo JSON di configurazione.	Crea un nuovo Hopper.
GetHopper	aid: id dell'hopper	Recupera Hopper
Delete-Hopper	aid: id dell'hopper	Elimina Hopper
Update-Hopper	aid: id dell'hopper aBody: testo json della configurazione.	Aggiorna Hopper

Ordini

Gestisce gli ordini del proprio Hopper.

Method	Argomenti	Description
GetOpenOrders	aid: id dell'hopper	Recupera tutti gli ordini aperti dell'hopper.
CreateNewOrder	aid: id dell'hopper aOrder: istanza di TsgcHTT-PCTHOrder	Crea un nuovo ordine di acquisto o vendita. Per la vendita, preferire l'endpoint sell.
PlaceMarketOrder	aid: id dell'hopper aOrderSide: cthosBuy o cthosSell. aCoin: nome della moneta, esempio: EOS aAmount: dimensione dell'ordine.	Inserisce un ordine di mercato.
PlaceLimitOrder	aid: id dell'hopper aOrderSide: cthosBuy o cthosSell. aCoin: nome della moneta, esempio: EOS aAmount: dimensione dell'ordine. aPrice: prezzo limite.	Inserisci un Ordine Limite
DeleteOrder	aid: id dell'hopper aOrderId: id dell'ordine	Elimina l'ordine per l'hopper selezionato.
DeleteAllOrders	aid: id dell'hopper	Elimina tutti gli ordini aperti per l'hopper selezionato.
GetOpenOrder	aid: id dell'hopper aOrderId: id dell'ordine	Ottieni ordine aperto nell'hopper tramite ID.

CancelOrder	aid: id dell'hopper aOrderId: id dell'ordine	Annula un ordine aperto.

Position

Gestire le posizioni del proprio Hopper.

Method	Argomenti	Description
GetPosition	aid: id dell'hopper	Ottieni le posizioni aperte dell'hopper.

Trade

Cronologia dei trade dal proprio Hopper.

Method	Argomenti	Description
GetTradeHistory		Recupera la cronologia delle operazioni dell'hopper.
GetTradeHistoryById	aid: id dell'hopper aTradeId: id dell'operazione	Recupera un trade tramite l'id dell'hopper.

Exchange

Recupera informazioni dagli exchange disponibili su CryptoHopper

Method	Argomenti	Description
GetExchange		Ottiene tutti gli exchange disponibili su Cryptohopper.
GetAllTickers	aExchange: nome dell'exchange	Ottiene il ticker per tutte le coppie
GetMarketTicker	aExchange: nome dell'exchange aPair: nome della coppia	Ottieni il ticker dalla coppia di mercato.
GetOrderBook	aExchange: nome dell'exchange aPair: nome della coppia	Ottiene l'orderbook per l'exchange, il mercato e la profondità dell'orderbook selezionati.

aDepth: profondità del book degli ordini

Webhook

Cronologia dei trade dal proprio Hopper.

Method	Argomenti	Description
CreateWebhook	aURL: url del webhook aMessageTypes: tipi di messaggio separati da virgola.	Aggiorna o crea un Webhook
DeleteWebhook	aURL: url del webhook	Elimina un Webhook esistente.

Segnali

Inviare segnali all'API CryptoHopper.

Method	Argomenti	Description
SendSignal	aSignal: è la classe con tutti i campi necessari per inviare un segnale.	Invia un Segnale
SendTestSignal	aSignal: è la classe con tutti i campi necessari per inviare un segnale.	Invia un Segnale di Test
GetSignalStats	aSignalId: id del segnale. aExchange: opzionale, nome dell'exchange.	Recupera alcune statistiche del segnale.

Come aggiornare la configurazione di Cryptohopper

Utilizzare il metodo `UpdateHopper` per aggiornare la configurazione dell'Hopper. Il metodo è sovraccaricato, quindi è possibile passare la stringa JSON o utilizzare l'oggetto `TsgcHTTPCTHopper` e usare le proprietà per abilitare o disabilitare le proprietà dell'Hopper.

```
public string EnableHopper()
{
    TsgcHTTPCTHopper oHopper = new TsgcHTTPCTHopper();
    oHopper->Enabled = 1;
    return Cryptohopper->UpdateHopper("1234", oHopper);
}
```

Come Configurare il Webhook

Il webhook consente di ricevere notifiche quando accade qualcosa in un hopper. I webhook richiedono un server HTTPS pubblico che ascolti su un indirizzo URL tutti i messaggi inviati da cryptohopper. Il server pubblico deve essere protetto con un certificato SSL (i certificati autofirmati non sono consentiti).

È necessario innanzitutto creare un webhook: configurare la proprietà `Webhook` del client `Cryptohopper` impostando l'Host e la Porta su cui il server sarà in ascolto. Quindi configurare il certificato nella proprietà `SSLOptions`.

Esempio: L'indirizzo IP pubblico sarà 1.1.1.1 e la porta di ascolto sarà 443. Il certificato è memorizzato come file PEM con il nome file `sgc.pem` e senza password.

```
/* OAuth2 */
cryptohopper.CryptohopperOptions.OAuth2.ClientId = "client_id";
cryptohopper.CryptohopperOptions.OAuth2.ClientSecret = "client_secret";
cryptohopper.CryptohopperOptions.OAuth2.LocalIP = "127.0.0.1";
cryptohopper.CryptohopperOptions.OAuth2.LocalPort = 8080;
/* Webhook */
cryptohopper.CryptohopperOptions.Webhook.Enabled = True;
cryptohopper.CryptohopperOptions.Webhook.Host = "1.1.1.1";
cryptohopper.CryptohopperOptions.Webhook.Port = 443;
cryptohopper.CryptohopperOptions.Webhook.ValidationCode = "1234";
cryptohopper.CryptohopperOptions.Webhook.SSLOptions.CertFile = "sgc.pem";
cryptohopper.CryptohopperOptions.Webhook.SSLOptions.KeyFile = "sgc.pem";
cryptohopper.CryptohopperOptions.Webhook.SSLOptions.RootCertFile = "sgc.pem";
cryptohopper.CryptohopperOptions.Webhook.SSLOptions.Password = "";
cryptohopper.StartWebhook();
```

RTCMultiConnection

RTCMultiConnection

RTCMultiConnection è una libreria JavaScript WebRTC per applicazioni peer-to-peer (condivisione schermo, conferenze audio/video, condivisione file, streaming multimediale ecc.)

Configurazione

RTCMultiConnection richiede un server WebSocket per la segnalazione; pertanto collegare la proprietà server di RTCMultiConnection a un server WebSocket (come [TsgcWebSocketHTTPServer](#)). Di seguito sono riportate le proprietà da configurare.

Server

Host: è l'indirizzo IP pubblico o il nome DNS del server WebSocket.

Port: è la porta di ascolto del server WebSocket.

IceServers

È la configurazione dei server ICE (STUN/TURN) per consentire la comunicazione tra peer. Esempio:

```
[
{
  "urls": "stun:www.yourstun.com"},
{
  "urls": "turn:www.yourturn.com",
  "username": "user",
  "credential": "secret"
}
```

VideoResolution

Qui è possibile configurare la risoluzione video delle videoconferenze; maggiore è la risoluzione, maggiore è la larghezza di banda richiesta dalla connessione.

HTMLDocuments

Configurare per ogni applicazione quale è il nome della pagina HTML che serve questo contenuto.

Esempio: se il server è in esecuzione sul sito web www.webrtc.com sulla porta 8443 e `HTMLDocuments.VideoConferencing = /RTCMultiConnection-VideoConferencing.html`, l'url per accedere a VideoConferencing sarà

```
https://www.webrtc.com:8443/RTCMultiConnection-VideoConferencing.html
```

WebRTC richiede una connessione sicura (HTTPS) e pertanto richiede l'uso di certificati; leggere ulteriori informazioni in [Server SSL](#).

Applicazioni

Nome	Description
VideoConferencing	Videochiamata multiutente (many-to-many) tramite modello di rete mesh.

Screen-Sharing	Condivisione dello schermo multi-utente (uno a molti) con topologia a stella.
Video-Broadcasting	Trasmissione video multi-utente (uno-a-molti) tramite topologia a stella.

WebPush

[RFC 8030](#)

[RFC 8291](#)

Il protocollo WebPush è definito dall'**RFC 8030** (Consegna tramite HTTP Push) e dall'**RFC 8291** (Cifratura dei messaggi).

Web Push è un **protocollo standardizzato** per la **consegna di notifiche push ai browser web**. Utilizza la Push API, un'API web standard che consente ai siti web di registrare e ricevere messaggi push. La Push API permette a un sito web di inviare messaggi push al browser di un utente, anche quando l'utente non sta navigando attivamente sul sito.

Per utilizzare Web Push, un sito web deve prima **ottenere una sottoscrizione push dal browser dell'utente**. La sottoscrizione è composta da un URL endpoint univoco e una chiave di crittografia. L'URL endpoint è un URL che il sito web può utilizzare per inviare messaggi push al browser dell'utente, mentre la chiave di crittografia viene usata per cifrare e decifrare i messaggi push.

Una volta che il sito web ha **ottenuto una sottoscrizione push**, può **inviare messaggi push** al browser dell'utente effettuando una richiesta HTTP all'URL dell'endpoint. Il messaggio push viene inviato in un formato speciale chiamato Web Push Protocol Message, che consiste in un insieme di intestazioni e un payload. Le intestazioni contengono informazioni come la chiave di crittografia e il TTL (time-to-live) del messaggio, mentre il payload contiene il contenuto effettivo del messaggio.

Quando il **browser dell'utente riceve un messaggio push**, **prima decifra il messaggio** utilizzando la chiave di cifratura. Poi **mostra la notifica all'utente**, insieme a eventuali azioni aggiuntive che l'utente può intraprendere, come ignorare la notifica o aprire il sito web.

Per garantire la sicurezza e la privacy dei messaggi push, Web Push utilizza la cifratura end-to-end e richiede che le sottoscrizioni push vengano ottenute tramite una connessione sicura (ad es. HTTPS). Inoltre, il protocollo fornisce meccanismi per autenticare il mittente di un messaggio push e prevenire abusi (ad es. limitando il numero di messaggi push che un sito web può inviare a un utente).

Componenti

Esistono 2 componenti che supportano WebPush:

- **TsgcWSAPIServer_WebPush**: implementa il protocollo WebPush lato server, consentendo di richiedere il permesso agli utenti, registrare le sottoscrizioni, inviare notifiche e altro ancora. Questo componente incap-sula già un client WebPush per l'invio di notifiche.
- **TsgcWebPush_Client**: implementa il protocollo WebPush lato client, consentendoLe di inviare notifiche agli utenti tramite il web desktop e mobile. Questo è utile se dispone già delle chiavi e dell'endpoint e desidera solo pubblicare messaggi WebPush ai client iscritti.

TsgcWSAPIServer_WebPush

Componente *TsgcWSAPIServer_WebPush*.

Introduzione

TsgcWSServer_API_WebPush è un componente che fornisce funzionalità per la gestione delle sottoscrizioni WebPush. WebPush è un protocollo per l'invio di notifiche in tempo reale alle applicazioni web che vengono eseguite nel browser. Questo componente può essere utilizzato per gestire le sottoscrizioni e inviare notifiche ai client sottoscritti. Di seguito sono riportate le proprietà, gli eventi e i metodi forniti dalla classe **TsgcWSServer_API_WebPush**, insieme ad esempi di codice che illustrano come utilizzarli.

Configurazione

1. Collegare un **TsgcWSServer_API_WebPush** a un server WebSocket utilizzando la proprietà **Server**.
2. Configurare le **chiavi pubblica e privata** nella proprietà **WebPush.VAPID**. (Gli utenti registrati possono scaricare un eseguibile che genera le chiavi VAPID per Windows).
3. Richiede la distribuzione della **versione OpenSSL 3.0.0**
4. Nella proprietà **WebPush.Endpoints** è possibile definire i propri endpoint per gestire le sottoscrizioni WebPush; per impostazione predefinita, accedendo all'endpoint `"/sgcWebPush.html"` verrà mostrata una semplice pagina web che consente di sottoscrivere le notifiche WebPush.
5. Avviare il server e accedere all'endpoint configurato per testarlo.

Riferimento

- Proprietà — 3 proprietà disponibili sul componente.
- Metodi — 3 metodi pubblici.
- Eventi — 3 eventi gestibili.

TsgcWebPush_Client

TsgcWebPush_Client è una classe che consente di inviare una notifica una volta ottenuti i dettagli di sottoscrizione.

Introduzione

TsgcWebPush_Client è una classe che consente di inviare una notifica una volta ottenuti i dettagli di sottoscrizione.

Di seguito è riportato un esempio di utilizzo del client WebPush per inviare una notifica dato un endpoint, una chiave pubblica e un segreto di autenticazione da una sottoscrizione WebPush.

```
public void SendWebPushNotification()
{
    var oSubscription = new TsgcHTTP_API_WebPush_PushSubscription();
    oSubscription.Endpoint = "endpoint";
    oSubscription.PublicKey = "public key";
    oSubscription.AuthSecret = "authentication secret";
    var oWebPush = new TsgcWebPush_Client();
    oWebPush.VAPID.PEM.PrivateKey.Text = "private_key_pem";
    oWebPush.VAPID.DER.PrivateKey = "private_key";
    oWebPush.VAPID.DER.PublicKey = "public_key";
    oWebPush.SendNotification(oSubscription, "{\"title\": \"eSeGeCe Notification\", \"body\": \"Hello from es\"}");
}
```

Riferimento

- Proprietà — 3 proprietà disponibili sul componente.
- Metodi — 1 metodo pubblico.
- Eventi — 0 eventi gestibili.

Estensioni

Il protocollo WebSocket è progettato per essere esteso. I client WebSocket possono richiedere estensioni e i server WebSocket possono accettare alcune o tutte le estensioni richieste dai client.

Estensioni supportate:

1. [Deflate-Frame](#): comprimi i frame WebSocket.
2. [PerMessage-Deflate](#): comprime i messaggi WebSocket.

Extensions | PerMessage-Deflate

PerMessage è un'estensione del protocollo WebSocket; se l'estensione è supportata sia dal server che dal client, entrambi possono comprimere i messaggi trasmessi:

- Utilizza Deflate come metodo di compressione.
- La compressione si applica solo ai dati applicativi (i frame di controllo e le intestazioni non sono interessati).
- Il server e il client possono selezionare quali messaggi verranno compressi.

Max Window Bits

Questa estensione consente di personalizzare la dimensione della finestra scorrevole lato server e lato client utilizzata dall'algoritmo LZ77 (tra 8 e 15). Maggiore è questo valore, più è probabile trovare ed eliminare i duplicati, ma si consuma più memoria e cicli CPU. 15 è il valore predefinito.

Nessuna sovrascrittura del contesto

Per impostazione predefinita, i messaggi precedenti vengono utilizzati per la compressione e la decompressione. Se i messaggi sono simili, questo migliora il rapporto di compressione. Se abilitato, ogni messaggio viene compresso utilizzando solo i propri dati. Per impostazione predefinita, è disabilitato.

MemLevel

Questo valore non viene negoziato tra server e client. Impostato a 1, utilizza la quantità minima di memoria ma rallenta l'algoritmo di compressione e riduce il rapporto di compressione; impostato a 9, utilizza la massima quantità di memoria e offre le prestazioni migliori. Per impostazione predefinita è impostato a 1.

Extensions | Deflate-Frame

Questa è un'estensione del protocollo WebSocket che consente la compressione dei frame inviati tramite il protocollo WebSocket, supportata dai browser WebKit come Chrome o Safari. Questa estensione è supportata nei componenti Server e Client.

Questa estensione è stata deprecata.

MCP

MCP (Model Context Protocol) è un protocollo standardizzato progettato per definire e gestire lo **scambio contestuale di informazioni** tra modelli linguistici e sistemi esterni.

Consente una comunicazione strutturata, sicura e dinamica che permette ai modelli AI di **comprendere, estendere e interagire** con strumenti esterni o fonti di dati al di là del loro ambiente nativo.

MCP fornisce un'**interfaccia uniforme** per il passaggio di metadati contestuali, consentendo ai modelli di interpretare istruzioni, recuperare dati ed eseguire attività in modo prevedibile e interoperabile, simile nello spirito al modo in cui *WebAuthn* standardizza i flussi di autenticazione.

Scopo

MCP è stato sviluppato per risolvere le sfide della **frammentazione del contesto** e della **incoerenza dell'integrazione** tra modelli, client e strumenti.

Con MCP, qualsiasi client conforme può comunicare con qualsiasi modello o servizio compatibile con MCP attraverso una struttura condivisa e ben definita.

I casi d'uso tipici includono:

- Allegare **metadati di contesto** (sessione utente, stato dell'applicazione, permessi, ecc.) alle query del modello.
- Scambio dinamico di **definizioni di funzioni o strumenti**.
- Gestione di **interazioni con stato** tra sessioni del modello.
- Stabilire una **interoperabilità sicura** tra LLM, SDK e servizi backend.

Componenti

- **TsgcWSAPIServer_MCP**: Il componente fornisce una soluzione semplice ma potente per implementare il server MCP, con funzionalità come richieste di Tools, Prompts e Resources, Autenticazione, controllo del flusso tramite eventi e molto altro.
- **TsgcWSAPIClient_MCP**: Il componente fornisce una soluzione semplice ma potente per implementare il client MCP, con funzionalità come richieste di Tools, Prompts e Resources e altro ancora.

MCP Server | Sessioni

Dopo una inizializzazione riuscita tra il server e il client, viene creata una nuova sessione. La sessione viene sempre inviata nella richiesta e nella risposta tra il client e il server per identificare quale sessione viene utilizzata in un flusso richiesta/risposta. Quando si utilizza il trasporto HTTP, la durata di una sessione è una singola richiesta e risposta HTTP, il che significa che una volta inviata la risposta, la sessione viene chiusa.

L'oggetto sessione viene passato ad ogni evento; per sapere quando una sessione è stata creata o eliminata, utilizzare i seguenti eventi:

- **OnMCPSessionNew**: ogni volta che viene creato un nuovo id di sessione questo evento viene chiamato.
- **OnMCPSessionEnd**: quando una sessione è terminata, viene chiamato questo evento.

OnMCPSessionNew

L'evento viene chiamato quando viene creata una nuova sessione. È possibile accedere al parametro Session per ottenere l'ID della sessione, quando la sessione scade e altre informazioni.

OnMCPSessionEnd

L'evento viene chiamato quando una sessione è terminata.

```
public void OnMCPSessionEnd(object sender, TsgcAI_MCP_Session aSession)
{
    Console.WriteLine("#session_end: " + aSession.Id);
}
public void OnMCPSessionNew(object sender, TsgcAI_MCP_Session aSession)
{
    Console.WriteLine("#session_new: " + aSession.Id);
}
```

MCP Server | Strumenti

Il **TsgcWSAPIServer_MCP** è il componente di alto livello che espone le funzionalità del server Model Context Protocol (MCP) tramite `sgcWebSockets`. Quando i **client MCP richiedono tool** secondo la specifica MCP Server Tools, questo componente orchestra il gateway HTTP, il ciclo di vita della sessione e la serializzazione JSON-RPC in modo che il Suo codice Delphi possa concentrarsi sull'implementazione della logica di business.

Le richieste HTTP in entrata che raggiungono l'endpoint MCP definito in **EndpointOptions.Endpoint** vengono intercettate da **TsgcWSServer_API_MCP**. Il componente crea (o riprende) una sessione MCP, inoltra la richiesta al motore `TsgcAI_MCP_Server` sottostante e infine scrive la risposta JSON-RPC al client HTTP preservando l'istestazione `mcp-session-id` richiesta dalla specifica.

Elenco degli strumenti

Il server mantiene un catalogo in memoria degli strumenti in **TsgcAI_MCP_ToolsList**. Gli strumenti sono garantiti come unici per nome, espongono descrizioni e forniscono una descrizione dell'input simile a JSON Schema, che viene emessa nella risposta al metodo `tools.list` della specifica. Quando un client invoca `tools.list`, `TsgcWSAPIServer_MCP` carica la richiesta, serializza il catalogo corrente e lo restituisce con un codice di stato HTTP 200.

Codice di esempio che pubblica un semplice strumento calcolatrice:

```
public partial class MainForm : Form
{
    private void MainForm_Load(object sender, EventArgs e)
    {
        var oTool = MCPServer.Tools.AddTool("math.add", "Adds two numbers");
        // Define input schema properties
        oTool.InputSchema.Properties.AddProperty("a", true, AIMCPJsonType.Number, "Left operand");
        oTool.InputSchema.Properties.AddProperty("b", true, AIMCPJsonType.Number, "Right operand");
    }
}
```

Una risposta `tools.list` conforme inviata al client rispecchierà il payload dello schema JSON MCP. Ad esempio:

```
{
  "jsonrpc": "2.0",
  "id": "42",
  "result": {
    "tools": [
      {
        "name": "math.add",
        "description": "Adds two numbers",
        "inputSchema": {
          "type": "object",
          "required": ["a", "b"],
          "properties": {
            "a": { "type": "number", "description": "Left operand" },
            "b": { "type": "number", "description": "Right operand" }
          }
        }
      }
    ]
  }
}
```

Tool Request

Quando un client emette una richiesta JSON-RPC `tools.call`, **TsgcWSAPIServer_MCP** popola l'oggetto di richiesta fortemente tipizzato (incluso il nome dello strumento e gli argomenti forniti) prima di generare l'evento **OnMCPRe-**

questTool . Il gestore popola il payload della risposta che viene poi serializzato e inviato al client, insieme a un codice di stato HTTP di successo.

Un gestore tipico ha questo aspetto:

```
private void MCPServer_MCPRequestTool(
    object sender,
    TsgcAI_MCP_Session session,
    TsgcAI_MCP_Request_ToolsCall request,
    TsgcAI_MCP_Response_ToolsCall response)
{
    double LA, LB;
    if (request.Params.Name == "math.add")
    {
        LA = request.Params.Arguments.Node["a"].AsNumber;
        LB = request.Params.Arguments.Node["b"].AsNumber;
        response.Result.Content.AddText(
            string.Format("Sum = {0:F2}", LA + LB)
        );
    }
    else
    {
        response.Result.IsError = true;
    }
}
```

La risposta JSON-RPC generata seguirà lo schema tools.call della specifica, racchiudendo una o più parti di contenuto e opionalmente un payload strutturato. Poiché il componente cancella la risposta e imposta il codice HTTP per conto dell'utente, non è necessario alcun collegamento aggiuntivo.

Superficie API pubblica

Diverse classi helper espongono il catalogo degli strumenti e i helper per le risposte agli eventi. Gli elenchi seguenti riassumono i metodi tipicamente chiamati dal codice di produzione.

TsgcAI_MCP_ToolsList

- **AddTool(const Name, Description):** crea o sostituisce una voce di strumento e restituisce il **TsgcAI_MCP_Tool** sottostante in modo da poterne configurare schemi e valori predefiniti.
- **Clear:** rimuove tutti gli strumenti registrati. Chiamare questo metodo prima di ricostruire il catalogo quando la configurazione viene caricata dal disco.
- **Count / Items[Index]:** espongono il contenuto dell'elenco per enumerazione, diagnostica o serializzazione personalizzata.

TsgcAI_MCP_Tool

- **InputSchema.Properties.AddProperty(Name, Required, JsonType, Description):** documenta il payload JSON di input che verrà pubblicizzato tramite **tools.list**.
- **InputSchema.Properties.Clear:** ricrea lo schema dinamicamente quando l'applicazione aggiorna le capacità degli strumenti in fase di esecuzione.
- **Description:** proprietà scrivibile che memorizza il riepilogo leggibile condiviso con i client.

TsgcAI_MCP_Response_ToolsCall

- **Result.Content.AddText(Value):** aggiunge blocchi di testo alla risposta dello strumento. Più chiamate producono risposte in più parti.
- **Result.Content.AddImage(Data, MimeType):** aggiunge blocchi immagine alla risposta dello strumento. Più chiamate producono risposte multi-parte
- **Result.Content.AddAudio(Data, MimeType):** aggiunge blocchi audio alla risposta dello strumento. Chiamate multiple producono risposte multi-parte.

- **Result.Content.AddEmbeddedResource(Uri, Title, Text, MimeType):** accoda blocchi di risorse incorporate alla risposta dello strumento. Più chiamate producono risposte multi-parte.
- **Result.Content.AddResourceLink(Uri, Name, MimeType):** aggiunge blocchi di link a risorse alla risposta dello strumento. Più chiamate producono risposte multi-parte.
- **Result.Content.Clear:** reimposta qualsiasi output precedentemente generato prima di aggiungere nuovo contenuto.
- **Result.IsError:** impostare questo flag su **True** quando la chiamata allo strumento non può essere soddisfatta, in modo che il framework serializzi un oggetto di errore MCP.

Esempio avanzato

Il prossimo snippet costruisce uno strumento di ricerca geografica che valida gli input, restituisce più blocchi di testo e imposta **IsError** su **True** quando i parametri non sono validi.

```
private void MainForm_Load(object sender, EventArgs e)
{
    MCPServer.Tools.Clear();
    var tool = MCPServer.Tools.AddTool("geo.locate", "Returns coordinates for a city and country");
    tool.InputSchema.Properties.Clear();
    tool.InputSchema.Properties.AddProperty("city", true, AIMCPJsonType.String, "City to search");
    tool.InputSchema.Properties.AddProperty("country", false, AIMCPJsonType.String, "Optional ISO country code");
}

private void MCPServer_MCPRequestTool(
    object sender,
    TsgcAI_MCP_Session session,
    TsgcAI_MCP_Request_ToolsCall request,
    TsgcAI_MCP_Response_ToolsCall response)
{
    if (!string.Equals(request.Params.Name, "geo.locate", StringComparison.OrdinalIgnoreCase))
        return;
    response.Result.Content.Clear();
    var city = request.Params.Arguments.Node["city"].AsString();
    var country = request.Params.Arguments.Node["country"].AsString();
    if (string.IsNullOrEmpty(city))
    {
        response.Result.Content.AddText("City cannot be empty.");
        response.Result.IsError = true;
        return;
    }
    response.Result.Content.AddText($"City: {city}");
    if (!string.IsNullOrEmpty(country))
        response.Result.Content.AddText($"Country: {country}");
    response.Result.Content.AddText($"Coordinates: {LookupCityCoordinates(city, country)}");
}
```

MCP Server | Prompt

In MCP, i *prompt* sono template o flussi di lavoro riutilizzabili esposti dal server. Sono controllati dall'utente (i client scelgono quale prompt invocare). I due metodi principali sono:

- **prompts/list**: Richiede l'elenco di tutte le definizioni di prompt disponibili esposte dal server. È gestito internamente dal componente `TsgcWSAPIServer_MCP`.
- **prompts/get**: Recupera il contenuto e i metadati opzionali di un prompt specifico, tipicamente una sequenza di messaggi che descrive come il client dovrebbe interagire. L'evento **OnMCPRequestPrompt** viene chiamato quando il server riceve una nuova richiesta.

Inoltre, se il server imposta `prompts.listChanged = true` nelle sue capacità dichiarate, può inviare notifiche come `notifications/prompts/list_changed` quando il catalogo dei prompt viene aggiornato.

Quando un client richiede `prompts/get`, invia gli argomenti (se presenti) e si aspetta una sequenza di messaggi (con ruoli `user` o `assistant`) con blocchi di contenuto (di solito testo, ma potrebbero incorporare risorse o immagini).

Se qualcosa è errato (prompt mancante, argomenti non validi), il server restituirà codici di errore JSON-RPC come `-32602` (Parametri non validi) o `-32603` (Errore interno).

Prompts List

Il server mantiene un catalogo in memoria dei prompt in **TsgcAI_MCP_PromptsList**. I prompt sono garantiti come univoci per nome, espongono descrizioni e forniscono una descrizione di input simile a JSON Schema che viene emessa nella risposta al metodo `prompts.list` della specifica. Quando un client invoca `prompts.list`, `TsgcWSAPIServer_MCP` carica la richiesta, serializza il catalogo corrente e lo invia con un codice di stato HTTP 200.

Codice di esempio che pubblica un prompt di revisione del codice:

```
public partial class MainForm : Form
{
    private void MainForm_Load(object sender, EventArgs e)
    {
        // Clear existing prompts
        MCPServer.Prompts.Clear();
        // Create a new prompt named "CodeReview"
        var oPrompt = MCPServer.Prompts.AddPrompt(
            "CodeReview",
            "Asks the LLM to analyze code quality and suggest improvements"
        );
        // Add argument 'code'
        oPrompt.Arguments.AddArgument(
            "code",
            "The code to review",
            true // required
        );
    }
}
```

Richiesta Prompt

Quando un client emette una richiesta JSON-RPC `prompts.get`, **TsgcWSAPIServer_MCP** popola l'oggetto di richiesta fortemente tipizzato (incluso il nome del prompt e gli argomenti forniti) prima di generare l'evento **OnMCPRequestPrompt**. Il gestore popola il payload della risposta che viene quindi serializzato e restituito al client, insieme a un codice di stato HTTP di successo.

Un gestore tipico ha questo aspetto:


```
private void MCPServer_MCPRequestPrompt(
    object sender,
    TsgcAI_MCP_Session session,
    TsgcAI_MCP_Request_PromptsGet request,
    TsgcAI_MCP_Response_PromptsGet response)
{
    if (request.Params.Name == "CodeReview")
    {
        response.Result.Description = "Code review prompt";
        response.Result.Messages.AddText(
            "user",
            "Please review this Delphi code: ShowMessage('Hello World');"
        );
    }
}
```

Superficie API pubblica

I prompt sono modellati tramite alcune classi helper. I seguenti metodi vengono utilizzati più frequentemente nella cura dei template di prompt e nella composizione di una risposta **prompts/get**.

TsgcAI_MCP_PromptsList

- **AddPrompt(const Name, Description):** registra una nuova voce di prompt e restituisce un oggetto **TsgcAI_MCP_Prompt** in modo che possano essere configurati argomenti e messaggi.
- **Clear:** svuota il catalogo, utile durante l'avvio quando i prompt vengono caricati dai file di configurazione.
- **Count / Items[Index]:** fornisce accesso al contenuto dell'elenco per diagnostica, visualizzazione nell'interfaccia utente o serializzazione manuale.

TsgcAI_MCP_Prompt

- **Arguments.AddArgument(Name, Description, Required):** dichiara gli input strutturati accettati dal prompt.
- **Arguments.Clear:** ricostruisce l'elenco degli argomenti quando il prompt cambia in fase di esecuzione.
- **Messages.AddText(Role, Text):** pre-popola i turni conversazionali che vengono rinviati al client come parte della definizione del prompt.

TsgcAI_MCP_Response_PromptsGet

- **Result.Description:** breve riepilogo leggibile dall'utente che accompagna il payload del prompt.
- **Result.Messages.AddText(Role, Text):** inserisce messaggi aggiuntivi dinamicamente quando il prompt viene recuperato.
- **Result.Messages.AddImage(Data, MimeType):** aggiunge blocchi di immagini alla risposta del prompt. Più chiamate producono risposte multi-parte.
- **Result.Messages.AddAudio(Data, MimeType):** aggiunge blocchi audio alla risposta del prompt. Più chiamate producono risposte multi-parte.
- **Result.Messages.AddEmbeddedResource(Uri, Title, Text, MimeType):** aggiunge blocchi di risorse incorporate alla risposta del prompt. Più chiamate producono risposte multi-parte.
- **Result.Messages.Clear:** azzera la risposta prima di riutilizzare l'oggetto.

Esempio avanzato

L'esempio successivo pubblica un prompt che guida un LLM attraverso un flusso di lavoro di triage. Mette in evidenza gli argomenti opzionali e come restituire istruzioni in più passaggi nel gestore.

```

private void MainForm_Load(object sender, EventArgs e)
{
    MCPServer.Prompts.Clear();
    var prompt = MCPServer.Prompts.AddPrompt("IssueTriage", "Collect details before escalating support tickets");
    prompt.Arguments.Clear();
    prompt.Arguments.AddArgument("summary", "One-line description supplied by the user", true);
    prompt.Arguments.AddArgument("customerPriority", "Optional priority label", false);
    prompt.Messages.AddText("system", "You are a support assistant that triages technical issues.");
}

private void MCPServer_MCPRequestPrompt(
    object sender,
    TsgcAI_MCP_Session session,
    TsgcAI_MCP_Request_PromptsGet request,
    TsgcAI_MCP_Response_PromptsGet response)
{
    if (!string.Equals(request.Params.Name, "IssueTriage", StringComparison.OrdinalIgnoreCase))
        return;
    response.Result.Description = "Guided checklist for support analysts.";
    response.Result.Messages.Clear();
    response.Result.Messages.AddText("user", "Review the ticket summary and ask clarifying questions.");
    response.Result.Messages.AddText("assistant", "Acknowledge the request and confirm the escalation path.");
    if (!string.IsNullOrEmpty(request.Params.Arguments.Node["customerPriority"].AsString()))
        response.Result.Messages.AddText("assistant", "Adjust SLA checks based on the provided customer priority.");
}

```

MCP Server | Risorse

In MCP, le **risorse** rappresentano oggetti dati indirizzabili esposti dal server, come file, record di database, documenti generati o output dinamici di API.

Sono *controllati dal client*, il che significa che il client può decidere quale risorsa richiedere e come interpretare il contenuto.

I due metodi principali coinvolti sono:

- **resources/list**: Richiede l'elenco di tutte le definizioni di risorse disponibili esposte dal server. È gestito internamente dal componente `TsgcWSAPIServer_MCP`.
- **resources/read**: Recupera il contenuto e i metadati opzionali di una risorsa specifica. L'evento **OnMCPRequestResource** viene chiamato quando una nuova richiesta viene ricevuta dal server. Una risorsa tipicamente include:
 - **uri**: identificatore della risorsa.
 - **mimeType**: tipo di contenuto (text/plain, application/json, image/png, ecc.).
 - **data o content**: il payload effettivo (testo o binario).

Inoltre, se il server imposta `resources.listChanged = true` nelle sue capacità dichiarate, può inviare notifiche come `notifications/resources/list_changed` quando il catalogo delle risorse si aggiorna.

Elenco Risorse

Il server mantiene un catalogo in memoria delle risorse in **TsgcAI_MCP_ResourcesList**. È garantito che le risorse siano univoche per uri, espongono descrizioni e forniscono un URI di input simile a JSON-Schema che viene emesso nella risposta al metodo `resources.list` della specifica. Quando un client invoca `resources.list`, **TsgcWSAPIServer_MCP** carica la richiesta, serializza il catalogo corrente e lo rinvia con un codice di stato HTTP 200.

Codice di esempio che pubblica una risorsa file:

```
public partial class MainForm : Form
{
    private void MainForm_Load(object sender, EventArgs e)
    {
        // Clear existing resources
        MCPServer.Resources.Clear();
        // Register a Rust source file resource
        MCPServer.Resources.AddResource(
            "file:///project/src/main.rs", // URI
            "main.rs", // Name
            "Rust Software Application Main File", // Title
            "Primary application entry point", // Description
            "text/x-rust" // MIME type
        );
    }
}
```

Richiesta di risorse

Quando un client emette una richiesta JSON-RPC `resource.read`, **TsgcWSAPIServer_MCP** popola l'oggetto richiesta fortemente tipizzato (incluso l'uri della risorsa, il nome e gli argomenti forniti) prima di generare l'evento **OnMCPRequestResource**. Il gestore popola il payload della risposta che viene quindi serializzato e restituito al client, insieme a un codice di stato HTTP di successo.

Un gestore tipico ha questo aspetto:

```
private void MCPServer_MCPRequestResource(
    object sender,
    TsgcAI_MCP_Session session,
    TsgcAI_MCP_Request_ResourcesRead request,
    TsgcAI_MCP_Response_ResourcesRead response)
{
    if (request.Params.Uri == "file:///project/src/main.rs")
    {
        response.Result.Contents.AddContentText(
            "file:///project/src/main.rs", // URI
            "main.rs", // Name
            "Rust Software Application Main File", // Description
            "text/x-rust", // MIME type
            "fn main() {\n    println!(\"Hello world!\");\n}" // Content
        );
    }
}
```

Superficie API pubblica

Le risorse combinano una voce di catalogo (**TsgcAI_MCP_Resource**) e gli helper di streaming utilizzati all'interno di **OnMCPRequestResource**. Questi sono i metodi che si utilizzeranno più frequentemente.

TsgcAI_MCP_ResourcesList

- **AddResource(Uri, Name, Title, Description, MimeType):** registra una risorsa statica o generata dinamicamente.
- **Clear:** rimuove ogni definizione di risorsa, utile prima di ricostruire l'elenco dai file di configurazione.
- **Count / Items[Index]:** espone il catalogo per l'enumerazione o le visualizzazioni diagnostiche.

TsgcAI_MCP_Resource

- **MimeType / Title / Description:** proprietà scrivibili che descrivono il payload della risorsa restituito al client.
- **Name / Uri:** utilizzare queste proprietà per aggiornare gli identificatori senza dover registrare nuovamente la voce della risorsa.

TsgcAI_MCP_Response_ResourcesRead

- **Result.Contents.AddContentText(Uri, Name, Title, MimeType, Text):** invia un payload di testo al chiamante.
- **Result.Contents.AddContentBlob(Uri, Name, Title, MimeType, Text):** invia un payload di testo al chiamante.
- **Result.Contents.Clear:** reimposta il payload in uscita prima di riutilizzare l'oggetto risposta.
- **Result.IsError:** contrassegna la risposta come errore quando la risorsa non può essere prodotta.

Esempio avanzato

L'esempio seguente pubblica una risorsa JSON che restituisce metriche aggregate. Dimostra come validare l'URI in arrivo e creare risposte diverse in base alla richiesta.

```
private void MainForm_Load(object sender, EventArgs e)
{
    MCPServer.Resources.Clear();
    MCPServer.Resources.AddResource(
        "metrics://daily",
        "DailyMetrics",
        "Daily KPI snapshot",
        "Summaries generated each night",
    );
}
```

```

        "application/json");
    }
    private void MCPServer_MCPRequestResource(
        object sender,
        TsgcAI_MCP_Session session,
        TsgcAI_MCP_Request_ResourcesRead request,
        TsgcAI_MCP_Response_ResourcesRead response)
    {
        response.Result.Contents.Clear();
        if (string.Equals(request.Params.Uri, "metrics://daily", StringComparison.OrdinalIgnoreCase))
        {
            response.Result.Contents.AddContentText(
                "metrics://daily",
                "DailyMetrics",
                "Daily KPI snapshot",
                "application/json",
                BuildMetricsPayload(DateTime.Today));
            return;
        }
        response.Result.IsError = true;
        response.Result.Contents.AddContentText(
            request.Params.Uri,
            "UnknownResource",
            "Not found",
            "text/plain",
            "Resource is not published by this server.");
    }
}

```

MCP Server | Roots

Le radici indicano le posizioni nel filesystem che un client consente al server di esplorare. Ogni voce è un URI `file://` con un nome visualizzato opzionale, che permette alle automazioni MCP di rispettare i confini del progetto mentre l'utente mantiene il controllo su ciò che viene condiviso. I server interrogano il catalogo emettendo `roots/list` e, quando il client annuncia `listChanged`, ricevono `notifications/roots/list_changed` per aggiornare le viste memorizzate nella cache.

Poiché le root appartengono al client, il server agisce sempre come richiedente. La capability `roots` deve essere presente nel payload di inizializzazione del client prima di tentare una chiamata di discovery; altrimenti `RequestRootsList` genera `CapabilityNotSupported`.

Richiesta del catalogo radice

`TsgcWSAPIServer_MCP` espone un helper che serializza `roots/list` e instrada la risposta al codice:

- **Helper per le richieste:** chiamare `RequestRootsList(Session)` una volta che la sessione MCP annuncia la capacità `roots`. Il metodo assegna automaticamente un identificatore (a meno che non se ne fornisca uno) e mantiene la richiesta in sospeso in modo che la risposta possa essere correlata.
- **Evento di risposta:** gestire `OnMCPResponseRootsList(Sender, Session, Request, Response)`. L'evento viene generato quando il client risponde con le directory disponibili. Ispezionare `Response.Roots` o registrare `Response.Write` per esaminare il payload JSON.
- **Rilevamento modifiche:** se il client invia successivamente `notifications/roots/list_changed`, inviare un'altra `RequestRootsList` in modo che la cache lato server rimanga sincronizzata.

Codice di esempio

I frammenti di codice seguenti richiedono le radici non appena viene stabilita una sessione e registrano le voci restituite.

```
public partial class FRMMCPServer : Form
{
    public FRMMCPServer()
    {
        InitializeComponent();
        MCPServer.OnMCPSessionNew += MCPServer_SessionNew;
        MCPServer.OnMCPResponseRootsList += MCPServer_RootsList;
    }

    private void MCPServer_SessionNew(object sender, TsgcAI_MCP_Session session)
    {
        if (session.ClientCapabilities.Roots.Enabled)
        {
            var requestId = MCPServer.RequestRootsList(session);
            memoLog.AppendText($"roots/list sent id {requestId}\r\n");
        }
    }

    private void MCPServer_RootsList(
        object sender,
        TsgcAI_MCP_Session session,
        TsgcAI_MCP_Request_RootsList request,
        TsgcAI_MCP_Response_RootsList response)
    {
        memoLog.AppendText($"roots/list returned {response.Roots.Count} entries\r\n");
        memoLog.AppendText("roots/list payload " + response.Write() + "\r\n");
    }
}
```

MCP Server | Campionamento

Il campionamento consente al server di chiedere al client di eseguire una generazione LLM per suo conto. Le richieste descrivono una cronologia della conversazione, un prompt di sistema opzionale, preferenze per il modello e budget di token. Il client mantiene il controllo sul provider utilizzato, esegue eventuali revisioni human-in-the-loop e infine restituisce il completamento approvato.

I client devono dichiarare la capacità sampling durante initialize; altrimenti un tentativo di campionamento viene rifiutato con un errore di capacità. La specifica incoraggia l'approvazione dell'utente e consente blocchi di contenuto testo, immagine o audio all'interno della cronologia conversazionale.

Emissione di una richiesta di sampling

TsgcWSAPIServer_MCP semplifica il flusso sampling/createMessage con un helper e un evento di risposta:

- **Helper per le richieste:** costruire un `TsgcAI_MCP_Request_SamplingCreateMessage`, popolare `Params.Messages`, `Params.ModelPreferences`, `Params.SystemPrompt` e facoltativamente `Params.MaxTokens`. Passarlo a `RequestSamplingCreateMessage(Session, Request)`; il metodo aggiunge un id (se necessario) e registra la chiamata in sospeso.
- **Evento di risposta:** gestire `OnMCPResponseSamplingCreateMessage(Sender, Session, Request, Response)`. Esaminare `Response.Role`, `Response.Content`, `Response.Model` e `Response.StopReason` per continuare il flusso di lavoro.
- **Gestione degli errori:** le richieste rifiutate o annullate arrivano come errori JSON-RPC, quindi è necessario racchiudere il metodo helper in try/except e monitorare `OnMCPException` per la registrazione.

Codice di esempio

Gli snippet seguenti richiedono un breve riepilogo al client e stampano la risposta dell'assistant.

```
public partial class FRMMCPServer : Form
{
    public FRMMCPServer()
    {
        InitializeComponent();
        MCPServer.OnMCPResponseSamplingCreateMessage += MCPServer_SamplingResponse;
    }
    private void RequestTicketSummary(TsgcAI_MCP_Session session, string ticketText)
    {
        if (!session.ClientCapabilities.Sampling.Enabled)
            return;
        var request = new TsgcAI_MCP_Request_SamplingCreateMessage();
        request.Params.Messages.AddTextMessage(
            "user",
            $"Summarise the following ticket: {ticketText}"
        );
        request.Params.SystemPrompt = "You are a concise support assistant.";
        request.Params.ModelPreferences.Hints.AddHint("claude-3-sonnet");
        request.Params.ModelPreferences.SpeedPriority = 0.6;
        request.Params.ModelPreferences.IntelligencePriority = 0.8;
        request.Params.HasMaxTokens = true;
        request.Params.MaxTokens = 200;
        var requestId = MCPServer.RequestSamplingCreateMessage(session, request);
        memoLog.AppendText($"sampling/createMessage id {requestId}\r\n");
    }
    private void MCPServer_SamplingResponse(
        object sender,
        TsgcAI_MCP_Session session,
        TsgcAI_MCP_Request_SamplingCreateMessage request,
        TsgcAI_MCP_Response_SamplingCreateMessage response)
    {
        if (response.Content is TsgcAI_MCP_Response_Result_Content_Text text)
        {
            memoLog.AppendText($"assistant: {text.Text}\r\n");
        }
        memoLog.AppendText($"model: {response.Model} reason: {response.StopReason}\r\n");
    }
}
```

MCP Server | Elicitation

L'elicitazione consente al server di chiedere al client di raccogliere input strutturato dall'utente durante il flusso di lavoro. La richiesta contiene un messaggio comprensibile all'utente più uno schema JSON vincolato che descrive i campi da acquisire. Il client visualizza un modulo appropriato, valida la risposta localmente e restituisce se l'utente ha accettato, rifiutato o annullato il prompt.

Il client deve dichiarare la capability elicitation durante l'initialize. Gli schemi sono intenzionalmente limitati a oggetti piatti con proprietà primitive (stringa, numero, booleano o enum) per semplificare il rendering e mantenere i dati sensibili fuori dall'ambito.

Richiesta di input dall'utente

TsgcWSAPIServer_MCP fornisce un helper per elicitation/create e genera un evento di risposta quando l'utente finisce di interagire:

- **Helper della richiesta:** creare `TsgcAI_MCP_Request_ElicitationCreate`, impostare `Params._Message` e descrivere il payload atteso tramite `Params.RequestedSchema`. Chiamare `RequestElicitationCreate(Session, Request)` per inviarlo al client.
- **Evento di risposta:** gestire `OnMCPResponseElicitationCreate(Sender, Session, Request, Response)`. Ispezionare `Response.Action` (accept, decline o cancel) e analizzare `Response.Content` quando l'utente ha accettato.
- **Validazione:** le richieste rifiutate arrivano come errori JSON-RPC. Verificare sempre `Action` prima di consumare qualsiasi contenuto e prepararsi a ripetere la richiesta se l'utente ha annullato.

Codice di esempio

Gli snippet seguenti richiedono informazioni di contatto, attendono la risposta e registrano l'azione risultante più il payload.

```
public partial class FRMMCPServer : Form
{
    public FRMMCPServer()
    {
        InitializeComponent();
        MCPServer.OnMCPResponseElicitationCreate += MCPServer_ElicitationResponse;
    }
    private void RequestContactDetails(TsgcAI_MCP_Session session)
    {
        if (!session.ClientCapabilities.Elicitation.Enabled)
            return;
        var request = new TsgcAI_MCP_Request_ElicitationCreate();
        request.Params._Message = "Please confirm your contact information.";
        request.Params.RequestedSchema.Title = "Contact information";
        request.Params.RequestedSchema.Description =
            "Used to keep you updated about this ticket.";
        var name = request.Params.RequestedSchema.Properties.AddProperty("name");
        name._Type = "string";
        name.Title = "Full name";
        name.Required = true;
        var email = request.Params.RequestedSchema.Properties.AddProperty("email");
        email._Type = "string";
        email.Title = "Email address";
        email.Format = "email";
        email.Required = true;
        var updates = request.Params.RequestedSchema.Properties.AddProperty("updates");
        updates._Type = "boolean";
        updates.Title = "Receive status updates";
        updates.Description = "Tick to receive notifications when the ticket changes.";
        updates.DefaultBoolean = false;
        var requestId = MCPServer.RequestElicitationCreate(session, request);
        memoLog.AppendText($"elicitation/create id {requestId}\r\n");
    }
    private void MCPServer_ElicitationResponse(
        object sender,
        TsgcAI_MCP_Session session,
        TsgcAI_MCP_Request_ElicitationCreate request,
        TsgcAI_MCP_Response_ElicitationCreate response)
    {
        memoLog.AppendText($"elicitation action: {response.Action}\r\n");
        if (string.Equals(response.Action, "accept", StringComparison.OrdinalIgnoreCase))
        {
            // ...
        }
    }
}
```



```
        && response.Content != null)
    {
        memoLog.AppendText("payload: " + response.Content.Text + "\r\n");
    }
}
```

OpenAI

OpenAI è un laboratorio di ricerca privato che mira a sviluppare e orientare l'intelligenza artificiale (IA) in modi che avvantaggino l'umanità nel suo insieme. OpenAI ha sviluppato i seguenti progetti:

- **GPT-3:** Questo potente modello linguistico costituisce la base di altri prodotti OpenAI. Analizza testo generato da esseri umani per imparare a generare testo simile autonomamente.
- **DALL-E e DALL-E 2:** Queste piattaforme di IA generativa possono analizzare le descrizioni testuali delle immagini che gli utenti desiderano produrre e generare quindi quelle immagini esattamente come descritto.
- **CLIP:** CLIP è una rete neurale che sintetizza contenuti visivi e testo ad essi correlati per prevedere le migliori didascalie possibili che descrivano con maggiore accuratezza tali contenuti visivi. Grazie alla sua capacità di apprendere da più di un tipo di dato (sia immagini che testo), può essere classificata come AI multimodale.
- **ChatGPT:** ChatGPT è attualmente il chatbot AI più avanzato, progettato per generare testo simile a quello umano e produrre risposte alle domande degli utenti. Addestrato su grandi set di dati, è in grado di generare risposte nel modo in cui lo farebbe un essere umano.
- **Codex:** Codex è stato addestrato su miliardi di righe di codice in vari linguaggi di programmazione per aiutare gli sviluppatori software a semplificare i processi di codifica. Si basa sulla tecnologia GPT-3, ma anziché generare testo, genera codice.
- **Whisper:** Whisper è uno strumento di riconoscimento vocale automatico (ASR). È stato addestrato su una moltitudine di dati audio per riconoscere, trascrivere e tradurre il parlato in circa 100 lingue diverse, incluso il linguaggio tecnico e diversi accenti.

OpenAI API

L'API OpenAI può essere applicata a praticamente qualsiasi attività che implichi la comprensione o la generazione di linguaggio naturale, codice o immagini. OpenAI offre una gamma di modelli con diversi livelli di potenza adatti a diversi compiti, nonché la possibilità di perfezionare i propri modelli personalizzati. Questi modelli possono essere utilizzati per tutto, dalla generazione di contenuti alla ricerca semantica e alla classificazione.

Usi più comuni

- **Completamento**
 - [Esempi di Completamento OpenAI](#)
- **Chat**
 - [Esempi di Chat OpenAI](#)
- **Modifica**
 - [Esempi di modifica OpenAI](#)
- **Audio**
 - [Esempi di trascrizione e traduzione OpenAI](#)
- **Moderazione**
 - [Esempi di Moderazione OpenAI](#)
- **RealTime**
 - [Esempi OpenAI RealTime](#)
- **Risposte**
 - [Esempi di Risposte OpenAI](#)
- **Speech**
 - [Esempi OpenAI Speech](#)
- **Fine-Tuning**
 - [Esempi di Fine-Tuning OpenAI](#)
- **Batch**

- [Esempi di Batch OpenAI](#)
- Uploads
 - [Esempi di caricamento OpenAI](#)

Configurazione

OpenAI

L'API OpenAI utilizza chiavi API per l'autenticazione. Visitare la pagina delle [Chiavi API](#) per recuperare la chiave API da utilizzare nelle richieste.

Si ricordi che la Sua chiave API è un segreto! Non la condivida con altri e non la esponga in alcun codice lato client (browser, app). Le richieste di produzione devono essere instradate attraverso il Suo server backend, dove la chiave API può essere caricata in modo sicuro da una variabile d'ambiente o da un servizio di gestione delle chiavi.

Questa **API Key** deve essere configurata nella proprietà **OpenAIOptions.ApiKey** del componente. Facoltativamente, per gli utenti che appartengono a più organizzazioni, è possibile impostare la propria organizzazione nella proprietà **OpenAIOptions.Organization** se il proprio account appartiene a un'organizzazione.

Una volta configurata la chiave API, di seguito è riportato un elenco delle funzioni disponibili per interagire con l'API OpenAI.

Azure

Il client supporta i servizi Microsoft Azure OpenAI, quindi è possibile utilizzare il proprio account Azure per interagire anche con le API Azure OpenAI. Per configurare il client per lavorare con Azure, seguire i passaggi indicati di seguito:

1. Configuri la proprietà **OpenAIOptions.Provider** = oapvAzure
2. Impostare i valori di **ResourceName** e **DeploymentId** (questi valori si trovano nel proprio account Azure)
 1. **OpenAIOptions.AzureOptions.ResourceName** = <nome della risorsa>.
 2. **OpenAIOptions.AzureOptions.DeploymentId** = <id del deployment>.
3. Impostare la API Key del proprio Account Azure
 1. **OpenAIOptions.ApiKey** = <azure api key>.

Tenga presente che non tutti i metodi OpenAI sono supportati da Azure, attualmente sono supportati solo i seguenti metodi:

1. Completamento
2. Chat Completion

Proprietà

OpenAIOptions

- **ApiKey:** La chiave API per l'autenticazione con l'API di OpenAI.
- **Organization:** ID organizzazione facoltativo per gli utenti che appartengono a più organizzazioni.
- **Provider:** Selezionare il provider: oapvOpenAI (predefinito) o oapvAzure per Microsoft Azure OpenAI Services.
- **AzureOptions:** Configurazione per i servizi Azure OpenAI.
 - **ResourceName:** Il nome della risorsa Azure.
 - **DeploymentId:** L'ID di deployment Azure.
 - **APIVersion:** La versione dell'API Azure.
- **HttpOptions:** Impostazioni di connessione HTTP.
 - **ReadTimeout:** Timeout in millisecondi per la lettura delle risposte HTTP. Il valore predefinito è 0 (nessun timeout).
- **LogOptions:** Configurazione del logging.
 - **Enabled:** Quando True, le richieste e le risposte HTTP vengono registrate in un file.
 - **FileName:** Il percorso del file in cui viene scritto l'output del log.
- **RetryOptions:** Configurazione del retry automatico per le richieste API non riuscite.
 - **Enabled:** Quando è True, le richieste non riuscite vengono ritentate automaticamente.
 - **Retries:** Numero massimo di tentativi di retry.
 - **Wait:** Tempo di attesa in millisecondi tra i tentativi di retry.

Models

Elencare e descrivere i vari modelli disponibili nell'API.

- **GetModels:** Elenca i modelli attualmente disponibili e fornisce informazioni di base su ciascuno, come il proprietario e la disponibilità.
- **GetModel:** Recupera un'istanza del modello, fornendo informazioni di base sul modello come il proprietario e i permessi.
 - **Model:** L'ID del modello da utilizzare per questa richiesta

Completamenti

Dato un prompt, il modello restituirà uno o più completamenti previsti e potrà anche restituire le probabilità di token alternativi in ciascuna posizione.

- **CreateCompletion:** Crea un completamento per il prompt e i parametri forniti
 - **Model:** ID del modello da utilizzare. È possibile usare l'API List models per vedere tutti i modelli disponibili, oppure consultare la panoramica dei modelli per le relative descrizioni.
 - **Prompt:** Il prompt per generare i completamenti.

Chat

Data una conversazione in chat, il modello restituirà una risposta di completamento della chat.

- **Model:** ID del modello da utilizzare. Chiamare GetModels per ottenere un elenco di tutti i modelli supportati dall'API Chat.
- **Message:** Il messaggio per cui generare i completamenti della chat.
- **Role:** per impostazione predefinita user, le altre opzioni sono: system, assistant.

Modifiche

Dato un prompt e un'istruzione, il modello restituirà una versione modificata del prompt.

- **CreateEdit:** Crea una nuova modifica per l'input, l'istruzione e i parametri forniti.
 - **Model:** ID del modello da utilizzare. È possibile usare il modello text-davinci-edit-001 o code-davinci-edit-001 con questo endpoint.
 - **Instruction:** L'istruzione che indica al modello come modificare il prompt.
 - **Input:** (opzionale) Il testo di input da utilizzare come punto di partenza per la modifica.

Immagini

Dato un prompt e/o un'immagine di input, il modello genererà una nuova immagine.

- **CreateImage:** Crea un'immagine dato un prompt.
 - **Prompt:** Una descrizione testuale dell'immagine o delle immagini desiderate. La lunghezza massima è di 1000 caratteri.
- **CreateImageEdit:** Crea un'immagine modificata o estesa data un'immagine originale e un prompt.
 - **Image:** L'immagine da modificare. Deve essere un file PNG valido, di dimensioni inferiori a 4MB e quadrato. Se non viene fornita una maschera, l'immagine deve avere trasparenza, che verrà utilizzata come maschera.
 - **Prompt:** Una descrizione testuale dell'immagine o delle immagini desiderate. La lunghezza massima è di 1000 caratteri.
- **CreateImageVariations:** Crea una variazione di un'immagine fornita.
 - **Image:** L'immagine da utilizzare come base per la/le variazione/i. Deve essere un file PNG valido, di dimensioni inferiori a 4 MB e quadrato.

Embeddings

Ottenere una rappresentazione vettoriale di un dato input facilmente utilizzabile da modelli e algoritmi di machine learning.

- **CreateEmbeddings:** Crea un vettore di embedding che rappresenta il testo di input.
 - **Model:** ID del modello da utilizzare.
 - **Input:** Testo di input per ottenere gli embedding.

Audio

Trasforma l'audio in testo.

- **CreateTranscriptionFromFile:** Trascrive l'audio nella lingua di input da un nome file
 - **Model:** ID del modello da utilizzare. Attualmente è disponibile solo whisper-1.
 - **Filename:** Il file audio da trascrivere, in uno dei seguenti formati: mp3, mp4, mpeg, mpga, m4a, wav o webm.
- **CreateTranscription:** Registra audio per X secondi e lo trascrive.
 - **Model:** ID del modello da utilizzare. Attualmente è disponibile solo whisper-1.
 - **Time:** tempo in millisecondi, per impostazione predefinita 10 secondi.
- **CreateTranslationFromFile:** Traduce l'audio in inglese.
 - **Model:** ID del modello da utilizzare. Attualmente è disponibile solo whisper-1.
 - **Filename:** Il file audio da tradurre, in uno dei seguenti formati: mp3, mp4, mpeg, mpga, m4a, wav o webm.
- **CreateTranslation:** Registra audio per X secondi e lo traduce.
 - **Model:** ID del modello da utilizzare. Attualmente è disponibile solo whisper-1.
 - **Time:** durata in millisecondi, Per impostazione predefinita 10 secondi.

File

I file vengono utilizzati per caricare documenti che possono essere impiegati con funzionalità come il Fine-tuning.

- **ListFiles:** Restituisce un elenco dei file appartenenti all'organizzazione dell'utente.
- **UploadFile:** Carica un file contenente uno o più documenti da utilizzare in vari endpoint/funzionalità. Attualmente, la dimensione totale di tutti i file caricati da un'organizzazione può essere fino a 1 GB.
 - **Filename:** Nome del file JSON Lines da caricare. Se lo scopo è impostato su "fine-tune", ogni riga è un record JSON con i campi "prompt" e "completion" che rappresentano gli esempi di addestramento.
 - **Purpose:** Lo scopo previsto dei documenti caricati. Utilizzare "fine-tune" per il Fine-tuning.
- **DeleteFile:** Elimina un file.
 - **FileId:** L'ID del file da utilizzare per questa richiesta
- **RetrieveFile:** Restituisce informazioni su un file specifico.
 - **FileId:** L'ID del file da utilizzare per questa richiesta
- **RetrieveFileContent:** Restituisce il contenuto del file specificato
 - **FileId:** L'ID del file da utilizzare per questa richiesta.

Fine-Tunes

Gestisce i job di fine-tuning per adattare un modello ai Suoi dati di addestramento specifici.

- **CreateFineTune:** Crea un job che esegue il fine-tuning di un modello specificato a partire da un dataset fornito. La risposta include i dettagli del job accodato, incluso lo stato del job e il nome dei modelli ottimizzati una volta completato.
 - **TrainingFile:** L'ID di un file caricato che contiene i dati di addestramento.
- **ListFineTunes:** Elenca i job di fine-tuning della Sua organizzazione

- **RetrieveFineTune**: Ottiene informazioni sul job di fine-tuning.
 - **FineTuneId**: L'ID del job di fine-tuning
- **CancelFineTune**: Annulla immediatamente un job di fine-tuning.
 - **FineTuneId**: L'ID del job di fine-tuning
- **ListFineTuneEvents**: Ottieni aggiornamenti dettagliati sullo stato di un job di fine-tuning.
 - **FineTuneId**: L'ID del job di fine-tuning
- **DeleteFineTuneModel**: Elimina un modello fine-tuned. È necessario avere il ruolo di Owner nella propria organizzazione.
 - **Model**: Il modello da eliminare.

Moderazioni

Dato un testo in input, indica se il modello lo classifica come violazione della policy sui contenuti di OpenAI.

- **CreateModeration**: Classifica se un testo viola la Content Policy di OpenAI
 - **Input**: Il testo di input da classificare

RealTime

L'API OpenAI Realtime abilita interazioni multimodali a bassa latenza, incluse esperienze conversazionali speech-to-speech e trascrizione in tempo reale.

- **Trascrizione**: È possibile utilizzare la Realtime API per casi d'uso di sola trascrizione, sia con input da un microfono sia da un file. Ad esempio, è possibile utilizzarla per generare sottotitoli o trascrizioni in tempo reale. Con la modalità di sola trascrizione, il modello non genererà risposte.

Assistenti

Crea assistenti AI in grado di chiamare modelli e utilizzare strumenti per eseguire attività.

- **CreateAssistant**: Crea un assistente con un modello e istruzioni.
- **ListAssistants**: Restituisce un elenco di assistenti.
- **RetrieveAssistant**: Recupera un assistente tramite ID.
- **ModifyAssistant**: Modifica un assistente esistente.
- **DeleteAssistant**: Elimina un assistant.

Thread

I thread rappresentano una sessione di conversazione. I messaggi vengono aggiunti ai thread, che vengono poi elaborati dalle esecuzioni.

- **CreateThread**: crea un nuovo thread.
- **RetrieveThread**: Recupera un thread tramite ID.
- **ModifyThread**: Modifica un thread.
- **DeleteThread**: Elimina un thread.

Messaggi del thread

I messaggi vengono aggiunti ai thread e contengono il contenuto di una conversazione.

- **CreateMessage**: Crea un messaggio all'interno di un thread.
 - **ThreadId**: L'ID del thread.

- **ListMessages:** Restituisce un elenco di messaggi per un dato thread.
- **RetrieveMessage:** Recupera un messaggio tramite il thread e l'ID del messaggio.
- **ModifyMessage:** Modifica un messaggio.
- **DeleteMessage:** Elimina un messaggio da un thread.

Runs

Le esecuzioni rappresentano un'esecuzione su un thread con un assistente. L'assistente utilizza la propria configurazione e i messaggi del thread per eseguire compiti chiamando modelli e strumenti.

- **CreateRun:** Crea un'esecuzione per un thread con un assistente specificato.
- **ListRuns:** Restituisce un elenco di run appartenenti a un thread.
- **RetrieveRun:** Recupera un'esecuzione tramite ID thread e ID esecuzione.
- **ModifyRun:** Modifica una run.
- **SubmitToolOutputsToRun:** Invia gli output degli strumenti per un run che richiede un'azione.
- **CancelRun:** Annulla un'esecuzione in corso.

Passaggi di esecuzione

I passaggi di esecuzione rappresentano i singoli passi effettuati durante l'esecuzione di un run.

- **ListRunSteps:** Restituisce un elenco dei passaggi di esecuzione appartenenti a un run.
- **RetrieveRunSteps:** Recupera uno step di esecuzione tramite thread, esecuzione e ID dello step.

Vector Stores

I vector store vengono utilizzati per memorizzare e cercare tra i file utilizzando gli embedding per la generazione aumentata dal recupero (RAG).

- **CreateVectorStore:** Crea un vector store.
- **ListVectorStores:** Restituisce un elenco di vector store.
- **RetrieveVectorStore:** Recupera un vector store tramite ID.
- **ModifyVectorStore:** Modifica un vector store.
- **DeleteVectorStore:** Elimina un vector store.

File dello Store vettoriale

Gestisce i file all'interno dei vector store.

- **CreateVectorStoreFile:** Allega un file a un vector store.
- **ListVectorStoreFiles:** Restituisce un elenco di file in un archivio vettoriale.
- **RetrieveVectorStoreFile:** Recupera un file di un vector store.
- **DeleteVectorStoreFile:** Rimuove un file da un vector store.

Batch di File del Vector Store

Operazioni batch per l'aggiunta di file ai vector store.

- **CreateVectorStoreFileBatch:** Crea un batch per aggiungere più file a un vector store.
- **RetrieveVectorStoreFileBatch:** Recupera un batch di file tramite ID.
- **CancelVectorStoreFileBatch:** Annulla un batch di file in corso.
- **ListVectorStoreFilesBatch:** Elenca i file in un batch.

Discorso

Genera audio parlato dal testo utilizzando modelli di sintesi vocale.

- **CreateSpeech:** Genera audio dal testo di input. Restituisce i dati audio come stringa o li scrive in uno stream di risposta.
 - **Model:** Il modello TTS da utilizzare (ad es. tts-1, tts-1-hd).
 - **Input:** Il testo per cui generare l'audio.
 - **Voice:** La voce da utilizzare (alloy, echo, fable, onyx, nova, shimmer).

Job di Fine-Tuning

Gestisce i job di fine-tuning per creare modelli personalizzati. Questa è la nuova API di fine-tuning che sostituisce il precedente endpoint Fine-Tunes.

- **CreateFineTuningJob:** Crea un job di fine-tuning da un file di addestramento.
- **ListFineTuningJobs:** Elenca i job di fine-tuning dell'organizzazione.
- **RetrieveFineTuningJob:** Recupera informazioni su un job di fine-tuning.
- **CancelFineTuningJob:** annulla un job di fine-tuning.
- **ListFineTuningJobEvents:** Elenca gli aggiornamenti di stato per un job di fine-tuning.
- **ListFineTuningJobCheckpoints:** Elenca i checkpoint per un job di fine-tuning.

Risposte

Crea e gestisce le risposte del modello. La Responses API supporta conversazioni multi-turno, utilizzo di strumenti e output strutturati.

- **CreateResponse:** Crea una risposta del modello.
- **RetrieveResponse:** Recupera una risposta tramite ID.
- **DeleteResponse:** Elimina una risposta.
- **CancelResponse:** Annulla una risposta in corso.
- **ListInputItems:** Elenca gli elementi di input per una risposta.

Batch

Creare e gestire richieste API batch per l'elaborazione asincrona a costo ridotto.

- **CreateBatch:** Crea un lavoro batch.
- **RetrieveBatch:** Recupera un batch tramite ID.
- **ListBatches:** Elenca tutti i batch.
- **CancelBatch:** Annulla un batch in corso.

Uploads

Carica file di grandi dimensioni in parti. Utile per file che superano il limite di dimensione di upload standard.

- **CreateUpload:** Crea una sessione di caricamento.
- **AddUploadPart:** Aggiunge una parte a un upload.
- **CompleteUpload:** Completa un upload multi-parte e crea un file.
- **CancelUpload:** Annulla un caricamento in corso.

OpenAI | Completion

Dato un prompt, il modello restituirà uno o più completamenti previsti, e può anche restituire le probabilità dei token alternativi in ogni posizione.

Esempio Semplice

Utilizzare il modello text-davinci-003 per ottenere un completamento previsto.

```
TsgcHTTP_API_OpenAI OpenAI = new TsgcHTTP_API_OpenAI();
OpenAI.OpenAIOptions.ApiKey = "API_KEY";
MessageBox.Show(OpenAI._CreateCompletion("text-davinci-003", "Say this is a test"));
```

Esempio avanzato

Utilizzare il modello text-davinci-003 per ottenere un completamento previsto con output più casuale e generare 2 completamenti per ogni prompt.

```
TsgcHTTP_OpenAI_JSON OpenAI = new TsgcHTTP_OpenAI_JSON();
OpenAI.OpenAIOptions.ApiKey = "API_KEY";

TsgcOpenAIClass_Request_Completion oRequest = new TsgcOpenAIClass_Request_Completion();
oRequest.Model = "text-davinci-003";
oRequest.Prompt = "Say this is a test";
oRequest.Temperature = 1;
oRequest.N = 2;
TsgcOpenAIClass_Response_Completion oResponse = OpenAI.CreateCompletion(oRequest);

if (oResponse.Choices.Length > 0)
{
    MessageBox.Show(oResponse.Choices[0].Text);
}
```

OpenAI | Chat

Data una conversazione in chat, il modello restituirà una risposta di completamento della chat.

Esempio Semplice

Interagire con ChatGPT inviando un messaggio di saluto.

```
TsgcHTTP_API_OpenAI OpenAI = new TsgcHTTP_API_OpenAI();
OpenAI.OpenAIOptions.ApiKey = "API_KEY";
MessageBox.Show(OpenAI._CreateChatCompletion("gpt-3.5-turbo", "Hello!"));
```

Esempio avanzato

Utilizzi il modello gpt-3-5 per chattare con un output più casuale e generare 2 completion per ogni prompt.

```
TsgcHTTP_OpenAI_JSON OpenAI = new TsgcHTTP_OpenAI_JSON();
OpenAI.OpenAIOptions.ApiKey = "API_KEY";

TsgcOpenAIClass_Request_ChatCompletion oRequest = new TsgcOpenAIClass_Request_ChatCompletion();
oRequest.Model = "gpt-3.5-turbo";
TsgcOpenAIClass_Request_Completion_Message oMessage = new TsgcOpenAIClass_Request_Completion_Message();
oMessage.Role = "user";
oMessage.Content = "Hello!";
oMessages = oRequest.Messages;
SetLength(oMessages, 1);
oMessages[0] = oMessage;
oRequest.Messages = oMessages;
oRequest.Temperature = 1;
oRequest.N = 2;
TsgcOpenAIClass_Response_ChatCompletion oResponse = OpenAI.CreateChatCompletion(oRequest);

if (oResponse.Choices.Length > 0)
{
    MessageBox.Show(oResponse.Choices[0]._Message.Content);
}
```

OpenAI | Modifica

Dato un prompt e un'istruzione, il modello restituirà una versione modificata del prompt.

Esempio Semplice

Dice a OpenAI di correggere gli errori di ortografia di un prompt.

```
TsgcHTTP_API_OpenAI OpenAI = new TsgcHTTP_API_OpenAI();
OpenAI.OpenAIOptions.ApiKey = "API_KEY";
MessageBox.Show(OpenAI._CreateEdit("text-davinci-edit-001", "Fix the spelling mistakes", "What day of the wek is
```

Esempio avanzato

Chiedere a OpenAI di correggere gli errori ortografici di un prompt, con output più casuale e generare 2 completamenti per ogni prompt.

```
TsgcHTTP_OpenAI_JSON OpenAI = new TsgcHTTP_OpenAI_JSON();
OpenAI.OpenAIOptions.ApiKey = "API_KEY";

TsgcOpenAIClass_Request_Edit oRequest = new TsgcOpenAIClass_Request_Edit();
oRequest.Model = "text-davinci-edit-001";
oRequest.Input = "What day of the wek is it?";
oRequest.Instruction = "Fix the spelling mistakes";
oRequest.Temperature = 1;
oRequest.N = 2;
TsgcOpenAIClass_Response_Edit oResponse = OpenAI.CreateEdit(oRequest);

if (oResponse.Choices.Length > 0)
{
    MessageBox.Show(oResponse.Choices[0].Text);
}
```

OpenAI | Audio

Crea Trascrizione

Trascrive l'audio nella lingua di input.

```
TsgcHTTP_API_OpenAI OpenAI = new TsgcHTTP_API_OpenAI();  
OpenAI.OpenAIOptions.ApiKey = "API_KEY";  
  
MessageBox.Show(OpenAI._CreateTranscriptionFromFile("whisper-1", "c:\\media\\audio.mp3"));
```

Create Translation

Traduce l'audio in inglese.

```
TsgcHTTP_API_OpenAI OpenAI = new TsgcHTTP_API_OpenAI();  
OpenAI.OpenAIOptions.ApiKey = "API_KEY";  
  
MessageBox.Show(OpenAI._CreateTranslationFromFile("whisper-1", "c:\\media\\audio.mp3"));
```

OpenAI | Moderation

Dato un testo in ingresso, restituisce se il modello lo classifica come violazione della policy sui contenuti di OpenAI.

Esempio Semplice

Moderi il seguente testo

```
TsgcHTTP_API_OpenAI OpenAI = new TsgcHTTP_API_OpenAI();
OpenAI.OpenAIOptions.ApiKey = "API_KEY";

MessageBox.Show(OpenAI._CreateModeration("I want to kill them."));
```

Esempio avanzato

Moderare il seguente testo scegliendo il modello.

```
TsgcHTTP_OpenAI_JSON OpenAI = new TsgcHTTP_OpenAI_JSON();
OpenAI.OpenAIOptions.ApiKey = "API_KEY";

TsgcOpenAIClass_Request_Moderation oRequest = new TsgcOpenAIClass_Request_Moderation();
oRequest.Model = "text-moderation-latest";
oRequest.Input = "I want to kill them.";
TsgcOpenAIClass_Response_Moderation oResponse = OpenAI.CreateModeration(oRequest);

if (oResponse.results.Length > 0)
{
    MessageBox.Show(oResponse.results[0].flagged.ToString());
}
```

OpenAI | RealTime

È possibile utilizzare la Realtime API per casi d'uso di sola trascrizione, sia con input da un microfono che da un file. Ad esempio, può utilizzarla per generare sottotitoli o trascrizioni in tempo reale. Con la modalità di sola trascrizione, il modello non genera risposte.

Per utilizzare l'API Realtime per la trascrizione, è necessario creare una sessione di trascrizione, collegandosi tramite WebSockets. Utilizzare il componente [TsgcWSAPI_OpenAI](#) e [TsgcWebSocketClient](#) per avviare una nuova sessione di trascrizione.

Di seguito è riportato un esempio di trascrizione in tempo reale tramite OpenAI API.

```
var wsClient = new TsgcWebSocketClient(null);
var audioRecorder = new TsgcAudioRecorderWave(null);
var openAI = new TsgcWSAPI_OpenAI(null);
openAI.Client = wsClient;
openAI.AudioRecorder = audioRecorder;
openAI.OpenAIOptions.APIKey = "your-api-key-here";
openAI.OpenAIOptions.Method = OpenAIMethod.Transcription;
openAI.OpenAIOptions.Provider = OpenAIProvider.OpenAI;
openAI.InputAudio.Language = "en";
openAI.InputAudio.Model = "whisper-1";
openAI.AudioTranscriptionCompleted += OnAudioTranscriptionCompleted;
private void OnAudioTranscriptionCompleted(object sender, TsgcWSOpenAIConversation_Item_Completed aItem)
{
    Console.WriteLine("#transcription_completed: " + aItem.Transcript);
}
```

Invia audio manualmente

Il componente consente di inviare file audio manualmente. È possibile utilizzare il metodo `AppendInputAudioBuffer` e passare l'audio come `TStream`. Il formato audio deve essere PCM mono a 24 kHz (è supportata solo una frequenza di 24000).

OpenAI | Risposte

L'API Responses è l'API principale per interagire con i modelli OpenAI. Supporta input testuali e di immagini, uso di strumenti, streaming e output strutturati.

Esempio Semplice

Crea una risposta utilizzando la Responses API.

```
TsgcHTTP_API_OpenAI OpenAI = new TsgcHTTP_API_OpenAI();
OpenAI.OpenAIOptions.ApiKey = "API_KEY";
MessageBox.Show(OpenAI._CreateResponse("gpt-4o", "What is the capital of France?"));
```

Metodi

- **CreateResponse:** Crea una risposta del modello dati un modello e un input.
 - **Model:** ID del modello da utilizzare (ad es. gpt-4o, gpt-4o-mini).
 - **Input:** Testo o input strutturato per il modello.
- **RetrieveResponse:** Recupera una risposta precedentemente creata tramite il suo ID.
 - **ResponseId:** L'ID della risposta da recuperare.
- **DeleteResponse:** Elimina una risposta tramite il suo ID.
 - **ResponseId:** L'ID della risposta da eliminare.
- **CancelResponse:** Annulla una risposta in corso.
 - **ResponseId:** L'ID della risposta da annullare.
- **ListInputItems:** Restituisce un elenco di elementi di input per una determinata risposta.
 - **ResponseId:** L'ID della risposta.

OpenAI | Speech

Genera audio parlato dal testo utilizzando l'API Text-to-Speech (TTS). Supporta più voci e formati di output.

Esempio Semplice

Genera audio dal testo e salva in uno stream di file.

```
TsgcHTTP_API_OpenAI OpenAI = new TsgcHTTP_API_OpenAI();
OpenAI.OpenAIOptions.ApiKey = "API_KEY";

TFileStream oStream = new TFileStream("output.mp3", fmCreate);
try
{
    OpenAI._CreateSpeech("tts-1", "Hello, how are you?", "alloy", oStream);
}
finally
{
    oStream.Dispose();
}
```

Metodi

- **CreateSpeech:** Genera audio dal testo di input.
 - **Model:** ID del modello da utilizzare (tts-1 o tts-1-hd).
 - **Input:** Il testo per cui generare l'audio. Massimo 4096 caratteri.
 - **Voice:** La voce da utilizzare (alloy, echo, fable, onyx, nova, shimmer).
 - **ResponseStream:** Lo stream in cui verranno scritti i dati audio.
 - **ResponseFormat:** (opzionale) Il formato audio: mp3 (predefinito), opus, aac o flac.
 - **Speed:** (opzionale) La velocità dell'audio generato (da 0,25 a 4,0, predefinito 1,0).

OpenAI | Fine-Tuning

Gestire i lavori di fine-tuning per adattare un modello ai propri dati di addestramento specifici. Il fine-tuning consente di ottenere di più dai modelli fornendo risultati di qualità superiore, la possibilità di addestrare con più esempi e un risparmio sui costi.

Esempio Semplice

Crea un job di fine-tuning ed elenca i job esistenti.

```
TsgcHTTP_API_OpenAI OpenAI = new TsgcHTTP_API_OpenAI();
OpenAI.OpenAIOptions.ApiKey = "API_KEY";

// Create a fine-tuning job
MessageBox.Show(OpenAI._CreateFineTuningJob("gpt-4o-mini-2024-07-18", "file-abc123"));

// List fine-tuning jobs
MessageBox.Show(OpenAI._ListFineTuningJobs());
```

Metodi

- **CreateFineTuningJob:** Crea un job di fine-tuning che avvia il processo di creazione di un nuovo modello a partire da un dato set di dati.
 - **Model:** Il nome del modello da sottoporre a fine-tuning.
 - **TrainingFile:** L'ID di un file caricato che contiene i dati di addestramento.
- **ListFineTuningJobs:** Elenca i job di fine-tuning della Sua organizzazione.
- **RetrieveFineTuningJob:** Ottiene informazioni su un job di fine-tuning.
 - **JobId:** L'ID del job di fine-tuning.
- **CancelFineTuningJob:** Annulla immediatamente un job di fine-tuning.
 - **JobId:** L'ID del job di fine-tuning da annullare.
- **ListFineTuningJobEvents:** Ottieni aggiornamenti sullo stato di un job di fine-tuning.
 - **JobId:** L'ID del job di fine-tuning.
- **ListFineTuningJobCheckpoints:** Elenca i checkpoint per un job di fine-tuning.
 - **JobId:** L'ID del job di fine-tuning.

OpenAI | Batch

Crea grandi batch di richieste API da eseguire in modo asincrono. La Batch API restituisce i completamenti entro 24 ore con uno sconto del 50%.

Esempio Semplice

Crea un job batch e recupera il suo stato.

```
TsgcHTTP_API_OpenAI OpenAI = new TsgcHTTP_API_OpenAI();
OpenAI.OpenAIOptions.ApiKey = "API_KEY";

// Create a batch
MessageBox.Show(OpenAI._CreateBatch("file-abc123", "/v1/chat/completions"));

// List batches
MessageBox.Show(OpenAI._ListBatches());
```

Metodi

- **CreateBatch:** Crea ed esegue un batch da un file caricato di richieste.
 - **InputFileId:** L'ID di un file caricato che contiene le richieste per il batch.
 - **Endpoint:** L'endpoint da utilizzare per tutte le richieste nel batch (es. /v1/chat/completions).
 - **CompletionWindow:** (opzionale) Il periodo di tempo entro cui il batch deve essere elaborato. Il valore predefinito è 24h.
- **RetrieveBatch:** Recupera un batch tramite il suo ID.
 - **BatchId:** L'ID del batch da recuperare.
- **ListBatches:** Elenca i batch della Sua organizzazione.
- **CancelBatch:** Annulla un batch in corso.
 - **BatchId:** L'ID del batch da annullare.

OpenAI | Upload

Carica file di grandi dimensioni in parti. L'API Uploads consente di caricare file di dimensioni superiori a 512 MB suddividendoli in parti e caricando ciascuna parte separatamente.

Esempio Semplice

Crea un upload, aggiunge una parte e lo completa.

```
TsgcHTTP_API_OpenAI OpenAI = new TsgcHTTP_API_OpenAI();
OpenAI.OpenAIOptions.ApiKey = "API_KEY";

// Create an upload
string vUploadResponse = OpenAI._CreateUpload("training_data.jsonl",
    "fine-tune", 2147483648, "application/jsonl");

// Add a part
string vPartResponse = OpenAI._AddUploadPart("upload_abc123", "part1.jsonl");

// Complete the upload
MessageBox.Show(OpenAI._CompleteUpload("upload_abc123", "["part_abc123"]"));
```

Metodi

- **CreateUpload:** Crea un oggetto Upload intermedio per avviare un upload in più parti.
 - **Filename:** Il nome del file da caricare.
 - **Purpose:** Lo scopo previsto del file caricato (ad es. fine-tune).
 - **Bytes:** Il numero di byte nel file.
 - **MimeType:** Il tipo MIME del file (ad es. application/jsonl).
- **AddUploadPart:** Aggiunge una parte a un oggetto Upload.
 - **UploadId:** L'ID dell'Upload.
 - **Filename:** Il blocco di byte per questa parte.
- **CompleteUpload:** Completa il caricamento. Il file caricato è utilizzabile solo dopo il completamento.
 - **UploadId:** L'ID del caricamento.
 - **PartIds:** L'elenco ordinato degli ID delle parti come stringa di array JSON.
- **CancelUpload:** Annulla il caricamento. Non è possibile aggiungere parti dopo l'annullamento.
 - **UploadId:** L'ID del caricamento da annullare.

OpenAI Audio

Per utilizzare le API OpenAI con comandi vocali, sono necessari i seguenti passaggi:

1. L'audio del microfono deve essere acquisito, quindi è necessario un sistema di riconoscimento vocale per ottenere il testo che verrà inviato a OpenAI.
 1. L'acquisizione dell'audio del microfono viene effettuata tramite il componente TsgcAudioRecorderMCI.
 2. Una volta acquisito l'audio, viene inviato all'API OpenAI Whisper per convertire il file audio in testo.
2. Una volta ottenuto il risultato della sintesi vocale, inviamo il testo a OpenAI utilizzando l'API ChatCompletion.
3. La risposta da OpenAI deve quindi essere convertita in voce utilizzando uno dei seguenti componenti:
 1. TsgcTextToSpeechSystem: (attualmente solo per Windows) utilizza il text-to-speech di Windows del sistema operativo.
 2. TsgcTextToSpeechGoogle: invia la risposta da OpenAI ai server Google Cloud e viene restituito un file mp3 che viene riprodotto dal TsgcAudioPlayerMCI.
 3. TsgcTextToSpeechAmazon: invia la risposta da OpenAI ai server Amazon AWS e viene restituito un file mp3 che viene riprodotto dal TsgcAudioPlayerMCI.

Componenti

I seguenti componenti vengono utilizzati per acquisire audio dal microfono, riprodurre file audio e convertire testo in voce.

- **TsgcAudioRecorderMCI:** (solo per windows) questo componente le consente di accedere al microfono e convertire la voce in un file wave.
- **TsgcAudioPlayerMCI:** (solo per Windows) questo componente consente di riprodurre un file mp3.
- **TsgcTextToSpeechSystem:** (solo per Windows) converte il testo in voce senza la necessità di utilizzare un file mp3 esterno.
- **TsgcTextToSpeechGoogle:** converte il testo in voce utilizzando qualsiasi voce Google Cloud disponibile.
- **TsgcTextToSpeechAmazon:** converte il testo in voce utilizzando una qualsiasi delle voci Amazon AWS disponibili.

Anthropic Claude

Anthropic è un'azienda specializzata nella sicurezza dell'IA che sviluppa sistemi di intelligenza artificiale affidabili, interpretabili e orientabili. La sua famiglia di modelli di punta è Claude, che eccelle nel dialogo ragionato, nella creazione di contenuti, nel ragionamento complesso, nella programmazione e molto altro.

La libreria `sgcWebSockets` fornisce un componente Delphi **TsgcHTTP_API_Anthropic** per interagire con l'API Anthropic Claude.

Anthropic API

L'API Anthropic fornisce accesso ai modelli Claude per la creazione di applicazioni basate sull'intelligenza artificiale. L'API supporta la generazione di testo, la visione (comprensione delle immagini), l'uso di strumenti (function calling), il pensiero esteso, l'elaborazione di documenti/PDF, il prompt caching, le citazioni, la ricerca web, lo streaming, il conteggio dei token e i batch di messaggi.

Funzionalità

- Messaggi
 - [Esempi di messaggi Anthropic](#)
- Vision
 - [Esempi di Vision Anthropic](#)
- Uso degli strumenti
 - [Esempi di Anthropic Tool Use](#)
- Modelli
 - [Esempi di Modelli Anthropic](#)
- Batches
 - [Anthropic Batches Examples](#)
- Extended Thinking
 - [Esempi di Extended Thinking con Anthropic](#)
- Documenti
 - [Esempi di documenti Anthropic](#)
- Prompt Caching
 - [Esempi di Anthropic Prompt Caching](#)
- Citazioni
 - [Esempi di Citazioni Anthropic](#)
- Ricerca Web
 - [Esempi di ricerca Web con Anthropic](#)
- Output strutturati
 - [Esempi di output strutturati Anthropic](#)
- File
 - [Esempi di Anthropic Files API](#)
- MCP Connector
 - [Esempi di Anthropic MCP Connector](#)

Configurazione

L'API Anthropic utilizza chiavi API per l'autenticazione. Visiti la Sua pagina [API Keys](#) nella Anthropic Console per recuperare la chiave API che utilizzerà nelle Sue richieste.

Ricordare che la propria chiave API è segreta! Non condividerla con altri né esporla in alcun codice lato client.

Questa **API Key** deve essere configurata nella proprietà **AnthropicOptions.ApiKey** del componente. La proprietà **AnthropicOptions.AnthropicVersion** specifica la versione dell'API (predefinita: 2023-06-01).

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "YOUR_API_KEY";
```

Proprietà

AnthropicOptions

- **ApiKey:** La chiave API per l'autenticazione con l'API Anthropic.
- **AnthropicVersion:** La stringa della versione API (predefinita: 2023-06-01).
- **BetaHeaders:** Header facoltativi per funzionalità beta, da utilizzare per abilitare funzionalità API in pre-rilascio (es. files-api-2025-04-14, mcp-client-2025-11-20).
- **HttpOptions:** Impostazioni di connessione HTTP.
 - **ReadTimeout:** Timeout in millisecondi per la lettura delle risposte HTTP. Il valore predefinito è 0 (nessun timeout).
- **LogOptions:** Configurazione del logging.
 - **Enabled:** Quando True, le richieste e le risposte HTTP vengono registrate in un file.
 - **FileName:** Il percorso del file in cui viene scritto l'output del log.
- **RetryOptions:** Configurazione del retry automatico per le richieste API non riuscite.
 - **Enabled:** Quando è True, le richieste non riuscite vengono ritentate automaticamente.
 - **Retries:** Numero massimo di tentativi di retry.
 - **Wait:** Tempo di attesa in millisecondi tra i tentativi di retry.

Messaggi

Inviare un elenco strutturato di messaggi di input con contenuto testuale e/o immagini, e il modello genererà il prossimo messaggio nella conversazione.

- **CreateMessage:** Crea un messaggio con il modello e i parametri specificati.
 - **Model:** Il modello da utilizzare (es. claude-sonnet-4-20250514).
 - **Message:** Il contenuto del messaggio dell'utente.
 - **MaxTokens:** Il numero massimo di token da generare (obbligatorio, predefinito 4096).
- **CreateMessageWithSystem:** Crea un messaggio con un prompt di sistema.
 - **System:** Prompt di sistema che definisce il comportamento dell'assistente.
- **CreateMessageStream:** Crea un messaggio con streaming (SSE) abilitato. Gli eventi vengono consegnati tramite il gestore dell'evento OnHTTPAPISSE.
- **CreateMessageJSON:** Crea un messaggio che restituisce JSON strutturato conforme a uno schema JSON fornito.
 - **Schema:** Una stringa JSON Schema che definisce il formato di output.
- **CreateMessageWithThinking:** Crea un messaggio con il pensiero esteso abilitato.
 - **BudgetTokens:** Budget massimo di token per il pensiero (minimo 1024).
- **CreateDocumentMessage:** Crea un messaggio con un documento in base64 (PDF o testo).
 - **DocumentBase64:** I dati del documento codificati in base64.
 - **MediaType:** Il tipo MIME (ad es. application/pdf).

Vision

Claude può comprendere immagini passate come blocchi di contenuto con codifica base64 all'interno dei messaggi.

- **CreateVisionMessage:** Invia un'immagine con un prompt testuale.
 - **ImageBase64:** I dati dell'immagine codificati in base64.
 - **MediaType:** Il tipo MIME (image/jpeg, image/png, image/gif, image/webp).
 - **Prompt:** Il prompt testuale da accompagnare all'immagine.

Tool Use

Claude può utilizzare strumenti (function calling) per interagire con sistemi esterni. Si definiscono gli strumenti con i rispettivi nomi, descrizioni e schemi di input, e Claude genererà blocchi di contenuto tool_use quando desidera chiamare uno strumento.

Models

Elenca e descrive i modelli Claude disponibili.

- **GetModels:** Elenca tutti i modelli disponibili.
- **GetModel:** Recupera informazioni su un modello specifico.
 - **ModelId:** L'ID del modello da recuperare.

Extended Thinking

Il pensiero esteso consente a Claude di ragionare passo dopo passo prima di rispondere, migliorando la qualità per attività complesse come matematica, codice e analisi.

- **ThinkingType:** Imposti su 'enabled' per attivare il thinking, 'disabled' per disattivarlo.
- **ThinkingBudgetTokens:** Budget di token per il ragionamento (minimo 1024, deve essere inferiore a MaxTokens).
- **CreateMessageWithAdaptiveThinking:** Crea un messaggio con pensiero adattivo, lasciando che Claude decida autonomamente la profondità di ragionamento.

Documenti

Claude può elaborare documenti PDF e file di testo inviati come blocchi di contenuto. Supporta i tipi di origine base64, text e URL.

- **CreateDocumentMessage:** Invia un documento con un prompt testuale.
 - **DocumentBase64:** I dati del documento codificati in base64.
 - **MediaType:** Il tipo MIME (application/pdf, text/plain).

Prompt Caching

Memorizzare nella cache il contesto utilizzato di frequente (prompt di sistema, blocchi di contenuto, definizioni di strumenti) tra le chiamate API per ridurre i costi fino al 90% sulle letture dalla cache.

- **SystemCacheControl:** impostare su True nella richiesta per memorizzare nella cache il prompt di sistema.
- **CacheControl:** Impostare su 'ephemeral' sui blocchi di contenuto o sulle definizioni degli strumenti.
- **BetaHeaders:** Header opzionali per funzionalità beta in AnthropicOptions.

Citazioni

Quando i documenti vengono inviati con le citazioni abilitate, Claude include riferimenti alle fonti nella sua risposta. I tipi di citazione includono char_location (testo), page_location (PDF), content_block_location (contenuto personalizzato) e web_search_result_location (ricerca web).

Web Search

Claude può cercare informazioni in tempo reale sul web utilizzando lo strumento integrato web_search. Altri strumenti integrati includono code_execution e computer use.

- **CreateMessageWithWebSearch:** Crea un messaggio con la ricerca web abilitata.
 - **Model:** Il modello da utilizzare.
 - **Message:** La query dell'utente.

Conteggio token

Conta il numero di token in un messaggio prima di inviarlo.

- **CountTokens:** Conta il numero di token di input di un messaggio.
 - **Model:** Il modello da utilizzare per il conteggio dei token.
 - **Message:** Il contenuto del messaggio di cui contare i token.

Batch di messaggi

L'API Message Batches consente di elaborare grandi volumi di messaggi in modo asincrono.

- **ListBatches:** elenca tutti i batch di messaggi.
- **GetBatch:** Recupera un batch specifico tramite ID.
- **CancelBatch:** Annulla un batch ancora in elaborazione.
- **GetBatchResults:** Recupera i risultati di un batch completato.

Output Strutturati

Forzare Claude a restituire JSON valido conforme a uno schema JSON fornito. Combinare con il parametro Effort per controllare la qualità dell'output rispetto al costo.

- **CreateMessageJSON:** Crea un messaggio con output in schema JSON.
 - **Schema:** Una stringa JSON Schema che definisce il formato di output.
- **OutputFormatSchema:** (Proprietà della richiesta) Schema JSON per l'output strutturato.
- **Effort:** (proprietà della Richiesta) Controlla qualità vs. costo: 'low', 'medium', 'high', 'max'.
- **Strict:** (proprietà Tool) Quando è True, gli input del tool sono garantiti corrispondere all'input_schema.

Files API

Caricamento, elenco, recupero, download ed eliminazione di file. I file caricati possono essere referenziati nei messaggi utilizzando blocchi di contenuto documento con tipo di sorgente file. Richiede l'intestazione beta `files-api-2025-04-14`.

- **UploadFile:** Carica un file locale. Restituisce i metadati del file.
- **ListFiles:** Elenca i file caricati con paginazione.
- **GetFile:** Recupera i metadati di un file specifico.
- **DownloadFile:** Scarica il contenuto del file.
- **DeleteFile:** Elimina definitivamente un file.

Parametri della Richiesta

Parametri di richiesta aggiuntivi disponibili nell'API Messages.

- **ServiceTier:** Controlla l'utilizzo del livello di priorità: 'auto' (predefinito) o 'standard_only'.
- **InferenceGeo:** Controlla la geografia dell'inferenza: 'global' (predefinito) o 'us'.
- **Container:** ID del container per il riutilizzo degli ambienti di esecuzione del codice tra le richieste.
- **IsError:** (proprietà del blocco di contenuto) Impostare su True nei blocchi tool_result per indicare un errore nell'esecuzione dello strumento.
- **CacheTTL:** Tempo di vita della cache estesa: '5m' (predefinito) o '1h'.
- **ThinkingType:** Impostare su 'adaptive' per consentire a Claude di decidere automaticamente la profondità del thinking.

MCP Connector

Collega Claude a server MCP (Model Context Protocol) esterni per accedere a strumenti di terze parti. Richiede l'header beta `mcp-client-2025-11-20`.

- **CreateMessageWithMCP:** Crea un messaggio con una connessione al server MCP.
 - **MCPServerUrl:** URL HTTPS del server MCP.
 - **MCPServerName:** Nome univoco per il server.
- **MCPServers:** (proprietà Request) Array di definizioni di server MCP con ServerType, Url, Name, AuthorizationToken.
- **MCPServerName:** (proprietà dello strumento) Fa riferimento a un server MCP quando ToolType è 'mcp_toolset'.

Anthropic | Messaggi

Inviare un elenco strutturato di messaggi di input con contenuto testuale e il modello genererà il prossimo messaggio nella conversazione.

Esempio Semplice

Inviare un messaggio Hello a Claude.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";
MessageBox.Show(Anthropic._CreateMessage("claude-sonnet-4-20250514", "Hello!"));
```

Esempio di prompt di sistema

Invia un messaggio con un prompt di sistema per controllare il comportamento di Claude.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";
MessageBox.Show(Anthropic._CreateMessageWithSystem("claude-sonnet-4-20250514",
    "You are a helpful assistant that responds in Spanish.",
    "What is the capital of France?"));
```

Esempio avanzato

Utilizzare le classi di richiesta/risposta tipizzate per il controllo completo dei parametri dei messaggi.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";

TsgcAnthropicClass_Request_Messages oRequest = new TsgcAnthropicClass_Request_Messages();
oRequest.Model = "claude-sonnet-4-20250514";
oRequest.MaxTokens = 1024;
oRequest.System = "You are a helpful assistant.";
oRequest.Temperature = 0.7;

TsgcAnthropicClass_Request_Message oMessage = new TsgcAnthropicClass_Request_Message();
oMessage.Role = "user";
oMessage.Content = "Hello!";
var oMessages = oRequest.Messages;
SetLength(oMessages, 1);
oMessages[0] = oMessage;
oRequest.Messages = oMessages;

TsgcAnthropicClass_Response_Messages oResponse = Anthropic.CreateMessage(oRequest);
if (oResponse.Content.Length > 0)
    MessageBox.Show(oResponse.Content[0].Text);
```

Esempio di streaming

Utilizzare Server-Sent Events (SSE) per trasmettere la risposta in tempo reale. Assegnare il gestore dell'evento OnHTTPAPISSE per ricevere gli eventi in streaming.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";
Anthropic.OnHTTPAPISSE += OnSSEEvent;
Anthropic._CreateMessageStream("claude-sonnet-4-20250514", "Tell me a story.");

void OnSSEEvent(object Sender, string aEvent, string aData, ref bool Cancel)
{
    // aEvent contains the event type (e.g. content_block_delta)
    // aData contains the JSON data for this event
    Memo1.Lines.Add(aData);
}
```

```
}
```

Anthropic | Vision

Claude è in grado di comprendere e analizzare le immagini. È possibile inviare immagini come dati codificati in base64 all'interno di blocchi di contenuto.

Formati di immagine supportati

- **image/jpeg** - Immagini JPEG
- **image/png** - Immagini PNG
- **image/gif** - Immagini GIF
- **image/webp** - immagini WebP

Esempio Semplice

Invia un'immagine con un prompt chiedendo a Claude di descriverla.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";

// Load image and encode to base64
byte[] fileBytes = System.IO.File.ReadAllBytes("photo.png");
string vBase64 = Convert.ToBase64String(fileBytes);

MessageBox.Show(Anthropic._CreateVisionMessage("claude-sonnet-4-20250514",
    "What is in this image?", vBase64, "image/png"));
```

Esempio avanzato

Utilizzi i blocchi di contenuto per un maggiore controllo sul messaggio con immagine.

```
TsgcAnthropicClass_Request_Messages oRequest = new TsgcAnthropicClass_Request_Messages();
oRequest.Model = "claude-sonnet-4-20250514";
oRequest.MaxTokens = 4096;

TsgcAnthropicClass_Request_Message oMessage = new TsgcAnthropicClass_Request_Message();
oMessage.Role = "user";

// Image content block
TsgcAnthropicClass_Request_Content_Block oImageBlock = new TsgcAnthropicClass_Request_Content_Block();
oImageBlock.ContentType = "image";
oImageBlock.MediaType = "image/jpeg";
oImageBlock.Data = vBase64;

// Text content block
TsgcAnthropicClass_Request_Content_Block oTextBlock = new TsgcAnthropicClass_Request_Content_Block();
oTextBlock.ContentType = "text";
oTextBlock.Text = "Describe this image in detail.";

SetLength(oBlocks, 2);
oBlocks[0] = oImageBlock;
oBlocks[1] = oTextBlock;
oMessage.ContentBlocks = oBlocks;

SetLength(oMessages, 1);
oMessages[0] = oMessage;
oRequest.Messages = oMessages;

var oResponse = Anthropic.CreateMessage(oRequest);
if (oResponse.Content.Length > 0)
    MessageBox.Show(oResponse.Content[0].Text);
```

Anthropic | Utilizzo degli strumenti

Claude supporta l'uso di strumenti (function calling), consentendo di definire strumenti che Claude può invocare durante una conversazione. Quando Claude decide di utilizzare uno strumento, restituisce un blocco di contenuto **tool_use**. Si esegue quindi lo strumento e si invia il risultato come blocco di contenuto **tool_result**.

Flusso di utilizzo degli strumenti

1. Definire gli strumenti con nome, descrizione e input_schema (JSON Schema).
2. Invia un messaggio con gli strumenti definiti.
3. Claude risponde con un blocco di contenuto **tool_use** (stop_reason = 'tool_use').
4. Esegue lo strumento con l'input fornito.
5. Inviare un nuovo messaggio con un blocco di contenuto **tool_result** contenente l'output.
6. Claude risponde con la risposta finale.

Esempio

Definire uno strumento meteo e gestire il ciclo di utilizzo degli strumenti.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";

// Step 1: Create request with tools
TsgcAnthropicClass_Request_Messages oRequest = new TsgcAnthropicClass_Request_Messages();
oRequest.Model = "claude-sonnet-4-20250514";
oRequest.MaxTokens = 4096;

// Define user message
TsgcAnthropicClass_Request_Message oMessage = new TsgcAnthropicClass_Request_Message();
oMessage.Role = "user";
oMessage.Content = "What is the weather in San Francisco?";
SetLength(oMessages, 1);
oMessages[0] = oMessage;
oRequest.Messages = oMessages;

// Define tool
TsgcAnthropicClass_Request_Tool oTool = new TsgcAnthropicClass_Request_Tool();
oTool.Name = "get_weather";
oTool.Description = "Get the current weather in a given location";
oTool.InputSchema =
    "{\"type\":\"object\",\"properties\":{\"location\":{\"type\":\"string\"},\" +
    \"description\":{\"The city and state\"}},\"required\":[\"location\"]}";
SetLength(oTools, 1);
oTools[0] = oTool;
oRequest.Tools = oTools;

// Step 2: Send request
var oResponse = Anthropic.CreateMessage(oRequest);
// Step 3: Check if Claude wants to use a tool
if (oResponse.StopReason == "tool_use") {
    // Find the tool_use content block
    for (int i = 0; i < oResponse.Content.Length; i++) {
        if (oResponse.Content[i].ContentType == "tool_use") {
            vToolUseId = oResponse.Content[i].Id;
            vToolName = oResponse.Content[i].Name;
            vToolInput = oResponse.Content[i].Input;
            // Step 4: Execute your tool (get_weather) and get result
            vToolResult = "72 degrees and sunny";
            break;
        }
    }
}

// Step 5: Send tool result back
// Build new message array with assistant response + tool result
// ... (continue the conversation with tool_result content block)
}
```

Anthropic | Models

Elenca e recupera informazioni sui modelli Claude disponibili.

Elenca modelli

Elenca tutti i modelli Claude disponibili.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";
MessageBox.Show(Anthropic._GetModels());
```

Ottieni modello

Recupera informazioni su un modello specifico.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";
MessageBox.Show(Anthropic._GetModel("claude-sonnet-4-20250514"));
```

Risposta tipizzata

Utilizzare la classe di risposta tipizzata per l'accesso strutturato ai dati del modello.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";

TsgcAnthropicClass_Response_Models oModels = Anthropic.GetModels();
for (int i = 0; i < oModels.Data.Length; i++)
    MessageBox.Show(oModels.Data[i].Id + " - " + oModels.Data[i].DisplayName);
```

Anthropic | Message Batches

L'API Message Batches consente di elaborare grandi volumi di messaggi in modo asincrono. È ideale per attività che non richiedono risposte immediate, come la generazione di contenuti in blocco, l'analisi dei dati o i flussi di elaborazione batch.

List Batches

Elenca tutti i batch di messaggi per la propria organizzazione.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";
MessageBox.Show(Anthropic._ListBatches());
```

Recupera Batch

Recupera informazioni su un batch specifico.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";
MessageBox.Show(Anthropic._GetBatch("batch_id_here"));
```

Annulla batch

Annulla un batch ancora in elaborazione.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";
MessageBox.Show(Anthropic._CancelBatch("batch_id_here"));
```

Recupera i risultati del batch

Recupera i risultati di un batch completato.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";
MessageBox.Show(Anthropic._GetBatchResults("batch_id_here"));
```

Risposta tipizzata

Utilizzare la classe di risposta tipizzata per l'accesso strutturato ai dati batch.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";

var oBatches = Anthropic.ListBatches();
for (int i = 0; i < oBatches.Batches.Length; i++)
    MessageBox.Show(oBatches.Batches[i].Id + " - " + oBatches.Batches[i].ProcessingStatus);
```

Anthropic | Extended Thinking

Il pensiero esteso offre a Claude la capacità di ragionare passo dopo passo su problemi complessi prima di fornire una risposta. Quando abilitato, Claude crea un ragionamento interno (blocchi di pensiero) che migliora la qualità delle risposte per matematica, programmazione, analisi e altri compiti complessi.

Esempio Semplice

Invia un messaggio con il ragionamento esteso abilitato utilizzando il metodo di convenienza. La temperatura viene impostata automaticamente a 1,0 (richiesto dall'API quando il ragionamento è abilitato).

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";
MessageBox.Show(Anthropic._CreateMessageWithThinking("claude-sonnet-4-20250514",
    "How many r's are in the word strawberry?", 10000));
```

Esempio avanzato

Utilizzare le classi di richiesta/risposta tipizzate per il controllo completo. Impostare ThinkingType su 'enabled' e ThinkingBudgetTokens sul budget di token desiderato (minimo 1024). La risposta conterrà blocchi di contenuto di tipo thinking e testo.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";

TsgcAnthropicClass_Request_Messages oRequest = new TsgcAnthropicClass_Request_Messages();
oRequest.Model = "claude-sonnet-4-20250514";
oRequest.MaxTokens = 16384;
oRequest.ThinkingType = "enabled";
oRequest.ThinkingBudgetTokens = 10000;

TsgcAnthropicClass_Request_Message oMessage = new TsgcAnthropicClass_Request_Message();
oMessage.Role = "user";
oMessage.Content = "Explain the proof that there are infinitely many primes.";
var oMessages = oRequest.Messages;
SetLength(oMessages, 1);
oMessages[0] = oMessage;
oRequest.Messages = oMessages;

TsgcAnthropicClass_Response_Messages oResponse = Anthropic.CreateMessage(oRequest);
for (int i = 0; i < oResponse.Content.Length; i++)
{
    if (oResponse.Content[i].ContentType == "thinking")
        MessageBox.Show("Thinking: " + oResponse.Content[i].Thinking);
    else if (oResponse.Content[i].ContentType == "text")
        MessageBox.Show("Response: " + oResponse.Content[i].Text);
}
```

Multi-turno con Thinking

Quando si utilizza il pensiero esteso nelle conversazioni multi-turno, i blocchi thinking e redacted_thinking delle risposte precedenti devono essere reinseriti nella conversazione. Utilizzare l'array ContentBlocks per includere questi blocchi.

```
// Pass back thinking blocks from a previous response
TsgcAnthropicClass_Request_Content_Block oBlock = new TsgcAnthropicClass_Request_Content_Block();
oBlock.ContentType = "thinking";
oBlock.Text = oPrevThinkingBlock.Thinking; // thinking text
oBlock.Signature = oPrevThinkingBlock.Signature; // signature string

// Pass back redacted thinking blocks
TsgcAnthropicClass_Request_Content_Block oBlock2 = new TsgcAnthropicClass_Request_Content_Block();
oBlock2.ContentType = "redacted_thinking";
oBlock2.Data = oPrevRedactedBlock.Data;
```

Proprietà

- **ThinkingType:** Impostare su 'enabled' per attivare il pensiero esteso, o 'disabled' per disattivarlo.
- **ThinkingBudgetTokens:** Il numero massimo di token per il thinking (minimo 1024). Deve essere inferiore a MaxTokens.

Note

- Temperature deve essere 1.0 quando thinking è abilitato (il metodo di convenienza lo imposta automaticamente).
- TopK non può essere utilizzato quando il pensiero è abilitato.
- BudgetTokens deve essere almeno 1024 e inferiore a MaxTokens.
- I blocchi di contenuto della risposta possono includere i tipi: thinking, redacted_thinking e text.

Anthropic | Documenti

Claude può elaborare documenti PDF e file di testo inviati come blocchi di contenuto all'interno dei messaggi. I documenti vengono inviati come dati codificati in base64 o tramite URL, e Claude può analizzarli, riassumerli e rispondere a domande sul loro contenuto.

Esempio Semplice

Inviare un documento PDF con una domanda utilizzando il metodo di convenienza.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";

// Load PDF file and encode to base64
string vBase64 = sgcBase64Encode(LoadFileToBytes("document.pdf"));
MessageBox.Show(Anthropic._CreateDocumentMessage("claude-sonnet-4-20250514",
    "Summarize this document.", vBase64, "application/pdf"));
```

Esempio Avanzato con Citations

Utilizzi le classi tipizzate per inviare un documento con le citazioni abilitate. Quando le citazioni sono abilitate, la risposta di Claude includerà riferimenti a parti specifiche del documento sorgente.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";

TsgcAnthropicClass_Request_Messages oRequest = new TsgcAnthropicClass_Request_Messages();
oRequest.Model = "claude-sonnet-4-20250514";
oRequest.MaxTokens = 4096;

// Create document content block
TsgcAnthropicClass_Request_Content_Block oDocBlock = new TsgcAnthropicClass_Request_Content_Block();
oDocBlock.ContentType = "document";
oDocBlock.SourceType = "base64";
oDocBlock.MediaType = "application/pdf";
oDocBlock.Data = sgcBase64Encode(LoadFileToBytes("report.pdf"));
oDocBlock.Title = "Annual Report";
oDocBlock.CitationsEnabled = true;

// Create text prompt block
TsgcAnthropicClass_Request_Content_Block oTextBlock = new TsgcAnthropicClass_Request_Content_Block();
oTextBlock.ContentType = "text";
oTextBlock.Text = "What are the key findings?";

// Build message with content blocks
TsgcAnthropicClass_Request_Message oMessage = new TsgcAnthropicClass_Request_Message();
oMessage.Role = "user";
var oBlocks = oMessage.ContentBlocks;
SetLength(oBlocks, 2);
oBlocks[0] = oDocBlock;
oBlocks[1] = oTextBlock;
oMessage.ContentBlocks = oBlocks;

var oMessages = oRequest.Messages;
SetLength(oMessages, 1);
oMessages[0] = oMessage;
oRequest.Messages = oMessages;

TsgcAnthropicClass_Response_Messages oResponse = Anthropic.CreateMessage(oRequest);
if (oResponse.Content.Length > 0)
    MessageBox.Show(oResponse.Content[0].Text);
```

Proprietà

- **ContentType:** Impostare su 'document' per i blocchi di contenuto documento.
- **SourceType:** Il tipo di origine: 'base64' (predefinito), 'text' o 'url'.
- **MediaType:** Il tipo MIME (ad es. application/pdf, text/plain).
- **Data:** I dati del documento codificati in base64, oppure l'URL quando SourceType è 'url'.

- **Title:** Titolo del documento facoltativo, a scopo di riferimento.
- **Contesto:** Contesto o metadati opzionali relativi al documento.
- **CitationsEnabled:** Impostare su True per abilitare le citazioni delle fonti nella risposta.

Formati supportati

- **PDF:** application/pdf (max 32 MB, max 100 pagine per richiesta)
- **Plain Text:** text/plain

Anthropic | Prompt Caching

Il caching dei prompt consente di memorizzare nella cache il contesto utilizzato frequentemente tra le chiamate API, riducendo i costi fino al 90% sulle letture dalla cache e migliorando la latenza. Il contenuto memorizzato nella cache è contrassegnato con un parametro `cache_control` e viene riutilizzato tra le richieste all'interno della finestra TTL della cache.

Memorizzazione nella cache del prompt di sistema

Memorizzare nella cache un lungo prompt di sistema per evitare di rielaborarlo ad ogni richiesta. Impostare `SystemCacheControl` su `True` per includere automaticamente il prompt di sistema in `cache_control`.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";

TsgcAnthropicClass_Request_Messages oRequest = new TsgcAnthropicClass_Request_Messages();
oRequest.Model = "claude-sonnet-4-20250514";
oRequest.MaxTokens = 4096;
oRequest.System = "You are an expert legal assistant... (long system prompt)";
oRequest.SystemCacheControl = true; // Enable caching for system prompt

TsgcAnthropicClass_Request_Message oMessage = new TsgcAnthropicClass_Request_Message();
oMessage.Role = "user";
oMessage.Content = "Analyze this contract clause.";
var oMessages = oRequest.Messages;
SetLength(oMessages, 1);
oMessages[0] = oMessage;
oRequest.Messages = oMessages;

var oResponse = Anthropic.CreateMessage(oRequest);
// Check cache usage in response
MessageBox.Show("Cache created: " +
    oResponse.Usage.CacheCreationInputTokens.ToString());
MessageBox.Show("Cache read: " +
    oResponse.Usage.CacheReadInputTokens.ToString());
MessageBox.Show(oResponse.Content[0].Text);
```

Caching dei blocchi di contenuto

Memorizzare nella cache blocchi di contenuto specifici (ad es. documenti di grandi dimensioni o contesto) impostando `CacheControl` su `'ephemeral'` su singoli blocchi di contenuto.

```
TsgcAnthropicClass_Request_Content_Block oBlock = new TsgcAnthropicClass_Request_Content_Block();
oBlock.ContentType = "text";
oBlock.Text = "(large reference text to cache)";
oBlock.CacheControl = "ephemeral"; // Mark for caching
```

Memorizzazione nella cache delle definizioni degli strumenti

Memorizza nella cache le definizioni degli strumenti per evitare di rielaborarle a ogni richiesta. Questo è utile quando ha molte definizioni di strumenti che rimangono costanti tra le richieste.

```
TsgcAnthropicClass_Request_Tool oTool = new TsgcAnthropicClass_Request_Tool();
oTool.Name = "search_database";
oTool.Description = "Search the database for records.";
oTool.InputSchema = "{\n\"type\": \"object\", \"properties\": {\n\"query\": {\n\"type\": \"string\"}\n}}";
oTool.CacheControl = "ephemeral"; // Cache this tool definition
```

Proprietà

- **SystemCacheControl:** impostare su `True` nella richiesta per memorizzare nella cache il prompt di sistema con `cache_control` effimero.

- **CacheControl:** Impostare su 'ephemeral' nei blocchi di contenuto o nelle definizioni degli strumenti per contrassegnarli per la memorizzazione nella cache.
- **CacheCreationInputTokens:** (Risposta) Numero di token utilizzati per creare le voci della cache.
- **CacheReadInputTokens:** (Risposta) Numero di token letti dalla cache (risparmio sui costi).

Prezzi

- **Scritture in cache:** 1,25 volte il prezzo base del token di input (TTL di 5 minuti).
- **Lecture dalla cache:** 0,1x il prezzo base dei token di input (risparmio del 90%).
- **TTL della cache:** 5 minuti per impostazione predefinita. Le richieste successive aggiornano il TTL.

Anthropic | Citations

Quando i documenti vengono inviati con le citazioni abilitate, la risposta di Claude include riferimenti a parti specifiche dei documenti sorgente. Questo consente di verificare le affermazioni e di risalire alle informazioni alla loro origine.

Abilitazione delle Citazioni

Impostare `CitationsEnabled` su `True` sui blocchi di contenuto del documento. Le citazioni devono essere abilitate su TUTTI o NESSUNO dei documenti in una richiesta.

```
TsgcAnthropicClass_Request_Content_Block oDocBlock = new TsgcAnthropicClass_Request_Content_Block();
oDocBlock.ContentType = "document";
oDocBlock.SourceType = "base64";
oDocBlock.MediaType = "application/pdf";
oDocBlock.Data = sgcBase64Encode(LoadFileToBytes("report.pdf"));
oDocBlock.CitationsEnabled = true; // Enable citations
```

Lettura delle citazioni dalla risposta

I blocchi di contenuto testuale nella risposta possono contenere un array `Citations` con riferimenti alle fonti.

```
var oResponse = Anthropic.CreateMessage(oRequest);
for (int i = 0; i < oResponse.Content.Length; i++) {
    if (oResponse.Content[i].ContentType == "text") {
        MessageBox.Show(oResponse.Content[i].Text);

        // Process citations
        for (int j = 0; j < oResponse.Content[i].Citations.Length; j++) {
            var oCitation = oResponse.Content[i].Citations[j];
            MessageBox.Show(string.Format(" Citation [{0}]: \"{1}\"",
                oCitation.CitationType, oCitation.CitedText));

            if (oCitation.CitationType == "page_location")
                MessageBox.Show(string.Format(" Pages {0}-{1} of \"{2}\"",
                    oCitation.StartPageNumber, oCitation.EndPageNumber,
                    oCitation.DocumentTitle));
        }
    }
}
```

Tipi di citazione

- **char_location:** Riferimento a livello di carattere nei documenti di testo normale. Campi: `DocumentIndex`, `DocumentTitle`, `StartCharIndex`, `EndCharIndex`.
- **page_location:** Riferimento a livello di pagina nei documenti PDF. Campi: `DocumentIndex`, `DocumentTitle`, `StartPageNumber`, `EndPageNumber`.
- **content_block_location:** Riferimento a livello di blocco nei documenti di contenuto personalizzato. Campi: `DocumentIndex`, `DocumentTitle`, `StartBlockIndex`, `EndBlockIndex`.
- **web_search_result_location:** Riferimento ai risultati della ricerca web. Campi: `Url`, `Title`, `EncryptedIndex`.

Anthropic | Web Search

Lo strumento Web Search consente a Claude di cercare informazioni in tempo reale sul web durante una conversazione. Si tratta di uno strumento integrato lato server che Anthropic ospita ed esegue automaticamente.

Esempio Semplice

Utilizzare il metodo di convenienza per creare un messaggio con la ricerca web abilitata.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";
MessageBox.Show(Anthropic._CreateMessageWithWebSearch("claude-sonnet-4-20250514",
    "What are the latest news about Delphi programming?"));
```

Esempio avanzato

Utilizzare le classi tipizzate per il controllo completo dei parametri dello strumento di ricerca web, come MaxUses.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";

TsgcAnthropicClass_Request_Messages oRequest = new TsgcAnthropicClass_Request_Messages();
oRequest.Model = "claude-sonnet-4-20250514";
oRequest.MaxTokens = 4096;

// Add web search tool
TsgcAnthropicClass_Request_Tool oTool = new TsgcAnthropicClass_Request_Tool();
oTool.ToolType = "web_search_20250305";
oTool.Name = "web_search";
oTool.MaxUses = 5; // Max 5 searches per request
var oTools = oRequest.Tools;
SetLength(oTools, 1);
oTools[0] = oTool;
oRequest.Tools = oTools;

TsgcAnthropicClass_Request_Message oMessage = new TsgcAnthropicClass_Request_Message();
oMessage.Role = "user";
oMessage.Content = "Find the current price of Bitcoin.";
var oMessages = oRequest.Messages;
SetLength(oMessages, 1);
oMessages[0] = oMessage;
oRequest.Messages = oMessages;

TsgcAnthropicClass_Response_Messages oResponse = Anthropic.CreateMessage(oRequest);
for (int i = 0; i < oResponse.Content.Length; i++)
{
    if (oResponse.Content[i].ContentType == "text")
        MessageBox.Show(oResponse.Content[i].Text);
}
```

Tipi di strumenti integrati

- **web_search_20250305:** Ricerca web lato server. Proprietà: Name ('web_search'), MaxUses.
- **code_execution_20250825:** Esecuzione di codice in sandbox. Proprietà: Name ('code_execution'), MaxUses.
- **computer_20250124:** Uso del computer (mouse, tastiera, screenshot). Proprietà: Name ('computer'), DisplayWidthPx, DisplayHeightPx.

Tipi di Contenuto della Risposta

Quando vengono utilizzati strumenti integrati, la risposta può contenere tipi di blocchi di contenuto aggiuntivi:

- **server_tool_use:** Indica che Claude ha invocato uno strumento integrato. Campi: Id, Name, Input.
- **web_search_tool_result:** Contiene i risultati della ricerca web. Campi: ToolUsed, Data (contenuto JSON grezzo).

Anthropic | Structured Outputs

Gli Output Strutturati forzano Claude a restituire risposte conformi a uno schema JSON. Ciò garantisce un output JSON valido e analizzabile conforme alla definizione dello schema.

Esempio Semplice

Utilizzare il metodo conveniente per creare un messaggio con output in schema JSON.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";

string vSchema = "{\"type\":\"object\",\"properties\":{\"name\":{\"type\":\"string\"},\" +
    \"age\":{\"type\":\"integer\"}},\"required\":[\"name\",\"age\"],\" +
    \"additionalProperties\":false}";

MessageBox.Show(Anthropic._CreateMessageJSON("claude-sonnet-4-20250514",
    "Extract the name and age: John is 30 years old.", vSchema));
```

Esempio avanzato

Utilizzare le classi tipizzate per combinare l'output strutturato con il parametro effort.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";

TsgcAnthropicClass_Request_Messages oRequest = new TsgcAnthropicClass_Request_Messages();
oRequest.Model = "claude-sonnet-4-20250514";
oRequest.MaxTokens = 4096;

// JSON schema for structured output
oRequest.OutputFormatSchema =
    "{\"type\":\"object\",\"properties\":{\"sentiment\":{\"type\":\"string\", \" +
    \"enum\":[\"positive\",\"negative\",\"neutral\"]},\"confidence\": \" +
    \"type\":\"number\"}},\"required\":[\"sentiment\",\"confidence\"],\" +
    \"additionalProperties\":false}";

// Set effort level (low, medium, high, max)
oRequest.Effort = "medium";

TsgcAnthropicClass_Request_Message oMessage = new TsgcAnthropicClass_Request_Message();
oMessage.Role = "user";
oMessage.Content = "Analyze the sentiment: I love this product!";
var oMessages = oRequest.Messages;
SetLength(oMessages, 1);
oMessages[0] = oMessage;
oRequest.Messages = oMessages;

TsgcAnthropicClass_Response_Messages oResponse = Anthropic.CreateMessage(oRequest);
for (int i = 0; i < oResponse.Content.Length; i++)
{
    if (oResponse.Content[i].ContentType == "text")
        MessageBox.Show(oResponse.Content[i].Text);
}
```

Strict Tool Use

Abilitare la modalità strict sulle definizioni degli strumenti per garantire che gli input degli strumenti siano conformi esattamente all'input_schema.

```
TsgcAnthropicClass_Request_Tool oTool = new TsgcAnthropicClass_Request_Tool();
oTool.Name = "get_weather";
oTool.Description = "Get the current weather for a location";
oTool.Strict = true; // Guarantee schema conformance
oTool.InputSchema = "{\"type\":\"object\",\"properties\":{\"location\": \" +
    \"type\":\"string\"}},\"required\":[\"location\"],\"additionalProperties\":false}";
```

Proprietà

- **OutputFormatSchema:** Una stringa JSON Schema. Il testo di risposta di Claude sarà JSON valido conforme a questo schema.
- **Effort:** Controlla qualità dell'output rispetto al costo. Valori: 'low', 'medium', 'high' (predefinito), 'max' (solo Opus 4.6).
- **Strict:** (sugli strumenti) Quando True, gli input degli strumenti sono garantiti conformi all'input_schema.

Anthropic | Files API

L'API Files consente di caricare, elencare, recuperare, scaricare ed eliminare file. I file caricati possono essere referenziati nei messaggi utilizzando blocchi di contenuto di tipo documento con sorgente di tipo file.

Nota: La Files API richiede l'intestazione beta `files-api-2025-04-14`. Impostarla nella proprietà **AnthropicOptions.BetaHeaders**.

Carica un file

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";
Anthropic.AnthropicOptions.BetaHeaders = "files-api-2025-04-14";

TsgcAnthropicClass_Response_File oFile = Anthropic.UploadFile("C:\\documents\\report.pdf");
MessageBox.Show("File ID: " + oFile.Id);
MessageBox.Show("Filename: " + oFile.Filename);
MessageBox.Show("Size: " + oFile.SizeBytes.ToString() + " bytes");
MessageBox.Show("MIME Type: " + oFile.MimeType);
```

Elenco file

```
TsgcAnthropicClass_Response_FileList oList = Anthropic.ListFiles(50);
for (int i = 0; i < oList.Data.Length; i++)
    MessageBox.Show(oList.Data[i].Id + " - " + oList.Data[i].Filename +
        " (" + oList.Data[i].SizeBytes.ToString() + " bytes)");
MessageBox.Show("Has more: " + oList.HasMore.ToString());
```

Usa file nei messaggi

Fare riferimento ai file caricati utilizzando il blocco di contenuto documento con `SourceType` impostato su 'file'.

```
TsgcAnthropicClass_Request_Content_Block oDocBlock = new TsgcAnthropicClass_Request_Content_Block();
oDocBlock.ContentType = "document";
oDocBlock.SourceType = "file";
oDocBlock.FileId = "file_abc123"; // ID from UploadFile

TsgcAnthropicClass_Request_Content_Block oTextBlock = new TsgcAnthropicClass_Request_Content_Block();
oTextBlock.ContentType = "text";
oTextBlock.Text = "Summarize this document.";
```

Elimina un file

```
TsgcAnthropicClass_Response_File oDeleted = Anthropic.DeleteFile("file_abc123");
MessageBox.Show("Deleted: " + oDeleted.Id);
```

Metodi API

- **UploadFile:** Carica un file da un percorso locale. Restituisce i metadati del file (Id, Filename, MimeType, SizeBytes).
- **ListFiles:** elenca tutti i file caricati con supporto alla paginazione (Limit, AfterId, BeforeId).
- **GetFile:** Recupera i metadati di un file specifico tramite ID.
- **DownloadFile:** Scarica il contenuto di un file (solo i file creati dallo strumento di esecuzione del codice sono scaricabili).
- **DeleteFile:** Elimina permanentemente un file tramite ID.

Anthropic | MCP Connector

Il connettore MCP (Model Context Protocol) consente a Claude di accedere agli strumenti da server MCP esterni. Ciò abilita l'integrazione con servizi di terze parti e provider di strumenti personalizzati.

Nota: Il connettore MCP richiede l'intestazione beta `mcp-client-2025-11-20`. Impostarla nella proprietà `AnthropicOptions.BetaHeaders`.

Esempio Semplice

Utilizzare il metodo di convenienza per creare un messaggio con un server MCP.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";
Anthropic.AnthropicOptions.BetaHeaders = "mcp-client-2025-11-20";

MessageBox.Show(Anthropic._CreateMessageWithMCP("claude-sonnet-4-20250514",
    "What tools are available?",
    "https://my-mcp-server.example.com/sse",
    "my-mcp-server"));
```

Esempio avanzato

Utilizzare le classi tipizzate per il controllo completo della configurazione del server MCP, inclusa l'autenticazione.

```
TsgcHTTP_API_Anthropic Anthropic = new TsgcHTTP_API_Anthropic();
Anthropic.AnthropicOptions.ApiKey = "API_KEY";
Anthropic.AnthropicOptions.BetaHeaders = "mcp-client-2025-11-20";

TsgcAnthropicClass_Request_Messages oRequest = new TsgcAnthropicClass_Request_Messages();
oRequest.Model = "claude-sonnet-4-20250514";
oRequest.MaxTokens = 4096;

// Configure MCP server
TsgcAnthropicClass_Request_MCPServer oServer = new TsgcAnthropicClass_Request_MCPServer();
oServer.ServerType = "url";
oServer.Url = "https://my-mcp-server.example.com/sse";
oServer.Name = "my-server";
oServer.AuthorizationToken = "OAUTH_TOKEN"; // Optional auth
var oServers = oRequest.MCPServers;
SetLength(oServers, 1);
oServers[0] = oServer;
oRequest.MCPServers = oServers;

// Add MCP toolset
TsgcAnthropicClass_Request_Tool oTool = new TsgcAnthropicClass_Request_Tool();
oTool.ToolType = "mcp_toolset";
oTool.MCPServerName = "my-server";
var oTools = oRequest.Tools;
SetLength(oTools, 1);
oTools[0] = oTool;
oRequest.Tools = oTools;

// Add user message
TsgcAnthropicClass_Request_Message oMessage = new TsgcAnthropicClass_Request_Message();
oMessage.Role = "user";
oMessage.Content = "Search for recent news about AI.";
var oMessages = oRequest.Messages;
SetLength(oMessages, 1);
oMessages[0] = oMessage;
oRequest.Messages = oMessages;

TsgcAnthropicClass_Response_Messages oResponse = Anthropic.CreateMessage(oRequest);
for (int i = 0; i < oResponse.Content.Length; i++)
{
    if (oResponse.Content[i].ContentType == "text")
        MessageBox.Show(oResponse.Content[i].Text);
    else if (oResponse.Content[i].ContentType == "mcp_tool_use")
        MessageBox.Show("MCP tool call: " + oResponse.Content[i].Name +
            " on " + oResponse.Content[i].ServerName);
}
```

Proprietà del server MCP

- **ServerType:** Attualmente è supportato solo 'url'.
- **Url:** L'URL HTTPS del server MCP.
- **Name:** Identificatore univoco per questo server. Deve corrispondere a MCPServerName nel toolset.
- **AuthorizationToken:** Token Bearer OAuth facoltativo per i server autenticati.

Tipi di Contenuto della Risposta

- **mcp_tool_use:** Claude ha invocato uno strumento MCP. Campi: Id, Name, ServerName, Input.
- **mcp_tool_result:** Risultato dell'esecuzione di uno strumento MCP. Campi: ToolUseId, IsError, Data (contenuto).

Gemini

Google Gemini è una famiglia di modelli AI multimodali sviluppati da Google DeepMind. I modelli Gemini supportano generazione di testo, visione, output strutturati, embedding e uso di strumenti, offrendo potenti funzionalità per la creazione di applicazioni basate sull'AI.

La libreria sgcWebSockets fornisce un componente Delphi **TsgcHTTP_API_Gemini** per interagire con l'API Gemini.

API Gemini

L'API Gemini fornisce accesso ai modelli Google Gemini per la creazione di applicazioni basate sull'intelligenza artificiale. L'API supporta la generazione di contenuti, la visione (comprensione delle immagini), output JSON strutturati, streaming, conteggio dei token, embedding, uso di strumenti (function calling) ed elenco dei modelli.

Funzionalità

- **Messaggi**
 - [Esempi di messaggi Gemini](#)
- **Vision**
 - [Esempi Gemini Vision](#)
- **Modelli**
 - [Esempi di Modelli Gemini](#)
- **Structured Outputs**
 - [Esempi di Structured Outputs Gemini](#)
- **Conteggio Token**
 - [Esempi di Conteggio Token Gemini](#)
- **Embeddings**
 - [Esempi di Gemini Embeddings](#)
- **Tool Use**
 - [Esempi di Gemini Tool Use](#)

Configurazione

L'API Gemini utilizza chiavi API per l'autenticazione. Visitare la pagina [Chiavi API](#) in Google AI Studio per recuperare la chiave API da utilizzare nelle richieste.

Ricordare che la propria chiave API è segreta! Non condividerla con altri né esporla in alcun codice lato client. Questa **API Key** deve essere configurata nella proprietà **GeminiOptions.ApiKey** del componente.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "YOUR_API_KEY";
```

Messaggi

Invia contenuto a un modello Gemini e riceve le risposte generate. Il modello genera il messaggio successivo in base all'input fornito.

- **_CreateContent:** Crea contenuto con il modello e il prompt utente specificati.
 - **Model:** Il modello da utilizzare (es. gemini-2.0-flash).
 - **Message:** Il contenuto del messaggio utente.
 - **MaxOutputTokens:** Numero massimo di token da generare.
- **_CreateContentWithSystem:** Crea contenuto con un'istruzione di sistema.
 - **Model:** Il modello da utilizzare.
 - **System:** Istruzione di sistema che definisce il comportamento del modello.
 - **Message:** Il contenuto del messaggio dell'utente.
 - **MaxOutputTokens:** Numero massimo di token da generare.
- **_CreateContentStream:** Crea contenuti con lo streaming (SSE) abilitato. Gli eventi vengono forniti tramite il gestore eventi OnHTTPAPISSE.

Vision

I modelli Gemini possono elaborare immagini trasmesse come contenuto codificato in base64 insieme a prompt testuali.

- **_CreateVisionContent:** Invia un'immagine con un prompt testuale.
 - **Model:** Il modello da utilizzare.
 - **Prompt:** Il prompt testuale da accompagnare all'immagine.
 - **Base64:** I dati dell'immagine codificati in base64.
 - **MediaType:** Il tipo MIME (image/jpeg, image/png, image/gif, image/webp).
 - **MaxOutputTokens:** Numero massimo di token da generare.

Output Strutturati

Genera output JSON strutturato da un modello Gemini fornendo uno schema JSON che definisce il formato di risposta atteso.

- **_CreateContentJSON:** Crea contenuto con output JSON strutturato.
 - **Model:** Il modello da utilizzare.
 - **Message:** Il contenuto del messaggio dell'utente.
 - **Schema:** Uno schema JSON che definisce la struttura di output prevista.
 - **MaxOutputTokens:** Numero massimo di token da generare.

Models

Elencare e recuperare dettagli sui modelli Gemini disponibili.

- **_GetModels:** Elenca tutti i modelli disponibili.
- **_GetModel:** recupera i dettagli di un modello specifico.
 - **ModelId:** L'identificatore del modello da recuperare.

Conteggio token

Conta il numero di token in un messaggio prima di inviarlo a un modello.

- **_CountTokens:** Conta i token per un messaggio.
 - **Model:** Il modello da utilizzare per la tokenizzazione.
 - **Message:** Il contenuto testuale di cui contare i token.

Embeddings

Genera embedding vettoriali per contenuti testuali utilizzando i modelli Gemini.

- **_EmbedContent:** genera embedding per il testo.
 - **Model:** il modello da utilizzare per la generazione degli embedding.
 - **Text:** il contenuto di testo per cui generare gli embedding.

DeepSeek

DeepSeek è una società cinese di intelligenza artificiale focalizzata sulla creazione di potenti modelli linguistici open-source. I suoi modelli eccellono nella programmazione, nel ragionamento e nei compiti di uso generale, offrendo ottime prestazioni a prezzi competitivi.

La libreria `sgcWebSockets` fornisce un componente Delphi `TsgcHTTP_API_DeepSeek` per interagire con l'API DeepSeek.

DeepSeek API

L'API DeepSeek fornisce accesso ai modelli DeepSeek per la creazione di applicazioni basate sull'AI. L'API supporta la generazione di testo, la visione (comprensione delle immagini), lo streaming e l'elenco dei modelli. L'API segue un formato compatibile con OpenAI, rendendola facile da integrare.

Funzionalità

- **Messaggi**
 - [Esempi di messaggi DeepSeek](#)
- **Vision**
 - [Esempi DeepSeek Vision](#)
- **Modelli**
 - [Esempi di modelli DeepSeek](#)

Configurazione

L'API DeepSeek utilizza chiavi API per l'autenticazione. Visitare la pagina [Chiavi API](#) nella piattaforma DeepSeek per recuperare la chiave API da utilizzare nelle richieste.

Ricordare che la propria chiave API è segreta! Non condividerla con altri né esporla in alcun codice lato client. Questa **API Key** deve essere configurata nella proprietà **DeepSeekOptions.ApiKey** del componente.

```
TsgcHTTP_API_DeepSeek DeepSeek = new TsgcHTTP_API_DeepSeek();
DeepSeek.DeepSeekOptions.ApiKey = "YOUR_API_KEY";
```

Messaggi

Inviare un elenco strutturato di messaggi di input con contenuto testuale e il modello genererà il prossimo messaggio nella conversazione.

- **_CreateMessage:** Crea un messaggio con il modello specificato e il prompt utente.
 - **Model:** Il modello da utilizzare (es. deepseek-chat).
 - **Message:** Il contenuto del messaggio utente.
- **_CreateMessageWithSystem:** Crea un messaggio con un prompt di sistema.
 - **System:** Prompt di sistema che definisce il comportamento dell'assistente.
- **_CreateMessageStream:** Crea un messaggio con streaming (SSE) abilitato. Gli eventi vengono consegnati tramite il gestore dell'evento `OnHTTPAPISSE`.

Vision

DeepSeek è in grado di comprendere le immagini passate come contenuto codificato in base64 all'interno dei messaggi.

- **_CreateVisionMessage:** Invia un'immagine con un prompt testuale.
 - **ImageBase64:** I dati dell'immagine codificati in base64.
 - **MediaType:** Il tipo MIME (image/jpeg, image/png, image/gif, image/webp).
 - **Prompt:** Il prompt testuale da accompagnare all'immagine.

Models

Elenca i modelli DeepSeek disponibili.

- **_GetModels:** Elenca tutti i modelli disponibili.

DeepSeek | Messages

Inviare un elenco strutturato di messaggi di input con contenuto testuale e il modello genererà il prossimo messaggio nella conversazione.

Esempio Semplice

Invia un messaggio Hello a DeepSeek.

```
TsgcHTTP_API_DeepSeek DeepSeek = new TsgcHTTP_API_DeepSeek();
DeepSeek.DeepSeekOptions.ApiKey = "API_KEY";
MessageBox.Show(DeepSeek._CreateMessage("deepseek-chat", "Hello!"));
```

Esempio di prompt di sistema

Inviare un messaggio con un prompt di sistema per controllare il comportamento di DeepSeek.

```
TsgcHTTP_API_DeepSeek DeepSeek = new TsgcHTTP_API_DeepSeek();
DeepSeek.DeepSeekOptions.ApiKey = "API_KEY";
MessageBox.Show(DeepSeek._CreateMessageWithSystem("deepseek-chat",
    "You are a helpful assistant that responds in Spanish.",
    "What is the capital of France?"));
```

Esempio di streaming

Utilizzare Server-Sent Events (SSE) per trasmettere la risposta in tempo reale. Assegnare il gestore dell'evento OnHTTPAPISSE per ricevere gli eventi in streaming.

```
TsgcHTTP_API_DeepSeek DeepSeek = new TsgcHTTP_API_DeepSeek();
DeepSeek.DeepSeekOptions.ApiKey = "API_KEY";
DeepSeek.OnHTTPAPISSE += OnSSEEvent;
DeepSeek._CreateMessageStream("deepseek-chat", "Tell me a story.");

void OnSSEEvent(object Sender, string aEvent, string aData, ref bool Cancel)
{
    // aEvent contains the event type
    // aData contains the JSON data for this event
    Memo1.Lines.Add(aData);
}
```


DeepSeek | Vision

DeepSeek può comprendere e analizzare immagini. Lei può inviare immagini come dati codificati in base64 all'interno dei blocchi di contenuto.

Esempio Vision

Invia un'immagine con un prompt che chiede a DeepSeek di descriverla.

```
TsgcHTTP_API_DeepSeek DeepSeek = new TsgcHTTP_API_DeepSeek();
DeepSeek.DeepSeekOptions.ApiKey = "API_KEY";

// Load image and encode to base64
byte[] fileBytes = System.IO.File.ReadAllBytes("photo.jpg");
string vImageBase64 = Convert.ToBase64String(fileBytes);

MessageBox.Show(DeepSeek._CreateVisionMessage("deepseek-chat",
    "Describe this image", vImageBase64, "image/jpeg"));
```

DeepSeek | Modelli

Elenca i modelli DeepSeek disponibili.

Elenca modelli

Elenca tutti i modelli DeepSeek disponibili.

```
TsgcHTTP_API_DeepSeek DeepSeek = new TsgcHTTP_API_DeepSeek();  
DeepSeek.DeepSeekOptions.ApiKey = "API_KEY";  
MessageBox.Show(DeepSeek._GetModels());
```

Gemini

Google Gemini è una famiglia di modelli AI multimodali sviluppati da Google DeepMind. I modelli Gemini supportano generazione di testo, visione, output strutturati, embedding e uso di strumenti, offrendo potenti funzionalità per la creazione di applicazioni basate sull'AI.

La libreria `sgcWebSockets` fornisce un componente Delphi **TsgcHTTP_API_Gemini** per interagire con l'API Gemini.

API Gemini

L'API Gemini fornisce accesso ai modelli Google Gemini per la creazione di applicazioni basate sull'intelligenza artificiale. L'API supporta la generazione di contenuti, la visione (comprensione delle immagini), output JSON strutturati, streaming, conteggio dei token, embedding, uso di strumenti (function calling) ed elenco dei modelli.

Funzionalità

- **Messaggi**
 - [Esempi di messaggi Gemini](#)
- **Vision**
 - [Esempi Gemini Vision](#)
- **Modelli**
 - [Esempi di Modelli Gemini](#)
- **Structured Outputs**
 - [Esempi di Structured Outputs Gemini](#)
- **Conteggio Token**
 - [Esempi di Conteggio Token Gemini](#)
- **Embeddings**
 - [Esempi di Gemini Embeddings](#)
- **Tool Use**
 - [Esempi di Gemini Tool Use](#)

Configurazione

L'API Gemini utilizza chiavi API per l'autenticazione. Visitare la pagina [Chiavi API](#) in Google AI Studio per recuperare la chiave API da utilizzare nelle richieste.

Ricordare che la propria chiave API è segreta! Non condividerla con altri né esporla in alcun codice lato client. Questa **API Key** deve essere configurata nella proprietà **GeminiOptions.ApiKey** del componente.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "YOUR_API_KEY";
```

Messaggi

Invia contenuto a un modello Gemini e riceve le risposte generate. Il modello genera il messaggio successivo in base all'input fornito.

- **_CreateContent:** Crea contenuto con il modello e il prompt utente specificati.
 - **Model:** Il modello da utilizzare (es. gemini-2.0-flash).
 - **Message:** Il contenuto del messaggio utente.
 - **MaxOutputTokens:** Numero massimo di token da generare.
- **_CreateContentWithSystem:** Crea contenuto con un'istruzione di sistema.
 - **Model:** Il modello da utilizzare.
 - **System:** Istruzione di sistema che definisce il comportamento del modello.
 - **Message:** Il contenuto del messaggio dell'utente.
 - **MaxOutputTokens:** Numero massimo di token da generare.
- **_CreateContentStream:** Crea contenuti con lo streaming (SSE) abilitato. Gli eventi vengono forniti tramite il gestore eventi `OnHTTPAPISSE`.

Vision

I modelli Gemini possono elaborare immagini trasmesse come contenuto codificato in base64 insieme a prompt testuali.

- **_CreateVisionContent:** Invia un'immagine con un prompt testuale.
 - **Model:** Il modello da utilizzare.
 - **Prompt:** Il prompt testuale da accompagnare all'immagine.
 - **Base64:** I dati dell'immagine codificati in base64.
 - **MediaType:** Il tipo MIME (image/jpeg, image/png, image/gif, image/webp).
 - **MaxOutputTokens:** Numero massimo di token da generare.

Output Strutturati

Genera output JSON strutturato da un modello Gemini fornendo uno schema JSON che definisce il formato di risposta atteso.

- **_CreateContentJSON:** Crea contenuto con output JSON strutturato.
 - **Model:** Il modello da utilizzare.
 - **Message:** Il contenuto del messaggio dell'utente.
 - **Schema:** Uno schema JSON che definisce la struttura di output prevista.
 - **MaxOutputTokens:** Numero massimo di token da generare.

Models

Elencare e recuperare dettagli sui modelli Gemini disponibili.

- **_GetModels:** Elenca tutti i modelli disponibili.
- **_GetModel:** recupera i dettagli di un modello specifico.
 - **ModelId:** L'identificatore del modello da recuperare.

Conteggio token

Conta il numero di token in un messaggio prima di inviarlo a un modello.

- **_CountTokens:** Conta i token per un messaggio.
 - **Model:** Il modello da utilizzare per la tokenizzazione.
 - **Message:** Il contenuto testuale di cui contare i token.

Embeddings

Genera embedding vettoriali per contenuti testuali utilizzando i modelli Gemini.

- **_EmbedContent:** genera embedding per il testo.
 - **Model:** il modello da utilizzare per la generazione degli embedding.
 - **Text:** il contenuto di testo per cui generare gli embedding.

Gemini | Messaggi

Invia un prompt a un modello Gemini e riceve contenuto generato. L'API Gemini utilizza l'endpoint `generateContent` per elaborare l'input testuale e restituire le risposte del modello.

Esempio Semplice

Invia un messaggio Hello a Gemini.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "API_KEY";
MessageBox.Show(Gemini._CreateContent("gemini-2.0-flash", "Hello!"));
```

Esempio di istruzione di sistema

Invia un messaggio con un'istruzione di sistema per controllare il comportamento di Gemini.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "API_KEY";
MessageBox.Show(Gemini._CreateContentWithSystem("gemini-2.0-flash",
    "You are a helpful assistant that responds in Spanish.",
    "What is the capital of France?"));
```

Esempio di streaming

Utilizzare Server-Sent Events (SSE) per trasmettere la risposta in tempo reale. Assegnare il gestore dell'evento `OnHTTPAPISSE` per ricevere gli eventi in streaming.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "API_KEY";
Gemini.OnHTTPAPISSE += OnSSEEvent;
Gemini._CreateContentStream("gemini-2.0-flash", "Tell me a story.");

void OnSSEEvent(object Sender, string aEvent, string aData, ref bool Cancel)
{
    Memo1.Lines.Add(aData);
}
```

Esempio avanzato

Utilizzare le classi di richiesta/risposta tipizzate per un controllo completo sui parametri di generazione del contenuto.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "API_KEY";

TsgcGeminiClass_Request_GenerateContent oRequest = new TsgcGeminiClass_Request_GenerateContent();
oRequest.Model = "gemini-2.0-flash";
oRequest.MaxOutputTokens = 4096;
oRequest.Temperature = 0.7;
oRequest.SystemInstruction = "You are a creative writer.";

SetLength(oContents, 1);
oContents[0] = new TsgcGeminiClass_Request_Content();
oContents[0].Role = "user";
SetLength(oParts, 1);
oParts[0] = new TsgcGeminiClass_Request_Part();
oParts[0].Text = "Write a haiku about programming.";
oContents[0].Parts = oParts;
oRequest.Contents = oContents;

var oResponse = Gemini.CreateContent(oRequest);
if (oResponse.Candidates.Length > 0)
```

```
if (oResponse.Candidates[0].Parts.Length > 0)
    MessageBox.Show(oResponse.Candidates[0].Parts[0].Text);
```

Gemini | Visione

Gemini è in grado di comprendere e analizzare immagini. È possibile inviare immagini come dati codificati in base64 all'interno di parti di contenuto.

Esempio Vision

Inviare un'immagine codificata in base64 a Gemini per l'analisi.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "API_KEY";

// Load image and encode to base64
byte[] fileBytes = System.IO.File.ReadAllBytes("photo.jpg");
string vImageBase64 = Convert.ToBase64String(fileBytes);

MessageBox.Show(Gemini._CreateVisionContent("gemini-2.0-flash",
    "Describe this image", vImageBase64, "image/jpeg"));
```

Gemini | Models

Elenca e recupera informazioni sui modelli Gemini disponibili.

Elenca modelli

Elenca tutti i modelli disponibili.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "API_KEY";
MessageBox.Show(Gemini._GetModels());
```

Ottieni modello

Ottieni i dettagli per un modello specifico.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "API_KEY";
MessageBox.Show(Gemini._GetModel("gemini-2.0-flash"));
```

Esempio avanzato

Utilizzare le classi di risposta tipizzate per il controllo completo sui dati del modello.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "API_KEY";

TsgcGeminiClass_Response_Models oModels = Gemini.GetModels();
for (int i = 0; i < oModels.Models.Length; i++)
    MessageBox.Show(oModels.Models[i].DisplayName + " - " + oModels.Models[i].Description);
```


Gemini | Output strutturati

Gemini può restituire risposte in formato JSON strutturato conforme a uno schema da Lei definito. Ciò garantisce un output JSON valido e analizzabile che corrisponde alla definizione del Suo schema.

Esempio Semplice

Utilizzare il metodo di convenienza per creare contenuti con output in schema JSON.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "API_KEY";

string vSchema = "{\n\"type\": \"object\", \"properties\": {\n\"name\": {\n\"type\": \"string\"}, \" +
    \"age\": {\n\"type\": \"integer\"}}, \"required\": [\n\"name\", \"age\"]}";

MessageBox.Show(Gemini._CreateContentJSON("gemini-2.0-flash",
    "Extract the name and age: John is 30 years old.", vSchema));
```

Esempio avanzato

Utilizzare le classi tipizzate per configurare l'output strutturato con ResponseMimeType e ResponseSchema.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "API_KEY";

TsgcGeminiClass_Request_GenerateContent oRequest = new TsgcGeminiClass_Request_GenerateContent();
oRequest.Model = "gemini-2.0-flash";
oRequest.MaxOutputTokens = 4096;
oRequest.ResponseMimeType = "application/json";
oRequest.ResponseSchema =
    "{\n\"type\": \"object\", \"properties\": {\n\"sentiment\": {\n\"type\": \"string\"}, \" +
    \"confidence\": {\n\"type\": \"number\"}}, \"required\": [\n\"sentiment\", \"confidence\"]}";

SetLength(oContents, 1);
oContents[0] = new TsgcGeminiClass_Request_Content();
oContents[0].Role = "user";
SetLength(oParts, 1);
oParts[0] = new TsgcGeminiClass_Request_Part();
oParts[0].Text = "Analyze the sentiment: I love this product!";
oContents[0].Parts = oParts;
oRequest.Contents = oContents;

var oResponse = Gemini.CreateContent(oRequest);
if (oResponse.Candidates.Length > 0)
    if (oResponse.Candidates[0].Parts.Length > 0)
        MessageBox.Show(oResponse.Candidates[0].Parts[0].Text);
```

Proprietà

- **ResponseMimeType:** Impostare su 'application/json' per abilitare l'output JSON strutturato.
- **ResponseSchema:** Una stringa JSON Schema che definisce la struttura di output prevista.

Gemini | Token Counting

Contare il numero di token in un messaggio prima di inviarlo all'API. Questo aiuta a stimare i costi e a garantire che i messaggi rientrino nei limiti del modello.

Esempio Semplice

Conta i token per un messaggio.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "API_KEY";
MessageBox.Show(Gemini._CountTokens("gemini-2.0-flash", "Hello, how are you?"));
```

Esempio avanzato

Utilizzare le classi di richiesta/risposta tipizzate per il controllo completo dei parametri di conteggio dei token.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "API_KEY";

TsgcGeminiClass_Request_CountTokens oRequest = new TsgcGeminiClass_Request_CountTokens();
oRequest.Model = "gemini-2.0-flash";
var oContents = new TsgcGeminiClass_Request_Content[1];
oContents[0] = new TsgcGeminiClass_Request_Content();
oContents[0].Role = "user";
var oParts = new TsgcGeminiClass_Request_Part[1];
oParts[0] = new TsgcGeminiClass_Request_Part();
oParts[0].Text = "Explain quantum computing in simple terms.";
oContents[0].Parts = oParts;
oRequest.Contents = oContents;

TsgcGeminiClass_Response_CountTokens oResponse = Gemini.CountTokens(oRequest);
MessageBox.Show("Total tokens: " + oResponse.TotalTokens.ToString());
```

Gemini | Embeddings

Genera embedding vettoriali per contenuti testuali. Gli embedding catturano il significato semantico del testo e possono essere utilizzati per la ricerca per similarità, il clustering e altri task NLP.

Esempio Semplice

Generare embedding per il testo.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "API_KEY";
MessageBox.Show(Gemini._EmbedContent("text-embedding-004", "Hello world"));
```

Esempio avanzato

Utilizzare le classi di risposta tipizzate per un controllo completo sui dati di embedding.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "API_KEY";

TsgcGeminiClass_Response_Embedding oEmbedding = Gemini.EmbedContent("text-embedding-004", "Hello world");
MessageBox.Show("Dimensions: " + oEmbedding.Values.Length.ToString());
for (int i = 0; i < oEmbedding.Values.Length; i++)
    MessageBox.Show(oEmbedding.Values[i].ToString());
```

Gemini | Utilizzo degli strumenti

Gemini supporta la chiamata di funzioni (uso degli strumenti), consentendo di definire funzioni che il modello può invocare durante una conversazione. Quando Gemini decide di chiamare una funzione, restituisce una parte `functionCall` nella sua risposta. Si esegue quindi la funzione e si invia il risultato indietro.

Flusso di chiamata delle funzioni

1. Definire le dichiarazioni di funzione con nome, descrizione e parametri (JSON Schema).
2. Inviare una richiesta di generazione di contenuti con dichiarazioni di funzione.
3. Gemini risponde con una parte **functionCall** contenente il nome della funzione e gli argomenti.
4. Esegue la funzione con gli argomenti forniti.
5. Inviare una nuova richiesta con una parte **functionResponse** contenente il risultato.
6. Gemini risponde con la risposta finale.

Esempio

Definire una funzione meteo e gestire le chiamate di funzione.

```
TsgcHTTP_API_Gemini Gemini = new TsgcHTTP_API_Gemini();
Gemini.GeminiOptions.ApiKey = "API_KEY";

// Step 1: Create request with function declarations
TsgcGeminiClass_Request_GenerateContent oRequest = new TsgcGeminiClass_Request_GenerateContent();
oRequest.Model = "gemini-2.0-flash";
oRequest.MaxOutputTokens = 4096;

// Define user message
SetLength(oContents, 1);
oContents[0] = new TsgcGeminiClass_Request_Content();
oContents[0].Role = "user";
SetLength(oParts, 1);
oParts[0] = new TsgcGeminiClass_Request_Part();
oParts[0].Text = "What is the weather in San Francisco?";
oContents[0].Parts = oParts;
oRequest.Contents = oContents;

// Define function
SetLength(oFunctions, 1);
oFunctions[0] = new TsgcGeminiClass_Request_FunctionDeclaration();
oFunctions[0].Name = "get_weather";
oFunctions[0].Description = "Get the current weather in a given location";
oFunctions[0].Parameters =
    "{\"type\":\"object\",\"properties\":{\"location\":{\"type\":\"string\", \" +
    \"description\":\"The city and state\"}},\"required\":[\"location\"]}";
oRequest.FunctionDeclarations = oFunctions;

// Step 2: Send request
var oResponse = Gemini.CreateContent(oRequest);
// Step 3: Check for function call
if (oResponse.Candidates.Length > 0) {
    if (oResponse.Candidates[0].Parts.Length > 0) {
        if (oResponse.Candidates[0].Parts[0].FunctionCallName != "") {
            var vFunctionName = oResponse.Candidates[0].Parts[0].FunctionCallName;
            var vFunctionArgs = oResponse.Candidates[0].Parts[0].FunctionCallArgs;
            // Step 4: Execute function and get result
            var vResult = "72 degrees and sunny";
        }
    }
}
```

Proprietà

- **FunctionDeclarations:** Array di `TsgcGeminiClass_Request_FunctionDeclaration` che definisce le funzioni disponibili.
 - **Name:** Il nome della funzione.
 - **Description:** Una descrizione di ciò che fa la funzione.
 - **Parameters:** Stringa JSON Schema che definisce i parametri della funzione.

- **ToolChoice:** Controlla il comportamento della chiamata alle funzioni. Imposta la modalità (ad es. 'AUTO', 'ANY', 'NONE').

Ollama

Ollama è uno strumento open source per eseguire modelli linguistici di grandi dimensioni in locale. Supporta un'ampia gamma di modelli tra cui Llama, Mistral, Gemma, Phi e molti altri, consentendo l'inferenza AI locale senza richiedere l'accesso ad API cloud.

La libreria sgcWebSockets fornisce un componente Delphi **TsgcHTTP_API_Ollama** per interagire con le API Ollama.

Ollama API

L'API Ollama fornisce accesso a modelli in esecuzione locale per applicazioni basate sull'intelligenza artificiale. L'API supporta la generazione di testo, lo streaming, la gestione dei modelli e gli embedding. Per impostazione predefinita non è richiesta alcuna chiave API poiché i modelli vengono eseguiti localmente. L'host e la porta sono configurabili.

Funzionalità

- **Messaggi**
 - [Esempi Messaggi Ollama](#)
- **Modelli**
 - [Esempi di modelli Ollama](#)
- **Embeddings**
 - [Esempi di Ollama Embeddings](#)

Configurazione

Ollama viene eseguito localmente e per impostazione predefinita è in ascolto su **http://localhost:11434**. Configurare l'host nella proprietà **OllamaOptions.Host**. Una chiave API è opzionale e necessaria solo se è stata configurata l'autenticazione sulla propria istanza Ollama.

```
TsgcHTTP_API_Ollama Ollama = new TsgcHTTP_API_Ollama();
Ollama.OllamaOptions.Host = "http://localhost:11434";
```

Messaggi

Inviare un elenco strutturato di messaggi di input con contenuto testuale e il modello genererà il prossimo messaggio nella conversazione.

- **_CreateMessage**: Crea un messaggio con il modello e il prompt utente specificati.
 - **Model**: Il modello da utilizzare (es. llama3).
 - **Message**: Il contenuto del messaggio dell'utente.
- **_CreateMessageWithSystem**: Crea un messaggio con un prompt di sistema.
 - **System**: Prompt di sistema che definisce il comportamento dell'assistente.
- **_CreateMessageStream**: Crea un messaggio con streaming (SSE) abilitato. Gli eventi vengono consegnati tramite il gestore dell'evento OnHTTPAPISSE.

Models

Gestisce ed esegue query sui modelli disponibili localmente.

- **_GetModels**: Elenca tutti i modelli disponibili.
- **_GetTags**: Elenca tutti i tag dei modelli disponibili localmente.
- **_ShowModel**: Recupera informazioni dettagliate su un modello specifico.
 - **Model**: Il nome del modello da interrogare.
- **_PullModel**: Scarica un modello dalla libreria di modelli Ollama.
 - **Model**: Il nome del modello da scaricare.
- **_DeleteModel**: Elimina un modello disponibile localmente.
 - **Model**: Il nome del modello da eliminare.

Embeddings

Genera embedding vettoriali da testo di input utilizzando modelli in esecuzione localmente.

- **_CreateEmbeddings:** Genera embedding per il testo fornito.
 - **Model:** Il modello da utilizzare per gli embedding (es. llama3).
 - **Input:** Il testo per cui generare gli embedding.

Ollama | Messaggi

Inviare un elenco strutturato di messaggi di input con contenuto testuale e il modello genererà il prossimo messaggio nella conversazione.

Esempio Semplice

Inviare un messaggio Hello a un modello Ollama locale.

```
TsgcHTTP_API_Ollama Ollama = new TsgcHTTP_API_Ollama();
Ollama.OllamaOptions.Host = "http://localhost:11434";
MessageBox.Show(Ollama._CreateMessage("llama3", "Hello!"));
```

Esempio di prompt di sistema

Inviare un messaggio con un prompt di sistema per controllare il comportamento del modello.

```
TsgcHTTP_API_Ollama Ollama = new TsgcHTTP_API_Ollama();
Ollama.OllamaOptions.Host = "http://localhost:11434";
MessageBox.Show(Ollama._CreateMessageWithSystem("llama3",
    "You are a helpful assistant that responds in Spanish.",
    "What is the capital of France?"));
```

Esempio di streaming

Utilizzare Server-Sent Events (SSE) per trasmettere la risposta in tempo reale. Assegnare il gestore dell'evento OnHTTPAPISSE per ricevere gli eventi in streaming.

```
TsgcHTTP_API_Ollama Ollama = new TsgcHTTP_API_Ollama();
Ollama.OllamaOptions.Host = "http://localhost:11434";
Ollama.OnHTTPAPISSE += OnSSEEvent;
Ollama._CreateMessageStream("llama3", "Tell me a story.");

void OnSSEEvent(object Sender, string aEvent, string aData, ref bool Cancel)
{
    // aEvent contains the event type
    // aData contains the JSON data for this event
    Memo1.Lines.Add(aData);
}
```


Ollama | Modelli

Gestire e interrogare i modelli Ollama disponibili localmente.

Elenca modelli

Elenca tutti i modelli disponibili.

```
TsgcHTTP_API_Ollama Ollama = new TsgcHTTP_API_Ollama();  
Ollama.OllamaOptions.Host = "http://localhost:11434";  
MessageBox.Show(Ollama._GetModels());
```

Ottieni Tag

Elenca tutti i tag dei modelli disponibili localmente.

```
TsgcHTTP_API_Ollama Ollama = new TsgcHTTP_API_Ollama();  
Ollama.OllamaOptions.Host = "http://localhost:11434";  
MessageBox.Show(Ollama._GetTags());
```

Mostra Modello

Recupera informazioni dettagliate su un modello specifico.

```
TsgcHTTP_API_Ollama Ollama = new TsgcHTTP_API_Ollama();  
Ollama.OllamaOptions.Host = "http://localhost:11434";  
MessageBox.Show(Ollama._ShowModel("llama3"));
```

Modello pull

Scarica un modello dalla libreria di modelli Ollama.

```
TsgcHTTP_API_Ollama Ollama = new TsgcHTTP_API_Ollama();  
Ollama.OllamaOptions.Host = "http://localhost:11434";  
MessageBox.Show(Ollama._PullModel("llama3"));
```

Ollama | Embedding

Genera embedding vettoriali dall'input testuale utilizzando modelli Ollama in esecuzione locale. Gli embedding possono essere utilizzati per la ricerca semantica, il clustering e altre attività di NLP.

Creare Embeddings

Genera embedding per il testo di input fornito.

```
TsgcHTTP_API_Ollama Ollama = new TsgcHTTP_API_Ollama();  
Ollama.OllamaOptions.Host = "http://localhost:11434";  
MessageBox.Show(Ollama._CreateEmbeddings("llama3", "Hello world"));
```

Grok

Grok è un assistente AI conversazionale sviluppato da xAI, progettato per fornire risposte utili e informative con conoscenze in tempo reale. I modelli Grok sono noti per le loro forti capacità di ragionamento e l'accesso a informazioni aggiornate.

La libreria `sgcWebSockets` fornisce un componente Delphi **TsgcHTTP_API_Grok** per interagire con l'API Grok.

Grok API

L'API Grok fornisce accesso ai modelli Grok per la creazione di applicazioni basate sull'intelligenza artificiale. L'API supporta la generazione di testo, la visione (comprensione delle immagini), lo streaming e l'elencazione dei modelli.

Funzionalità

- **Messaggi**
 - [Esempi di Messaggi Grok](#)
- **Vision**
 - [Esempi Grok Vision](#)
- **Modelli**
 - [Esempi di modelli Grok](#)

Configurazione

L'API Grok utilizza chiavi API per l'autenticazione. Visitare la [Console xAI](#) per recuperare la chiave API da utilizzare nelle richieste.

Ricordare che la propria chiave API è segreta! Non condividerla con altri né esporla in alcun codice lato client.

Questa **API Key** deve essere configurata nella proprietà **GrokOptions.ApiKey** del componente.

```
TsgcHTTP_API_Grok Grok = new TsgcHTTP_API_Grok();
Grok.GrokOptions.ApiKey = "YOUR_API_KEY";
```

Messaggi

Inviare un elenco strutturato di messaggi di input con contenuto testuale e il modello genererà il prossimo messaggio nella conversazione.

- **_CreateMessage:** Crea un messaggio con il modello e il prompt utente specificati.
 - **Model:** Il modello da utilizzare (es. grok-3).
 - **Message:** Il contenuto del messaggio utente.
- **_CreateMessageWithSystem:** Crea un messaggio con un prompt di sistema.
 - **System:** Prompt di sistema che definisce il comportamento dell'assistente.
- **_CreateMessageStream:** Crea un messaggio con streaming (SSE) abilitato. Gli eventi vengono consegnati tramite il gestore dell'evento `OnHTTPAPISSE`.

Vision

Grok è in grado di elaborare immagini passate come contenuto codificato in base64 all'interno dei messaggi.

- **_CreateVisionMessage:** Invia un'immagine con un prompt testuale.
 - **ImageBase64:** I dati dell'immagine codificati in base64.
 - **MediaType:** Il tipo MIME (image/jpeg, image/png, image/gif, image/webp).
 - **Prompt:** Il prompt testuale da accompagnare all'immagine.

Models

Elenca i modelli Grok disponibili.

- **_GetModels:** Elenca tutti i modelli disponibili.

Grok | Messaggi

Inviare un elenco strutturato di messaggi di input con contenuto testuale e il modello genererà il prossimo messaggio nella conversazione.

Esempio Semplice

Invia un messaggio Hello a Grok.

```
TsgcHTTP_API_Grok Grok = new TsgcHTTP_API_Grok();
Grok.GrokOptions.ApiKey = "API_KEY";
MessageBox.Show(Grok._CreateMessage("grok-3", "Hello!"));
```

Esempio di prompt di sistema

Invia un messaggio con un prompt di sistema per controllare il comportamento di Grok.

```
TsgcHTTP_API_Grok Grok = new TsgcHTTP_API_Grok();
Grok.GrokOptions.ApiKey = "API_KEY";
MessageBox.Show(Grok._CreateMessageWithSystem("grok-3",
    "You are a helpful assistant that responds in Spanish.",
    "What is the capital of France?"));
```

Esempio di streaming

Utilizzare Server-Sent Events (SSE) per trasmettere la risposta in tempo reale. Assegnare il gestore dell'evento OnHTTPAPISSE per ricevere gli eventi in streaming.

```
TsgcHTTP_API_Grok Grok = new TsgcHTTP_API_Grok();
Grok.GrokOptions.ApiKey = "API_KEY";
Grok.OnHTTPAPISSE += OnSSEEvent;
Grok._CreateMessageStream("grok-3", "Tell me a story.");

void OnSSEEvent(object Sender, string aEvent, string aData, ref bool Cancel)
{
    Memo1.Lines.Add(aData);
}
```

Grok | Visione

Grok è in grado di comprendere le immagini passate come contenuto codificato in base64 all'interno dei messaggi. Utilizzare l'endpoint vision per inviare un'immagine insieme a un prompt testuale e ricevere una descrizione o un'analisi.

Esempio Vision

Invia un'immagine codificata in base64 a Grok per l'analisi.

```
TsgcHTTP_API_Grok Grok = new TsgcHTTP_API_Grok();
Grok.GrokOptions.ApiKey = "API_KEY";
MessageBox.Show(Grok._CreateVisionMessage("grok-2-vision",
    "Describe this image.", vBase64, "image/jpeg"));
```

Grok | Models

Elenca i modelli Grok disponibili e le loro informazioni di base.

Esempio di Elenco Modelli

Recupera l'elenco di tutti i modelli Grok disponibili.

```
TsgcHTTP_API_Grok Grok = new TsgcHTTP_API_Grok();  
Grok.GrokOptions.ApiKey = "API_KEY";  
MessageBox.Show(Grok._GetModels);
```

Mistral AI

Mistral AI è un'azienda francese di intelligenza artificiale che sviluppa modelli linguistici efficienti e potenti. I suoi modelli sono noti per le elevate prestazioni nel ragionamento, nella programmazione e nei compiti multilingue, mantenendo al contempo un'efficienza competitiva.

La libreria sgcWebSockets fornisce un componente Delphi **TsgcHTTP_API_Mistral** per interagire con l'API Mistral AI.

Mistral API

L'API Mistral fornisce accesso ai modelli Mistral per la creazione di applicazioni basate sull'IA. L'API supporta la generazione di testo, la visione (comprensione delle immagini), lo streaming, gli embedding, la modalità JSON e l'elenco dei modelli.

Funzionalità

- **Messaggi**
 - [Esempi di messaggi Mistral](#)
- **Visione**
 - [Esempi di visione con Mistral](#)
- **Modelli**
 - [Esempi di modelli Mistral](#)
- **Embeddings**
 - [Esempi di Mistral Embeddings](#)

Configurazione

L'API Mistral utilizza le API key per l'autenticazione. Visiti la Sua pagina [API Keys](#) nella Mistral Console per recuperare la API key da utilizzare nelle Sue richieste.

Ricordare che la propria chiave API è segreta! Non condividerla con altri né esporla in alcun codice lato client.

Questa **API Key** deve essere configurata nella proprietà **MistralOptions.ApiKey** del componente.

```
TsgcHTTP_API_Mistral Mistral = new TsgcHTTP_API_Mistral();
Mistral.MistralOptions.ApiKey = "YOUR_API_KEY";
```

Messaggi

Inviare un elenco strutturato di messaggi di input con contenuto testuale e il modello genererà il prossimo messaggio nella conversazione.

- **_CreateMessage:** Crea un messaggio con il modello e il prompt utente specificati.
 - **Model:** Il modello da utilizzare (ad esempio mistral-large-latest).
 - **Message:** Il contenuto del messaggio dell'utente.
- **_CreateMessageWithSystem:** Crea un messaggio con un prompt di sistema.
 - **System:** Prompt di sistema che definisce il comportamento dell'assistente.
- **_CreateMessageStream:** Crea un messaggio con streaming (SSE) abilitato. Gli eventi vengono consegnati tramite il gestore dell'evento OnHTTPAPISSE.
- **_CreateMessageJSON:** Crea un messaggio con la modalità JSON abilitata. Il modello restituirà una risposta JSON valida.
 - **Model:** Il modello da utilizzare.
 - **Message:** Il contenuto del messaggio utente.

Vision

Mistral può comprendere immagini passate come contenuto codificato in base64 all'interno dei messaggi.

- **_CreateVisionMessage:** Invia un'immagine con un prompt testuale.
 - **ImageBase64:** I dati dell'immagine codificati in base64.

- **MediaType:** Il tipo MIME (image/jpeg, image/png, image/gif, image/webp).
- **Prompt:** Il prompt testuale da accompagnare all'immagine.

Models

Elenca i modelli Mistral disponibili.

- **_GetModels:** Elenca tutti i modelli disponibili.

Embeddings

Ottieni una rappresentazione vettoriale di un determinato input che può essere utilizzata per la ricerca semantica, il clustering e altre attività di machine learning.

- **_CreateEmbeddings:** Crea un vettore di embedding che rappresenta il testo di input.
 - **Model:** Il modello da utilizzare (es. mistral-embed).
 - **Input:** Testo di input per cui ottenere gli embedding.

Mistral | Messaggi

Inviare un elenco strutturato di messaggi di input con contenuto testuale e il modello genererà il prossimo messaggio nella conversazione.

Esempio Semplice

Invia un messaggio Hello a Mistral.

```
TsgcHTTP_API_Mistral Mistral = new TsgcHTTP_API_Mistral();
Mistral.MistralOptions.ApiKey = "API_KEY";
MessageBox.Show(Mistral._CreateMessage("mistral-large-latest", "Hello!"));
```

Esempio di prompt di sistema

Invia un messaggio con un prompt di sistema per controllare il comportamento di Mistral.

```
TsgcHTTP_API_Mistral Mistral = new TsgcHTTP_API_Mistral();
Mistral.MistralOptions.ApiKey = "API_KEY";
MessageBox.Show(Mistral._CreateMessageWithSystem("mistral-large-latest",
    "You are a helpful assistant that responds in Spanish.",
    "What is the capital of France?"));
```

Esempio di streaming

Utilizzare Server-Sent Events (SSE) per trasmettere la risposta in tempo reale. Assegnare il gestore dell'evento OnHTTPAPISSE per ricevere gli eventi in streaming.

```
TsgcHTTP_API_Mistral Mistral = new TsgcHTTP_API_Mistral();
Mistral.MistralOptions.ApiKey = "API_KEY";
Mistral.OnHTTPAPISSE += OnSSEEvent;
Mistral._CreateMessageStream("mistral-large-latest", "Tell me a story.");

void OnSSEEvent(object Sender, string aEvent, string aData, ref bool Cancel)
{
    Memo1.Lines.Add(aData);
}
```

Esempio modalità JSON

Richiedere una risposta in formato JSON da Mistral.

```
TsgcHTTP_API_Mistral Mistral = new TsgcHTTP_API_Mistral();
Mistral.MistralOptions.ApiKey = "API_KEY";
MessageBox.Show(Mistral._CreateMessageJSON("mistral-large-latest",
    "List 3 colors as JSON"));
```

Mistral | Vision

Mistral è in grado di comprendere immagini passate come contenuto codificato in base64 all'interno dei messaggi. Utilizzi l'endpoint vision per inviare un'immagine insieme a un prompt di testo e ricevere una descrizione o un'analisi.

Esempio Vision

Invia un'immagine in formato base64 a Mistral per l'analisi.

```
TsgcHTTP_API_Mistral Mistral = new TsgcHTTP_API_Mistral();
Mistral.MistralOptions.ApiKey = "API_KEY";
MessageBox.Show(Mistral._CreateVisionMessage("mistral-large-latest",
    "Describe this image.", vBase64, "image/jpeg"));
```

Mistral | Modelli

Elenca i modelli Mistral disponibili e le relative informazioni di base.

Esempio di Elenco Modelli

Recupera l'elenco di tutti i modelli Mistral disponibili.

```
TsgcHTTP_API_Mistral Mistral = new TsgcHTTP_API_Mistral();  
Mistral.MistralOptions.ApiKey = "API_KEY";  
MessageBox.Show(Mistral._GetModels);
```

Mistral | Embeddings

Ottieni una rappresentazione vettoriale di un determinato input che può essere utilizzata per la ricerca semantica, il clustering e altre attività di machine learning.

Embeddings Example

Crea un vettore di embedding per un input di testo.

```
TsgcHTTP_API_Mistral Mistral = new TsgcHTTP_API_Mistral();  
Mistral.MistralOptions.ApiKey = "API_KEY";  
MessageBox.Show(Mistral._CreateEmbeddings("mistral-embed", "Hello world"));
```

IoT

L'Internet delle cose (IoT) si riferisce al concetto di estendere la connettività Internet oltre le piattaforme informatiche convenzionali, come personal computer e dispositivi mobili, fino a qualsiasi gamma di dispositivi fisici e oggetti di uso quotidiano tradizionalmente "non intelligenti" o non connessi a Internet. Dotati di elettronica, connettività Internet e altre forme di hardware (come sensori), questi dispositivi possono comunicare e interagire con altri tramite Internet e possono essere monitorati e controllati da remoto.

Il pacchetto `sgcWebSockets` implementa i seguenti client IoT:

1. Amazon AWS IoT: AWS IoT fornisce una comunicazione sicura e bidirezionale tra dispositivi connessi a Internet come sensori, attuatori, microcontroller integrati o elettrodomestici intelligenti e il cloud AWS. Ciò consente di raccogliere dati di telemetria da più dispositivi, nonché di archivarli e analizzarli. È inoltre possibile creare applicazioni che permettano agli utenti di controllare questi dispositivi dai loro telefoni o tablet.

2. Azure IoT Hub: IoT Hub è un servizio gestito, ospitato nel cloud, che funge da hub centrale per la comunicazione bidirezionale tra l'applicazione IoT e i dispositivi che gestisce. È possibile utilizzare Azure IoT Hub per creare soluzioni IoT con comunicazioni affidabili e sicure tra milioni di dispositivi IoT e un backend di soluzione ospitato nel cloud.

IoT Amazon MQTT Client

Che cos'è AWS IoT?

AWS IoT fornisce comunicazione sicura e bidirezionale tra dispositivi connessi a Internet come sensori, attuatori, microcontrollori incorporati o elettrodomestici intelligenti e il Cloud AWS. Ciò consente di raccogliere dati di telemetria da più dispositivi, archivarli e analizzarli. È anche possibile creare applicazioni che consentano agli utenti di controllare questi dispositivi dai propri telefoni o tablet.

Message broker

Fornisce un meccanismo sicuro per dispositivi e applicazioni AWS IoT per pubblicare e ricevere messaggi l'uno dall'altro. È possibile utilizzare direttamente il protocollo MQTT oppure MQTT su WebSocket per pubblicare e sottoscrivere.

L'AWS IoT message broker è un servizio broker publish/subscribe che consente l'invio e la ricezione di messaggi da e verso AWS IoT. Quando comunica con AWS IoT, un client invia un messaggio indirizzato a un topic come Sensor/temp/room1.

Il message broker, a sua volta, invia il messaggio a tutti i client che si sono registrati per ricevere messaggi su quel topic. L'atto di inviare il messaggio è denominato pubblicazione. L'atto di registrarsi per ricevere messaggi su un filtro di topic è denominato sottoscrizione.

Lo spazio dei nomi dei topic è isolato per ogni coppia account e regione AWS. Per esempio, il topic Sensor/temp/room1 per un account AWS è indipendente dal topic Sensor/temp/room1 per un altro account AWS. Questo vale anche per le regioni. Il topic Sensor/temp/room1 nello stesso account AWS in us-east-1 è indipendente dallo stesso topic in us-east-2. AWS IoT non supporta l'invio e la ricezione di messaggi tra account e regioni AWS diversi.

Il broker di messaggi mantiene un elenco di tutte le sessioni client e delle sottoscrizioni per ciascuna sessione. Quando viene pubblicato un messaggio su un argomento, il broker verifica le sessioni con sottoscrizioni corrispondenti all'argomento. Il broker inoltra quindi il messaggio pubblicato a tutte le sessioni che hanno un client attualmente connesso.

Client MQTT

TsgcIoTAmazon_MQTT_Client è il componente utilizzato per connettersi ad AWS IoT. Un client può connettersi a un solo dispositivo. Il client si connette utilizzando il protocollo MQTT semplice e si autentica utilizzando un certificato client X.509.

Per connettersi ad AWS IoT, il client necessita delle seguenti proprietà:

Amazon.ClientId: identificazione del client, facoltativa.

Amazon.Endpoint: nome del server a cui si conatterà il client MQTT.

Amazon.Port: per impostazione predefinita utilizza la porta 8883. Se la porta è 443, utilizza ALPN automaticamente per la connessione (richiede la versione Indy personalizzata).

AWS IoT Core supporta dispositivi e client che utilizzano i protocolli MQTT e MQTT over WebSocket Secure (WSS) per pubblicare e sottoscrivere messaggi. La tabella seguente elenca i protocolli supportati dagli endpoint del dispositivo AWS IoT e i metodi di autenticazione e le porte utilizzati.

Protocol	Authenticati- on	Port	Nome del protocollo ALPN
MQTT su WebSocket	Signature Version 4	443	

MQTT su WebSocket	Autenticazione personalizzata	443	
MQTT	Certificato client X.509	443	x-amzn-mqtt-ca
MQTT	Certificato client X.509	8883	
MQTT	Autenticazione personalizzata	443	mqtt

Autenticazione tramite certificati

È necessario creare certificati nella console Amazon AWS e impostare il percorso in cui sono archiviati.

Utilizzando **OpenSSL** come IOHandler è necessario impostare il certificato nei seguenti percorsi

Certificate.Enabled: imposti su True se desidera utilizzare i certificati.

Certificate.CertFile: percorso del certificato client X.509.

Certificate.KeyFile: percorso del file chiave client X.509.

Utilizzando **SChannel** come IOHandler, convertire prima il certificato PEM + chiave in un certificato PFX. Questa operazione richiede i binari OpenSSL:

```
openssl pkcs12 -inkey 884ccf73ff-private.pem.key -in 884ccf73ff-certificate.pem.crt -export -out 884ccf73ff-certi
```

Quindi impostare i seguenti percorsi (non è necessario impostare il file della chiave perché è già incluso nel certificato).

Certificate.Enabled: imposti su True se desidera utilizzare i certificati.

Certificate.CertFile: percorso del certificato PFX

Autenticazione SignatureV4

È necessario creare un utente nella console Amazon AWS e salvare le chiavi di accesso e segrete, che verranno utilizzate per firmare la richiesta WebSocket.

SignatureV4.Enabled: impostare su True se si desidera utilizzare questo tipo di autenticazione.

SignatureV4.Region: la regione in cui si trova il Suo dispositivo (esempio: us-east-1).

SignatureV4.AccessKey: la chiave di accesso creata nella console Amazon o ottenuta come credenziale temporanea.

SignatureV4.SecretKey: la chiave segreta creata nella console Amazon o ottenuta come credenziale temporanea

SignatureV4.SessionToken: (condizionale) se si utilizzano credenziali di sicurezza temporanee, impostare qui il token di sicurezza.

OpenSSL_Options: configurazione delle librerie openssl.

APIVersion: consente di definire quale API OpenSSL verrà utilizzata.

oslAPI_1_0: utilizza l'API OpenSSL 1.0, l'ultima supportata da Indy

oslAPI_1_1: utilizza OpenSSL API 1.1, richiede la nostra libreria Indy personalizzata e consente l'utilizzo delle librerie OpenSSL 1.1.1 (con supporto TLS 1.3).

oslAPI_3_0: utilizza l'API 3.0 OpenSSL, richiede la nostra libreria Indy personalizzata e consente l'uso delle librerie OpenSSL 3.0.0 (con supporto TLS 1.3).

LibPath: qui è possibile configurare la posizione delle librerie openssl

oslpNone: questa è l'impostazione predefinita; le librerie openssl devono trovarsi nella stessa cartella del binario o in un percorso noto.

oslpDefaultFolder: imposta automaticamente il percorso openssl dove le librerie devono essere situate per tutte le personalità IDE.

oslpCustomFolder: se questa è l'opzione selezionata, definire il percorso completo nella proprietà LibPathCustom.

LibPathCustom: quando LibPath = oslpCustomFolder, definire qui il percorso completo dove si trovano le librerie openssl.

UnixSymLinks: abilita o disabilita il caricamento dei SymLink sui sistemi Unix (per impostazione predefinita è abilitato, tranne che su OSX64):

oslsSymLinksDefault: per impostazione predefinita sono abilitati, eccetto su OSX64 (dopo MacOS Monterey si verifica un errore nel tentativo di caricare la libreria senza versione.).

oslsSymLinksLoadFirst: Carica i SymLink e li elabora prima di tentare il caricamento delle librerie di versione.

oslsSymLinksLoad: Carica i SymLink dopo aver tentato di caricare le librerie di versione.

oslsSymLinksDontLoad: non caricare i SymLink.

**SignatureV4 richiede Indy 10.5.7+*

Autenticazione personalizzata

L'autenticazione personalizzata consente di definire come autenticare e autorizzare i client utilizzando risorse di autorizzazione. Il dispositivo trasmette le credenziali nei campi header della richiesta o nei parametri di query (per i protocolli MQTT su WebSockets) oppure nei campi nome utente e password del messaggio MQTT CONNECT (per i protocolli MQTT e MQTT su WebSockets).

CustomAuthentication.Enabled: impostare a True se si desidera utilizzare questo tipo di autenticazione.

CustomAuthentication.Parameters: impostare qui i parametri di query che verranno passati al server (per impostazione predefinita è /mqtt)

CustomAuthentication.Headers: qui è possibile inserire i campi di intestazione personalizzati.

CustomAuthentication.WebSockets: se impostato a true, la connessione funzionerà tramite il protocollo WebSocket, altrimenti funzionerà tramite TCP semplice.

MQTTAuthentication.Enabled: se è necessario passare nome utente/password nella connessione mqtt, abilitare questa proprietà

MQTTAuthentication.Username: nome utente della connessione MQTT

MQTTAuthentication.Password: segreto della connessione mqtt.

Il client può inviare facoltativamente un ClientId per identificare la connessione client; altri client possono quindi iscriversi per ricevere una notifica ogni volta che questo client si connette, si iscrive, si disconnette...

Authorization

Se non è possibile connettersi utilizzando la porta 8883 e si utilizza TCP come trasporto (che è quello predefinito), Amazon utilizza la "AWS IoT Core policy" per fornire o meno l'autorizzazione ai client e alle sottoscrizioni. Molto probabilmente è necessario autorizzare il proprio client id.

Accedere alla console Amazon AWS, selezionare IoT Core e aprire il menu "Secure/Policies"; selezionare quindi il criterio collegato al proprio IoT Thing e verificare in fondo come è configurata la connessione. Esempio:

```
{
  "Effect": "Allow",
  "Action": [
    "iot:Connect"
  ],
  "Resource": [
    "arn:aws:iot:us-east-1:222178873557:client/sdk-java",
    "arn:aws:iot:us-east-1:222178873557:client/basicPubSub",
    "arn:aws:iot:us-east-1:222178873557:client/sdk-nodejs-*"
  ]
}
```

Questa configurazione significa che solo i client con ID: sdk-java, basicPubSub e sdk-nodejs-* saranno autorizzati a connettersi. Modificare di conseguenza e riprovare.

Se ancora non funziona, abiliti il log e verificaci in cloudwatch il motivo per cui non riesce a connettersi.

Altre proprietà

MQTTHeartBeat: se abilitato, tenta di mantenere attiva la connessione MQTT inviando un ping ogni x secondi.

Interval: numero di secondi tra ogni ping.

MQTTAuthentication: se abilitato, include nella connessione MQTT il nome utente e la password

UserName: nome dell'utente

Password: stringa segreta

WatchDog: se abilitato, quando viene rilevata una disconnessione inattesa, tenta di riconnettersi automaticamente al server.

Interval: secondi prima dei tentativi di riconnessione.

Attempts: numero massimo di tentativi di riconnessione; zero significa illimitato.

LogFile: se abilitato, salva i messaggi del socket in un file di log (utile per il debugging). L'accesso al file di log non è thread safe se vi si accede da più thread.

Enabled: se abilitato, ogni messaggio ricevuto e inviato tramite socket verrà salvato su un file.

FileName: percorso completo al nome del file.

Implementazione

L'implementazione Amazon MQTT è basata su MQTT versione 3.1.1 ma si discosta dalla specifica come segue:

- In AWS IoT, la sottoscrizione a un argomento con Quality of Service (QoS) 0 significa che un messaggio viene recapitato zero o più volte. Un messaggio potrebbe essere recapitato più di una volta. I messaggi recapitati più di una volta potrebbero essere inviati con un ID pacchetto diverso. In questi casi, il flag DUP non viene impostato.
- AWS IoT non supporta la pubblicazione e la sottoscrizione con QoS 2. Il message broker di AWS IoT non invia un PUBACK o un SUBACK quando viene richiesto QoS 2.
- Quando risponde a una richiesta di connessione, il message broker invia un messaggio CONNACK. Questo messaggio contiene un flag che indica se la connessione sta riprendendo una sessione precedente. Il valore di questo flag potrebbe essere errato se due client MQTT si connettono contemporaneamente con lo stesso client ID.
- Quando un client si iscrive a un argomento, potrebbe verificarsi un ritardo tra il momento in cui il broker di messaggi invia un SUBACK e il momento in cui il client inizia a ricevere nuovi messaggi corrispondenti.
- La specifica MQTT prevede una disposizione per cui il publisher può richiedere al broker di conservare l'ultimo messaggio inviato a un topic e inviarlo a tutti i futuri sottoscrittori del topic. AWS IoT non supporta i messaggi trattenuti (retained). Se viene effettuata una richiesta per trattenere messaggi, la connessione viene disconnessa.
- Il message broker utilizza il client ID per identificare ogni client. Il client ID viene passato dal client al message broker come parte del payload MQTT. Due client con lo stesso client ID non possono essere connessi contemporaneamente al message broker. Quando un client si connette al message broker utilizzando un client ID già in uso da un altro client, un messaggio CONNACK viene inviato a entrambi i client e il client attualmente connesso viene disconnesso.
- In rare occasioni, il message broker potrebbe inviare nuovamente lo stesso messaggio PUBLISH logico con un ID pacchetto diverso.
- Il message broker non garantisce l'ordine in cui vengono ricevuti i messaggi e gli ACK.

Connetti ad AWS IoT

Per prima cosa, deve accedere alla Sua console AWS, registrare un nuovo dispositivo e creare un certificato X.509 per questo dispositivo. Una volta fatto, può creare un nuovo `TsgIoTAmazon_MQTT_Client` e connettersi al server AWS IoT. Ad esempio:

Topics

Il message broker utilizza i topic per instradare i messaggi dai client publisher ai client subscriber. La barra (/) viene utilizzata per separare la gerarchia dei topic. La tabella seguente elenca i caratteri jolly che possono essere utilizzati nel filtro dei topic al momento della sottoscrizione. # Deve essere l'ultimo carattere nel topic a cui ci si iscrive. Funziona come jolly corrispondendo all'albero corrente e a tutti i sottoalberi.

Ad esempio, una sottoscrizione a `Sensor/#` riceve i messaggi pubblicati su `Sensor/`, `Sensor/temp`, `Sensor/temp/room1`, ma non i messaggi pubblicati su `Sensor`.

+ corrisponde esattamente a un elemento nella gerarchia degli argomenti. Ad esempio, una sottoscrizione a `Sensor+/room1` riceve messaggi pubblicati su `Sensor/temp/room1`, `Sensor/moisture/room1` e così via.

Argomenti riservati

I seguenti metodi vengono utilizzati per sottoscrivere/pubblicare su topic riservati.

Subscribe_ClientConnected(const aClientId: String): AWS IoT pubblica su questo topic quando un client MQTT con l'ID client specificato si connette ad AWS IoT

Subscribe_ClientDisconnected(const aClientId: String): AWS IoT pubblica su questo topic quando un client MQTT con l'ID client specificato si disconnette da AWS IoT

Subscribe_ClientSubscribed(const aClientId: String): AWS IoT pubblica su questo argomento quando un client MQTT con l'ID client specificato si iscrive a un argomento MQTT

Subscribe_ClientUnSubscribed(const aClientId: String): AWS IoT pubblica su questo topic quando un client MQTT con il client ID specificato annulla l'iscrizione a un topic MQTT

Publish_Rule(const aRuleName, aText: String): Un dispositivo o un'applicazione pubblica su questo topic per attivare le regole direttamente

Publish_DeleteShadow(const aThingName, aText: String): Un dispositivo o un'applicazione pubblica su questo topic per eliminare un shadow

Subscribe_DeleteShadow(const aThingName: String): Un dispositivo o un'applicazione si iscrive a questo topic per eliminare uno shadow

Subscribe_ShadowDeleted(const aThingName: String): Il servizio Device Shadow invia messaggi a questo topic quando un'ombra viene eliminata

Subscribe_ShadowRejected(const aThingName: String): Il servizio Device Shadow invia messaggi a questo topic quando una richiesta di eliminazione di uno shadow viene rifiutata

Publish_ShadowGet(const aThingName, aText: String): Un'applicazione o un dispositivo pubblica un messaggio vuoto su questo topic per ottenere uno shadow

Subscribe_ShadowGet(const aThingName: String): Un'applicazione o un dispositivo si iscrive a questo topic per ottenere uno shadow

Subscribe_ShadowGetAccepted(const aThingName: String): Il servizio Device Shadow invia messaggi a questo topic quando una richiesta di shadow viene effettuata con successo

Subscribe_ShadowGetRejected(const aThingName: String): Il servizio Device Shadow invia messaggi a questo topic quando una richiesta per uno shadow viene rifiutata

Publish_ShadowUpdate(const aThingName, aText: String): Un oggetto o un'applicazione pubblica su questo argomento per aggiornare un'ombra

Subscribe_ShadowUpdateAccepted(const aThingName: String): Il servizio Device Shadow invia messaggi a questo argomento quando un aggiornamento viene applicato con successo a un shadow

Subscribe_ShadowUpdateRejected(const aThingName: String): Il servizio Device Shadow invia messaggi a questo topic quando un aggiornamento a uno shadow viene rifiutato

Subscribe_ShadowUpdateDelta(const aThingName: String): Il servizio Device Shadow invia messaggi a questo argomento quando viene rilevata una differenza tra le sezioni reported e desired di un shadow

Subscribe_ShadowUpdateDocuments(const aThingName: String): AWS IoT pubblica un documento di stato su questo topic ogni volta che un aggiornamento al shadow viene eseguito correttamente

Sessioni Persistenti

Una sessione persistente rappresenta una connessione in corso a un broker di messaggi MQTT. Quando un client si connette al broker di messaggi AWS IoT utilizzando una sessione persistente, il broker di messaggi salva tutte le sottoscrizioni effettuate dal client durante la connessione. Quando il client si disconnette, il broker di messaggi memorizza i messaggi QoS 1 non confermati e i nuovi messaggi QoS 1 pubblicati sui topic a cui il client è sottoscritto. Quando il client si riconnette alla sessione persistente, tutte le sottoscrizioni vengono ripristinate e tutti i messaggi memorizzati vengono inviati al client a una frequenza massima di 10 messaggi al secondo.

Si crea una sessione MQTT persistente impostando il parametro **cleanSession** su **False** nell'evento **OnMQTTBeforeConnect**. Se non esiste alcuna sessione per il client, viene creata una nuova sessione persistente. Se esiste già una sessione per il client, viene ripresa.

I dispositivi devono verificare l'attributo **Session** nell'evento **OnMQTTConnect** per determinare se è presente una sessione persistente. Se **Session è True**, è presente una sessione persistente e i messaggi memorizzati vengono consegnati al client. Se **Session è False**, non è presente alcuna sessione persistente e il client deve ri-sottoscrivere ai propri filtri di argomento.

Le sessioni persistenti hanno un periodo di scadenza predefinito di 1 ora. Il periodo di scadenza inizia quando il message broker rileva che un client si disconnette (disconnessione MQTT o timeout). Il periodo di scadenza della sessione persistente può essere aumentato tramite il processo standard di aumento del limite. Se un client non ha ripreso la propria sessione entro il periodo di scadenza, la sessione viene terminata e tutti i messaggi archiviati associati vengono eliminati. Il periodo di scadenza è approssimativo: le sessioni potrebbero essere mantenute fino a 30 minuti in più (ma non meno) rispetto alla durata configurata.

Credenziali Temporanee

AWS IoT Core può funzionare con credenziali temporanee ottenute tramite Identity Pool; esistono 2 tipi di identità:

- **UnAuthenticated:** richiede solo di impostare il tipo di policy nell'IAM
- **Authenticated:** richiede di impostare il tipo di policy in IAM e nelle policy AWS IoT Core

Non autenticato

Se si utilizzano credenziali non autenticate, collegare semplicemente la policy nel ruolo UnAuthenticated creato automaticamente nel menu IAM. Quindi configurare il client impostando Access, Secret Key e Token restituiti dal servizio Cognito.

Di seguito è riportato un codice in .NET per ottenere credenziali non autenticate

```
CognitoAWSCredentials credentials = new CognitoAWSCredentials(
    "us-east-1:cc3c9c48-646d-44ef-bfd5-0c5fb2f0882f", // Identity pool ID
    Amazon.RegionEndpoint.USEast1 // Region
);

var identityPoolId = credentials.GetCredentialsAsync();

AmazonCognitoIdentityClient cognitoClient = new AmazonCognitoIdentityClient(
    credentials, // the anonymous credentials
    Amazon.RegionEndpoint.USEast1 // the Amazon Cognito region
);

GetIdRequest idRequest = new GetIdRequest();
idRequest.AccountId = "222178873557";
idRequest.IdentityPoolId = "us-east-1:cc3c9c48-646d-44ef-bfd5-0c5fb2f0882f";

GetIdResponse idResp = cognitoClient.GetId(idRequest);

string AccessKey = identityPoolId.Result.AccessKey;
string SecretKey = identityPoolId.Result.SecretKey;
string SessionToken = identityPoolId.Result.Token;
```

```
string IdentityId = idResp.IdentityId;
```

Autenticato

Credenziali autenticate, richiede di collegare la policy nell'Authenticated Role creato automaticamente nel menu IAM e di collegare la policy dell'utente nelle policy di AWS IoT Core.

Creare quindi una nuova policy nel menu delle policy di IoT Core e, ogni volta che un nuovo utente si autentica, associare questa policy a tale utente.

È possibile utilizzare il seguente comando di AWS per allegare una policy o creare una funzione lambda.

```
aws iot attach-policy --policy-name PolicyName --target us-east-1:XXXXX-XXXX-XXXX-XX-XX-XXXXXXXXXXXX
```

Provisioning del dispositivo

Il servizio Fleet Provisioning supporta le seguenti operazioni dell'API MQTT:

- **CreateCertificateFromCsr:** Crea un certificato da una richiesta di firma del certificato (CSR). AWS IoT fornisce certificati client firmati dall'autorità di certificazione (CA) radice di Amazon. Il nuovo certificato ha uno stato PENDING_ACTIVATION. Quando si chiama RegisterThing per effettuare il provisioning di un elemento con questo certificato, lo stato del certificato cambia in ACTIVE o INACTIVE come descritto nel template.
- **CreateKeysAndCertificate:** Crea nuove chiavi e un certificato. AWS IoT fornisce certificati client firmati dall'autorità di certificazione radice Amazon (CA). Il nuovo certificato ha uno stato PENDING_ACTIVATION. Quando si chiama RegisterThing per effettuare il provisioning di un dispositivo con questo certificato, lo stato del certificato cambia in ACTIVE o INACTIVE come descritto nel template.
- **RegisterThing:** Effettua il provisioning di un oggetto utilizzando un modello predefinito.

CreateCertificateFromCsr

Utilizzare il metodo CreateCertificateFromCsr passando il CertificateSigningRequest come parametro per creare il certificato. Per ricevere la risposta a questa richiesta, sottoscrivere prima i seguenti metodi: SubscribeCreateCertificateFromCsrResponse e SubscribeCreateCertificateFromCsrError

CreateKeysAndCertificate

Utilizzare il metodo CreateKeysAndCertificate per creare un nuovo certificato e nuove chiavi. Per ricevere la risposta a questa richiesta, sottoscrivere prima i seguenti metodi SubscribeCreateKeysAndCertificateResponse e SubscribeCreateKeysAndCertificateError

RegisterThing

Utilizzi il metodo RegisterThing per registrare un nuovo thing passando come parametri il nome del template e il payload in formato JSON. Per ricevere la risposta a questa richiesta, occorre prima iscriversi ai seguenti metodi: SubscribeRegisterThingResponse e SubscribeRegisterThingError.

IoT Azure MQTT Client

Che cos'è Azure IoT Hub?

IoT Hub è un servizio gestito, ospitato nel cloud, che funge da hub centrale di messaggistica per la comunicazione bidirezionale tra l'applicazione IoT e i dispositivi che gestisce. È possibile utilizzare Azure IoT Hub per costruire soluzioni IoT con comunicazioni affidabili e sicure tra milioni di dispositivi IoT e un backend della soluzione ospitato nel cloud. È possibile connettere praticamente qualsiasi dispositivo a IoT Hub.

IoT Hub supporta le comunicazioni sia dal dispositivo al cloud sia dal cloud al dispositivo. IoT Hub supporta più schemi di messaggistica come la telemetria da dispositivo a cloud, il caricamento di file dai dispositivi e i metodi di richiesta-risposta per controllare i dispositivi dal cloud. Il monitoraggio di IoT Hub aiuta a mantenere l'integrità della soluzione tenendo traccia di eventi quali la creazione di dispositivi, i guasti dei dispositivi e le connessioni dei dispositivi.

Le funzionalità di IoT Hub consentono di creare soluzioni IoT scalabili e complete, come la gestione di apparecchiature industriali nel settore manifatturiero, il monitoraggio di asset di valore nell'ambito sanitario e il controllo dell'utilizzo degli edifici per uffici.

Message broker

IoT Hub fornisce un canale di comunicazione sicuro per i dispositivi per inviare dati. IoT Hub e gli SDK dei dispositivi supportano i seguenti protocolli per la connessione dei dispositivi:

- **MQTT**
- **MQTT over WebSockets**

Più tipi di autenticazione supportano una varietà di funzionalità dei dispositivi:

- Autenticazione basata su token **SAS** per iniziare rapidamente con la propria soluzione IoT.
- Autenticazione tramite **certificato X.509** individuale per un'autenticazione sicura e basata su standard.

Client MQTT

TsgcIoTAzure_MQTT_Client è il componente utilizzato per connettersi ad Azure IoT. Un client può connettersi a un solo dispositivo. Il client si connette utilizzando il protocollo MQTT semplice e si autentica tramite SAS / Certificato client X.509.

Per connettersi ad Azure IoT Hub, il client necessita delle seguenti proprietà:

Azure.IoTHub: nome del server a cui si conatterà il client MQTT.
Azure.Deviceld: nome del dispositivo nell'hub IoT di Azure.

Azure consente più tipi di autenticazione, per impostazione predefinita usa i token SAS.

Autenticazione SAS

SAS.Enabled: abilitare se l'autenticazione utilizza SAS.
SAS.SecretKey: il token SAS del proprio account Azure IoT.
SAS.KeyName: il nome della chiave di accesso condiviso.
SAS.Expiry: impostare il numero di minuti prima della scadenza del token SAS. Il valore predefinito è 1440 (24 ore).

Se si dispone di una stringa di connessione, è possibile leggere automaticamente i valori della stringa di connessione utilizzando il metodo **ReadConnectionString**. Esempio:

```
ReadConnectionString('HostName=yourhub.azure-
devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=Yj7RRPnkSDTv+UCFLgwIP/
FrbDymZv4qVAIoTLHUFR8=');
```

Certificati X509

Utilizzo di **OpenSSL** come IOHandler

Certificate.Enabled: abilitare se l'autenticazione utilizza certificati.

Certificate.CertFile: percorso del certificato client X.509.

Certificate.KeyFile: percorso del file chiave client X.509.

Certificate.Password: se il certificato ha una password, impostarla qui.

Version: Versione TLS, per impostazione predefinita utilizza TLS 1.0

Utilizzo di **SChannel** come IOHandler

Certificate.Enabled: abilitare se l'autenticazione utilizza certificati.

Certificate.CertFile: percorso al certificato PFX (il certificato deve essere prima convertito in PFX). [Ulteriori informazioni](#).

Certificate.Password: se il certificato ha una password, impostarla qui.

Version: Versione TLS, per impostazione predefinita utilizza TLS 1.0

Altre proprietà:

MQTTHeartBeat: se abilitato, tenta di mantenere attiva la connessione MQTT inviando un ping ogni x secondi.

Interval: numero di secondi tra ogni ping.

WatchDog: se abilitato, quando viene rilevata una disconnessione inattesa, tenta di riconnettersi automaticamente al server.

Interval: secondi prima dei tentativi di riconnessione.

Attempts: numero massimo di tentativi di riconnessione; zero significa illimitato.

LogFile: se abilitato, salva i messaggi del socket in un file di log (utile per il debugging). L'accesso al file di log non è thread safe se vi si accede da più thread.

Enabled: se abilitato, ogni messaggio ricevuto e inviato tramite socket verrà salvato su un file.

FileName: percorso completo al nome del file.

L'implementazione MQTT di Azure è basata sulla versione MQTT 3.1.1, ma si discosta dalla specifica nei seguenti modi:

- IoT Hub non supporta i messaggi QoS 2. Se un'app per dispositivo pubblica un messaggio con QoS 2, IoT Hub chiude la connessione di rete.
- IoT Hub non persiste i messaggi Retain. Se un dispositivo invia un messaggio con il flag RETAIN impostato a 1, IoT Hub aggiunge la proprietà dell'applicazione x-opt-retain al messaggio. In questo caso, invece di mantenere il messaggio retain, IoT Hub lo trasmette all'app back-end.
- IoT Hub supporta solo una connessione MQTT attiva per dispositivo. Qualsiasi nuova connessione MQTT per lo stesso ID dispositivo causa la disconnessione della connessione esistente da parte di IoT Hub.

Connessione ad Azure IoT Hub

Prima è necessario accedere al proprio account Azure, registrare un nuovo dispositivo e creare un metodo di autenticazione per questo dispositivo. Una volta completato, è possibile creare un nuovo `TsgcloTAzure_MQTT_Client` e connettersi ad Azure IoT Hub.

Ad esempio:

Dal Dispositivo al Cloud

Quando si inviano informazioni dall'app del dispositivo al back-end della soluzione, IoT Hub espone le seguenti opzioni:

1. **Messaggi device-to-cloud** per telemetria delle serie temporali e avvisi.

È possibile inviare proprietà chiave-valore utilizzando un `TStringList`; è sufficiente riempire il `TStringList` con le proprietà del messaggio desiderate e passarle come argomento.

Se è necessario impostare il **ContentType** e il **ContentEncoding** del messaggio, è necessario aggiungere questi valori alla lista Properties. I nomi di queste proprietà sono definiti da Azure.

Nome	Valore
\$.ct	application/json
\$.ce	utf-8

2. **Proprietà segnalate del device twin** per segnalare informazioni sullo stato del dispositivo, come le capacità disponibili, le condizioni o lo stato dei flussi di lavoro a lunga esecuzione. Ad esempio, configurazione e aggiornamenti software.

Cloud To Device

IoT Hub fornisce tre opzioni alle app dei dispositivi per esporre funzionalità a un'app back-end:

1. **Metodi diretti** per le comunicazioni che richiedono la conferma immediata del risultato. I metodi diretti vengono spesso utilizzati per il controllo interattivo dei dispositivi, ad esempio per accendere un ventilatore.

È possibile rispondere ai metodi pubblici utilizzando il metodo seguente.

2. **Proprietà desired del Twin** per comandi a lunga esecuzione destinati a portare il dispositivo in un determinato stato desiderato. Ad esempio, impostare l'intervallo di invio della telemetria a 30 minuti. È possibile ottenere le proprietà utilizzando il metodo seguente.

3. **Messaggi cloud-to-device** per notifiche unidirezionali all'app del dispositivo. Per ricevere messaggi, deve prima effettuare la sottoscrizione.

I messaggi vengono ricevuti nell'evento **OnMQTTPublish** (payload di testo) o nell'evento **OnMQTTPublishEx** (payload come `TsgcWSMQTTPublishData` con le proprietà Value, Bytes e Stream).

Carica File

L'hub IoT facilita il caricamento di file dai dispositivi connessi fornendo loro URI di firma di accesso condiviso (SAS) o certificati X509.

Se si seleziona SAS, è necessario impostare le seguenti proprietà:

- **Azure.IoTHub:** esempio `youriothub.azure-devices.net`
- **Azure.Deviceld:** esempio: `myDevice`
- **SAS.SecretKey:** esempio: `Yj7RRPnkSDTv+UCFLgwIP/FrbDymZv4qVAIoTLHUFR8=`
- **SAS.KeyName:** esempio: `iothubowner`
- **SAS.Enabled:** il valore deve essere impostato a `true`.

Se si selezionano i certificati X509, è necessario impostare le seguenti proprietà:

- **Certificate.CertFile:** il percorso al certificato PEM.
- **Certificate.KeyFile:** il percorso del file chiave PEM.
- **Certificate.Enabled:** il valore deve essere impostato su `true`.

Usare il metodo **UploadFile** per caricare un file sui server Azure. Se il parametro **Overwrite** è impostato su `true`, sostituirà il file esistente. Se il parametro **Overwrite** è impostato su `false`, caricherà il file solo se non esiste e genererà un errore se esiste (questa è l'opzione predefinita).


```
void UploadFileToAzure()
{
    TOpenDialog oDialog = new TOpenDialog();
    if (oDialog.Execute())
    {
        AzureIoT.UploadFile(oDialog.FileName);
    }
}
```

Device Provisioning Service

Azure IoT consente di registrare i dispositivi da codice utilizzando DPS. Attualmente la libreria supporta la registrazione di un dispositivo passando come parametri lo Scope Id e il Registration Id.

```
TsgcIoTAzure_MQTT_Client oClient = new TsgcIoTAzure_MQTT_Client();
oClient.Certificate.CertFile = "cert.pem";
oClient.Certificate.KeyFile = "key.pem";
oClient.Certificate.Enabled = true;
TsgcIoT_Azure_OperationRegistrationState oResponse = new TsgcIoT_Azure_OperationRegistrationState();
if (oClient.ProvisioningDeviceClient_Register("scope_id", "registration_id", oResponse))
    MessageBox.Show("#Provisioning Register OK: " + oResponse.Status);
else
    MessageBox.Show("#Provisioning Register Error: " + oResponse.Status);
```

Azure IoT Explorer

È possibile utilizzare l'applicazione **Azure IoT Explorer** per interagire con i dispositivi connessi all'IoT Hub. È possibile visualizzare i messaggi di telemetria ricevuti, i dispositivi registrati e altro ancora. L'applicazione è gratuita e può essere scaricata da:

<https://github.com/Azure/azure-iot-explorer/releases>

HTTP

Il protocollo HTTP consente di recuperare risorse dai server, come immagini e documenti HTML. È un protocollo client-server, il che significa che il client richiede dal server le risorse di cui ha bisogno.

Quando un client vuole connettersi a un server, segue i passi successivi:

1. Aprire una nuova connessione TCP
2. Inviare un messaggio al server con i dati richiesti

```
GET / HTTP/1.1
Host: server.com
Accept-Language: en-us
```

3. Leggere la risposta inviata dal server

```
HTTP/1.1 200 OK
Server: Apache
Content-Length: 120
Content-Type: text/html
...
```

Componenti

- **HTTP/2:** HTTP/2 (o h2) è un protocollo binario che introduce push, stream multiplex e controllo dei frame per il web.
- **HTTP/1 Client:** un componente non visivo che eredita dal componente client TIdHTTP.
- **OAuth2:** OAuth2 consente ad applicazioni di terze parti di ricevere accesso limitato a un servizio HTTP.
- **JWT:** JWT consente di creare dati con firma e/o crittografia opzionali, il cui payload contiene JSON che asserisce un certo numero di claim.
- **Amazon SQS:** un servizio di accodamento messaggi completamente gestito per microservizi, sistemi distribuiti e applicazioni serverless.
- **Google Cloud Pub/Sub:** fornisce la messaggistica tra applicazioni ed è progettato per offrire una messaggistica asincrona affidabile, multi-a-molti, tra applicazioni.
- **Google Calendar:** consente di utilizzare l'API Google Calendar V3: ottenere Calendari, eventi, sincronizzarsi con il proprio calendario...
- **Google FCM:** invia notifiche tramite Firebase Cloud Messaging.

HTTP/2

HTTP/2 è un'evoluzione del protocollo HTTP 1.1, che tenta sostanzialmente di essere più efficiente nell'utilizzo delle reti. La semantica è la stessa, quindi è progettato per essere compatibile con i vecchi protocolli.

Limitazioni HTTP 1.1

HTTP 1.1 è limitato all'elaborazione di una richiesta per connessione, pertanto di solito i client utilizzano più di una connessione per richiedere file dai server. Questo crea un problema, poiché quando ci sono troppe connessioni TCP aperte, si instaura una competizione tra i client per l'uso delle risorse del server e le prestazioni calano progressivamente man mano che più client si connettono ai server.

Funzionalità principali

- HTTP/2 è un protocollo binario (ricordare che HTTP 1.1 è un protocollo testuale).
- HTTP/2 funziona su TLS e ALPN.
- È multiplexato (consente di inviare più di una richiesta su una singola connessione TCP).
- Il server può inviare risposte ai client.
- Riduce i tempi di andata e ritorno (Round Trip Times), in modo che i client possano caricarsi più rapidamente.

HTTP/2 introduce altri miglioramenti, maggiori dettagli: [RFC7540](#).

HTTP/2 richiede la nostra versione personalizzata di Indy in quanto necessita del protocollo ALPN.

Componenti

- **TsgcHTTP2Client**: componente client che supporta completamente il protocollo HTTP/2 (codice Pascal 100% sgcWebSockets, senza librerie esterne).
- **TsgcWebSocketHTTPServer**: componente server che supporta completamente il protocollo HTTP/2 (sgcWebSockets 100% codice Pascal, senza librerie esterne). Per impostazione predefinita HTTP/2 è disabilitato; è possibile abilitarlo tramite la proprietà `HTTPOptions` impostando `Enable = true`.
- **TsgcWebSocketServer_HTTPAPI**: componente server che supporta il protocollo HTTP/2 (implementazione Microsoft e richiede Windows 2016+ o Windows 10+).
- Server DataSnap: il server datasnap può supportare anche il protocollo HTTP/2.

API

- **Apple Push Notifications**: invia notifiche visibili all'utente sul dispositivo dell'utente da un provider server.

TsgcHTTP2Client

TsgcHTTP2Client implementa il componente Client HTTP/2 e può connettersi a server HTTP/2.

Introduzione

TsgcHTTP2Client implementa il componente Client HTTP/2 e può connettersi a server HTTP/2. Seguire i passaggi seguenti per configurare questo componente:

1. Creare una nuova istanza del componente **TsgcHTTP2Client**.
2. Invia la richiesta al server ed elabora la risposta utilizzando l'evento OnHTTP2Response. Esempio:

```
TsgcHTTP2Client oClient = new TsgcHTTP2Client();
oClient.OnHTTP2Response += OnHTTP2ResponseEvent;
oClient.Get("https://www.google.com");

void OnHTTP2ResponseEvent(object Sender,
    TsgcHTTP2ConnectionClient Connection, TsgcHTTP2RequestProperty Request,
    TsgcHTTP2ResponseProperty Response)
{
    MessageBox.Show(Response.DataString);
}
```

Riferimento

- Proprietà — 19 proprietà disponibili sul componente.
- Metodi — 19 metodi pubblici.
- Eventi — 13 eventi gestibili.

Guide

- **Richieste**
 - [Metodo richiesta HTTP/2](#)
 - [HTTP/2 Server Push](#)
 - [Download file](#)
 - [Risposte parziali HTTP/2](#)
 - [Intestazioni HTTP/2](#)
- **Connessione**
 - [Chiusura connessione client](#)
 - [Mantenimento attivo della connessione client](#)
 - [Motivo disconnessione HTTP/2](#)
 - [Richieste client in sospeso](#)
- **Autenticazione**
 - [Autenticazione del Client](#)
 - [HTTP/2 e OAuth2](#)
- **Classi**
 - [TsgcHTTP2ConnectionClient](#)
 - [TsgcHTTP2RequestProperty](#)

- [TsgcHTTP2ResponseProperty](#)

TsgcHTTP2Client | Metodo HTTP/2 per le richieste

Il Client HTTP/2 può operare in modalità bloccante e non bloccante; internamente il componente lavora in un thread secondario e le richieste vengono elaborate in modo asincrono, ma è possibile effettuare una richiesta e attendere fino al suo completamento.

Di seguito è riportato un esempio di come il client può richiedere una pagina HTML a un server HTTP/2 e come può funzionare in entrambe le modalità.

Modalità asincrona

Ottenga il seguente URL: <https://www.google.com> e venga notificato quando il client riceve la risposta completa. Dopo aver chiamato il metodo **GETASYNC**, il processo continua e l'evento OnHTTP2Response viene richiamato quando la risposta viene ricevuta.

```
TsgcHTTP2Client oClient = new TsgcHTTP2Client();
oClient.OnHTTP2Response += OnHTTP2ResponseEvent;
oClient.GetAsync("https://www.google.com");
void OnHTTP2ResponseEvent(object Sender, TsgcHTTP2ConnectionClient Connection,
    TsgcHTTP2RequestProperty Request, TsgcHTTP2ResponseProperty Response)
{
    MessageBox.Show(Response.Headers.Text + #13#10 + Response.DataString);
}
```

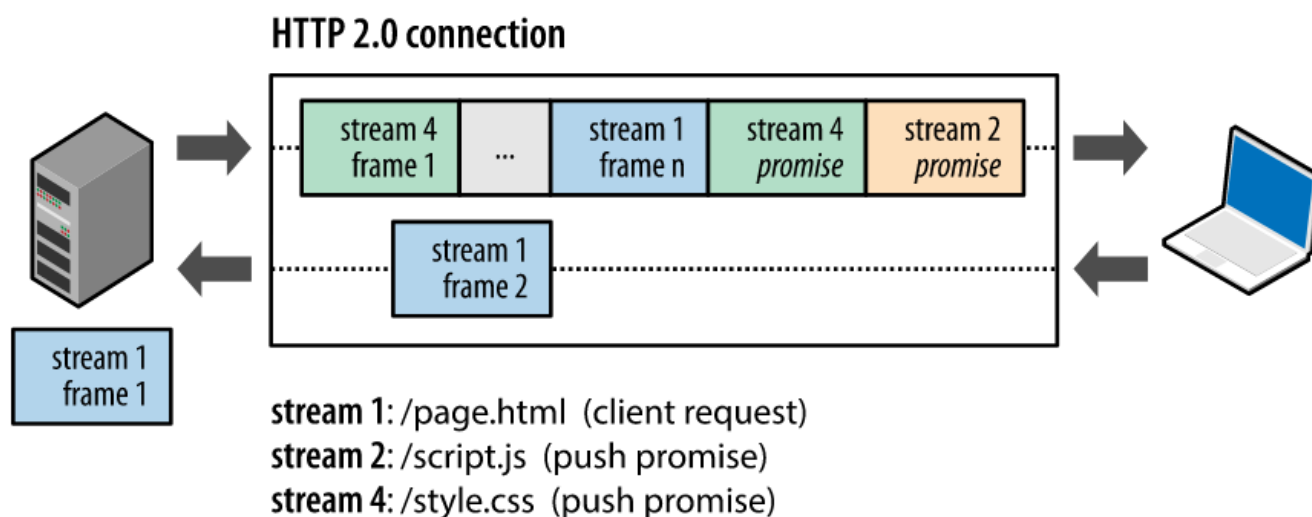
Modalità Bloccante

Ottiene il seguente url: <https://www.google.com> e attende finché il client riceve la risposta completa. Dopo aver chiamato il metodo **GET**, il processo attende finché non viene ricevuta la risposta o non viene raggiunto il time out. Può accedere ai dati Raw Response usando la proprietà Response del client HTTP/2. Qui può accedere a Raw Headers, codice di stato della risposta, Charset e altro.

```
TsgcHTTP2Client oClient = new TsgcHTTP2Client();
MessageBox.Show(oClient.Get("https://www.google.com"));
```

Richieste | HTTP/2 Server Push

Server Push è la capacità del server di inviare più risposte a una singola richiesta del client. Oltre alla risposta alla richiesta originale, il server può inviare risorse aggiuntive al client senza che questi debba richiederle esplicitamente.



Ogni volta che il server invia al client un messaggio PushPromise, viene chiamato l'evento OnHTTP2PushPromise. Quando il client riceve un PushPromise, significa che il server invierà questa risorsa nei pacchetti successivi, quindi il client può accettarla o rifiutarla.

```
TsgcHTTP2Client oClient = new TsgcHTTP2Client();
oClient.OnHTTP2PushPromise += OnHTTP2PushPromiseEvent;
oClient.Get("https://http2.golang.org/serverpush");
...
void OnHTTP2PushPromiseEvent(object Sender, TsgcHTTP2ConnectionClient Connection,
    TsgcHTTP2_Frame_PushPromise PushPromise, ref bool Cancel)
{
    if (PushPromise.URL == "/serverpush/static/godocs.js")
    {
        Cancel = true;
    }
    else
    {
        Cancel = false;
    }
}
```

TsgcHTTP2Client | HTTP/2 Download File

Quando il client richiede un file al server, utilizzare l'evento `OnHTTP2Response` per caricare la risposta dello stream.

Download di file di grandi dimensioni

Quando si scaricano file di grandi dimensioni (centinaia di MB o più), impostare `HTTP2Options.ReadTimeout` su 0 (nessun timeout) per garantire che il trasferimento venga completato senza essere interrotto dal timeout predefinito di 60 secondi:

```
TsgcHTTP2Client oClient = new TsgcHTTP2Client();
oClient.HTTP2Options.ReadTimeout = 0; // no timeout for large files
oClient.Get("https://server/largefile", oStream);
```

```
TsgcHTTP2Client oClient = new TsgcHTTP2Client();
oClient.OnHTTP2Response += OnHTTP2ResponseEvent;
oClient.Get("https://http2.golang.org/file/gopher.png");
...
void OnHTTP2ResponseEvent(object Sender, TsgcHTTP2ConnectionClient Connection,
    TsgcHTTP2RequestProperty Request, TsgcHTTP2ResponseProperty Response)
{
    using (FileStream oStream = new FileStream("file", FileMode.Create, FileAccess.Write))
    {
        Response.Data.CopyTo(oStream);
    }
}
```

TsgcHTTP2Client | Risposte parziali HTTP/2

Di solito quando si invia una richiesta HTTP, il server invia una risposta con il file richiesto; talvolta, invece di inviare una singola risposta, il server può inviare più risposte come uno stream; in questi casi può utilizzare l'evento **OnHTTP2ResponseFragment** per catturare queste risposte e mostrarle all'utente.

Esempio: inviare una richiesta a <https://http2.golang.org/clockstream> e il server invierà una risposta in streaming ogni secondo.

```
TsgcHTTP2Client oClient = new TsgcHTTP2Client();
oClient.OnHTTP2ResponseFragment += OnHTTP2ResponseFragmentEvent;
oClient.Get("https://http2.golang.org/clockstream");
...
void OnHTTP2ResponseFragmentEvent(object Sender, TsgcHTTP2ConnectionClient Connection,
    TsgcHTTP2RequestProperty Request, TsgcHTTP2ResponseFragmentProperty Fragment)
{
    MessageBox.Show(Fragment.DataString);
}
```


TsgcHTTP2Client | HTTP/2 Headers

TsgcHTTP2Client consente di personalizzare le intestazioni inviate al server quando il client si connette

Esempio: se ha bisogno di aggiungere questo Header HTTP "Client: sgcWebSockets"

```
void OnHTTP2BeforeRequest(object Sender, TsgcHTTP2ConnectionClient Connection,  
    ref string Headers)  
{  
    Headers = Headers + Environment.NewLine + "Client: sgcWebSockets";  
}
```

È possibile utilizzare Request.CustomHeaders per aggiungere anche le proprie intestazioni personalizzate.

TsgcHTTP2Client | Chiusura connessione client

La connessione può essere chiusa utilizzando la proprietà `Active` o i metodi `Close/Disconnect`.

Proprietà `Active`

Quando la connessione è attiva e imposta `Active := False`, la connessione verrà chiusa immediatamente senza inviare alcun messaggio al server riguardo alla disconnessione.

Disconnetti

È possibile utilizzare il metodo `Disconnect` (da `TsgcHTTP2Client` o `TsgcHTTP2ConnectionClient`) per disconnettere il socket.

Chiudi

Questo metodo invia un messaggio al server informando che la connessione verrà chiusa e consente di inviare facoltativamente alcune informazioni sul motivo della disconnessione. È un modo pulito per chiudere una connessione HTTP/2.

Il metodo `Close` può essere chiamato dagli oggetti `TsgcHTTP2Client` o `TsgcHTTP2ConnectionClient`.

È possibile inviare i seguenti motivi di errore:

- nessun errore
- errore di protocollo
- errore interno
- errore di controllo del flusso
- timeout impostazioni
- stream chiuso
- errore dimensione frame
- flusso rifiutato
- cancel
- errore di compressione
- errore di connessione
- enhance your calm
- sicurezza inadeguata
- required

TsgcHTTP2Client | Mantenere Attiva la Connessione Client

Una volta che il client si è connesso al server, a volte la connessione può essere interrotta a causa di un segnale scarso, errori di connessione... ci sono 2 proprietà che aiutano a mantenere la connessione attiva.

HeartBeat

La proprietà **HeartBeat** consente di **inviare un Ping** ogni X **secondi** per **mantenere attiva la connessione**. Alcuni server chiudono le connessioni TCP se non vengono scambiati dati tra i peer. HeartBeat risolve questo problema inviando un ping a un intervallo specifico. Di solito questo è sufficiente per mantenere una connessione attiva.

La proprietà HeartBeatType consente di personalizzare il funzionamento di HeartBeat:

1. **hbtAlways**: invia un ping ogni x secondi definiti nell'Intervallo.
2. **hbtOnlyIfNoMsgRcvInterval**: invia un ping ogni x secondi solo se non sono stati ricevuti messaggi durante gli ultimi x secondi definiti nella proprietà Interval.

Esempio: inviare un ping ogni 30 secondi

```
oClient = new TsgcHTTP2Client();
oClient.HeartBeat.Interval = 30;
oClient.HeartBeat.Enabled = true;
oClient.Active = true;
```

WatchDog

Se WatchDog è abilitato, quando il client rileva una disconnessione, WatchDog tenta di riconnettersi ogni X secondi finché la connessione non è nuovamente attiva.

Esempio: riconnessione ogni 10 secondi dopo una disconnessione con tentativi illimitati.

```
oClient = new TsgcHTTP2Client();
oClient.WatchDog.Interval = 10;
oClient.WatchDog.Attempts = 0;
oClient.WatchDog.Enabled = true;
oClient.Active = true;
```

TsgcHTTP2Client | HTTP/2 Reason Disconnection

Il server HTTP/2 può disconnettere un client per diversi motivi; quando il server vuole informare il client del motivo della disconnessione, invia al client un messaggio **GoAway** con informazioni sulla disconnessione.

Utilizzare l'evento `OnHTTP2GoAway` per rilevare il motivo per cui il server si è disconnesso (se il client vuole chiudere una connessione, può usare il metodo `close` per inviare il motivo della chiusura della connessione).

TsgcHTTP2GoAwayProperty L'oggetto contiene le informazioni sulla disconnessione

- **LastStreamId**: è l'ultimo stream elaborato dal server.
- **ErrorCode**: intero che identifica il codice di errore.
- **ErrorDescription**: descrizione dell'errore, uno dei seguenti:
 - nessun errore
 - errore di protocollo
 - errore interno
 - errore di controllo del flusso
 - timeout impostazioni
 - stream chiuso
 - errore dimensione frame
 - flusso rifiutato
 - cancel
 - errore di compressione
 - errore di connessione
 - enhance your calm
 - sicurezza inadeguata
 - required
- **AdditionalDebugData**: stringa opzionale che offre maggiori informazioni sulla disconnessione.

TsgcHTTP2Client | Richieste Client in Sospeso

Quando il client invia più richieste, queste vengono elaborate in un thread secondario; a volte la connessione può essere chiusa per qualsiasi motivo e ci sono ancora richieste in sospeso. Utilizzare l'evento `OnHTTP2PendingRequests` per gestire queste richieste in sospeso. Questo evento viene chiamato quando il client rileva una disconnessione e ci sono ancora richieste in sospeso. Questo evento ha 2 parametri:

1. **Reconnect:** disabilitato per impostazione predefinita, se impostato a `true`, il client si riconnetterà automaticamente.
2. **Clear:** abilitato per impostazione predefinita, se lo imposta su `false`, quando il client si connette di nuovo tenterà di rinviare le richieste in sospeso al server.

TsgcHTTP2Client | Autenticazione Client

Il client HTTP/2 supporta 2 tipi di autenticazione: Autenticazione Basic e Autenticazione OAuth2.

Utilizzare l'evento **OnHTTP2Authorization** per gestire entrambi i tipi di autenticazione.

Autenticazione di base

Se il server restituisce un'intestazione che richiede l'autenticazione Basic, impostare username e password in OnHTTP2Authorization.

```
TsgcHTTP2Client oClient = new TsgcHTTP2Client();
oClient.OnHTTP2Authorization += OnHTTP2AuthorizationEvent;
...
void OnHTTP2AuthorizationEvent(object Sender, TsgcHTTP2ConnectionClient Connection, string AuthType, string AuthC
{
    if (AuthType == "Basic")
    {
        UserName = "user";
        Password = "secret";
    }
}
```

Bearer Token

Se il server restituisce un'intestazione che richiede l'Autenticazione Bearer Token, impostare il token in OnHTTP2Authorization.

```
TsgcHTTP2Client oClient = new TsgcHTTP2Client();
oClient.OnHTTP2Authorization += OnHTTP2AuthorizationEvent;
...
void OnHTTP2AuthorizationEvent(object Sender, TsgcHTTP2ConnectionClient Connection, string AuthType, string AuthC
{
    if (AuthType == "Bearer")
    {
        aToken = "bearer token";
    }
}
```

Valore Bearer da terze parti

Se si conosce già il valore Bearer, perché è stato ottenuto con un altro metodo, è possibile passare il valore Bearer come intestazione HTTP usando le seguenti proprietà della richiesta, da impostare prima di chiamare qualsiasi metodo di richiesta HTTP:

```
TsgcHTTP2Client.Request.BearerAuthentication
= true

TsgcHTTP2Client.Request.BearerToken = "< value of the token >"
```

OAuth2

Legga il seguente articolo se desidera utilizzare il nostro [componente OAuth2 con il client HTTP/2](#).

TsgcHTTP2Client | HTTP/2 e OAuth2

OAuth2 è un metodo di autorizzazione comune utilizzato da diverse aziende come Google. Quando si desidera autenticarsi presso i server Google per utilizzare una qualsiasi delle loro API, di solito è richiesta un'autenticazione tramite OAuth2.

sgcWebSockets supporta OAuth2 sotto il client HTTP/2; è disponibile una proprietà chiamata **Authentication.Token.OAuth** a cui è necessario assegnare un'istanza di [TsgcHTTP_OAuth2](#).

Come connettersi a GMail Google API

Per connettersi alle API Google, è necessario creare un'istanza di TsgcHTTP_OAuth2 e compilare i seguenti dati:

```
TsgcHTTP_OAuth1.AuthorizationServerOptions.AuthURL := 'https://accounts.google.com/o/oauth2/auth';
TsgcHTTP_OAuth1.AuthorizationServerOptions.TokenURL := 'https://accounts.google.com/o/oauth2/token';

TsgcHTTP_OAuth1.LocalServerOptions.IP := '127.0.0.1';
TsgcHTTP_OAuth1.LocalServerOptions.Port := 8080;

TsgcHTTP_OAuth1.OAuth2Options.ClientId := 'your client id';
TsgcHTTP_OAuth1.OAuth2Options.ClientSecret := 'your client secret';
```

Dopo aver configurato il componente client OAuth2, creare una nuova istanza di TsgcHTTP2Client e assegnare il componente OAuth2 al client HTTP/2.

```
TsgcHTTP2Client1.Authentication.Token.OAuth := TsgcHTTP_OAuth1;
```

Infine, effettuare una richiesta per ottenere un elenco dei messaggi dell'account yourname@gmail.com

```
oStream := TStringStream.Create('');
Try
  TsgcHTTP2Client1.Get('https://gmail.googleapis.com/gmail/v1/users/yourname@gmail.com/messages', oStream);
  ShowMessage(oStream.DataString);
Finally
  oStream.Free;
End;
```

TsgcHTTP2ConnectionClient

TsgcHTTP2ConnectionClient è un wrapper delle connessioni client HTTP/2; è possibile accedere a questo oggetto negli eventi Client.

Metodi

- **Ping:** invia un ping al server per mantenere attiva la connessione.
- **Close:** invia un messaggio al server con informazioni sul motivo della disconnessione.
- **Disconnect:** chiude la connessione senza inviare alcun messaggio informativo al server.
- **Metodi HTTP/2:**

GET: Il metodo GET richiede una rappresentazione della risorsa specificata. Le richieste effettuate con GET dovrebbero recuperare dati soltanto.

HEAD: Il metodo HEAD richiede una risposta identica a quella di una richiesta GET, ma senza il corpo della risposta.

POST: Il metodo POST viene utilizzato per inviare un'entità alla risorsa specificata, spesso causando un cambiamento di stato o effetti collaterali sul server.

PUT: Il metodo PUT sostituisce tutte le rappresentazioni correnti della risorsa di destinazione con il payload della richiesta.

DELETE: Il metodo HEAD richiede una risposta identica a quella di una richiesta GET, ma senza il corpo della risposta.

CONNECT: Il metodo CONNECT stabilisce un tunnel verso il server identificato dalla risorsa di destinazione.

OPTIONS: Il metodo OPTIONS viene utilizzato per descrivere le opzioni di comunicazione per la risorsa di destinazione.

TRACE: Il metodo TRACE esegue un test di loopback del messaggio lungo il percorso verso la risorsa di destinazione.

PATCH: Il metodo PATCH viene utilizzato per applicare modifiche parziali a una risorsa.

TsgcHTTP2RequestProperty

Questo oggetto viene ricevuto come argomento nell'evento OnHTTP2Response; consente di conoscere la richiesta originale della risposta inviata dal server.

Proprietà

- **Method:** identifica il metodo HTTP/2 (GET, POST...)
- **URL:** è l'URL richiesto.
- **Request:** contiene i campi della richiesta.

TsgcHTTP2ResponseProperty

Questo oggetto viene ricevuto come argomento nell'evento OnHTTP2Response e consente di conoscere la risposta inviata dal server al client.

Proprietà

- **Headers:** contiene un elenco di intestazioni grezze ricevute dal server.
- **Data:** contiene il corpo grezzo inviato dal server come risposta alla richiesta.
- **DataString:** è la conversione in stringa di Data.
- **DataUTF8:** è la conversione in stringa UTF8 di Data.
- **PushPromise:** se assegnato, contiene l'oggetto PushPromise inviato dal server al client (significa che questo oggetto risposta non è stato richiesto dal client).

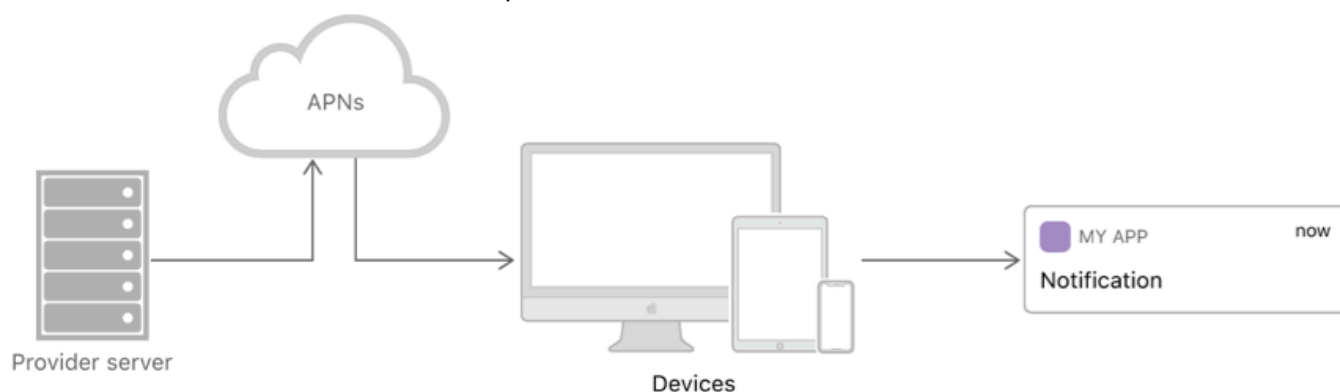
HTTP2 | Notifiche push Apple

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server

Apple consente l'invio di notifiche push ai dispositivi Apple tramite il servizio Apple Push Notification Service (APNs).

Quando si desidera inviare una notifica a un dispositivo, il provider deve inviare una richiesta HTTP/2 POST ad APNs includendo le seguenti informazioni:

- Payload JSON con le informazioni che si desidera inviare.
- Un Token Dispositivo che identifica il dispositivo dell'utente.
- Alcune intestazioni HTTP su come consegnare la notifica.
- Un Certificato SSL o un Token JWT per autenticare la richiesta contro APNs



Requisiti per l'invio di notifiche

Per inviare notifiche al proprio dispositivo utilizzando Rad Studio, è necessario seguire i passaggi successivi

- Registra la tua APP con APNs
- [Genera una notifica remota](#)
- [Invio di richieste di notifica ad APNs](#)
 - [Connessione basata su token ad APNs](#)
 - [Connessione basata su certificato ad APNs](#)

APN | Genera una notifica remota APNs

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/generating_a_remote_notification

Le notifiche Apple utilizzano un payload JSON per inviare l'oggetto notifica. La dimensione massima del payload è 4096 byte.

Esempi di payload JSON

Semplice messaggio di avviso

```
{
  "aps": {
    "alert": "Alert from sgcWebSockets!"
  }
}
```

Avviso con titolo e sottotitolo.

```
{
  "aps" : {
    "alert" : {
      "title" : "Game Request",
      "subtitle" : "Five Card Draw",
      "body" : "Bob wants to play poker",
    },
    "category" : "GAME_INVITATION"
  },
  "gameID" : "12345678"
}
```

APN | Invio di richieste di notifica ad APNs

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server

Inviare il payload di notifica remota e le informazioni sul token del dispositivo al servizio Apple Push Notification (APNs).

Come connettersi ad APNs

È necessario utilizzare il protocollo HTTP/2 e almeno TLS 1.2 o versioni successive per stabilire una connessione corretta tra il proprio Server Provider e uno dei seguenti server:

Server di Sviluppo: <https://api.sandbox.push.apple>

Server di produzione: <https://api.push.apple>

Codice di esempio

Creare una nuova istanza di `TsgcHTTP2Client` e chiamare il metodo `POST` per inviare una notifica ad APNs.

```
TsgcHTTP2Client oHTTP = new TsgcHTTP2Client();  
// ... requires authorization code  
oHTTP.Post("https://api.push.apple/3/device/device_token", "{\"aps\":{\"alert\":\"Alert from sgcWebSockets!\"}}")  
if (oHTTP.Response.Status == 200)  
{  
    MessageBox.Show("Notification Sent Successfully");  
}  
else  
{  
    MessageBox.Show("Notification error");  
}
```

Per inviare notifiche, deve stabilire una relazione di fiducia **basata su token** o **basata su certificato** con APNs utilizzando il protocollo HTTP/2 e TLS 1.2 o successivo.

- [Connessione basata su token a APNs](#)
- [Certificate-Based Connection to APNs](#)

APNs Trusted | Connessione basata su token ad APNs

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/establishing_a_token-based_connection_to_apns

Protegga le comunicazioni con Apple Push Notification service (APNs) utilizzando token di autenticazione senza stato.

Prima è necessario ottenere una **Chiave di Cifratura** e un **ID Chiave** dall'account Apple Developer. Una volta completata con successo la registrazione, si otterrà una stringa di 10 caratteri con l'ID Chiave e una chiave di firma del token di autenticazione come file con estensione .p8.

Deve utilizzare il **JWT Client** di sgcWebSockets per generare un JWT utilizzando **ES256** come algoritmo. Il token non deve essere generato per ogni richiesta HTTP/2, il token non deve essere aggiornato prima di 20 minuti e non oltre i 60 minuti.

Configurare il JWT Client

Configurare il JWT Client con i seguenti valori:

- **JWTOptions.Header.Algorithm:** è l'algoritmo di crittografia utilizzato per cifrare il token. APNs supporta solo l'algoritmo ES256.
- **JWTOptions.Header.kid:** è il Key ID di 10 caratteri ottenuto dal Suo account sviluppatore.
- **JWTOptions.Payload.iss:** il valore che corrisponde al Team ID di 10 caratteri utilizzato per sviluppare le app della propria azienda. Ottenere questo valore dal proprio account sviluppatore.
- **JWTOptions.Payload.iat:** Il tempo di "emissione", il cui valore indica il momento in cui questo token JSON è stato generato. Specificare il valore come numero di secondi dall'Epoch, in UTC. Il valore non deve essere superiore a un'ora rispetto all'orario corrente.
- **JWTOptions.RefreshTokenAfter:** impostare il valore in secondi a 40 minuti (60*40).

L'utilizzo di connessioni basate su Token richiede l'invio di **apns-topic** con il valore del bundle ID/app id dell'app (esempio: com.example.application).

```
TsgcHTTP2Client oHTTP = new TsgcHTTP2Client();
oHTTP.TLSOptions.IOHandler = TwstLSIOHandler.iohOpenSSL;

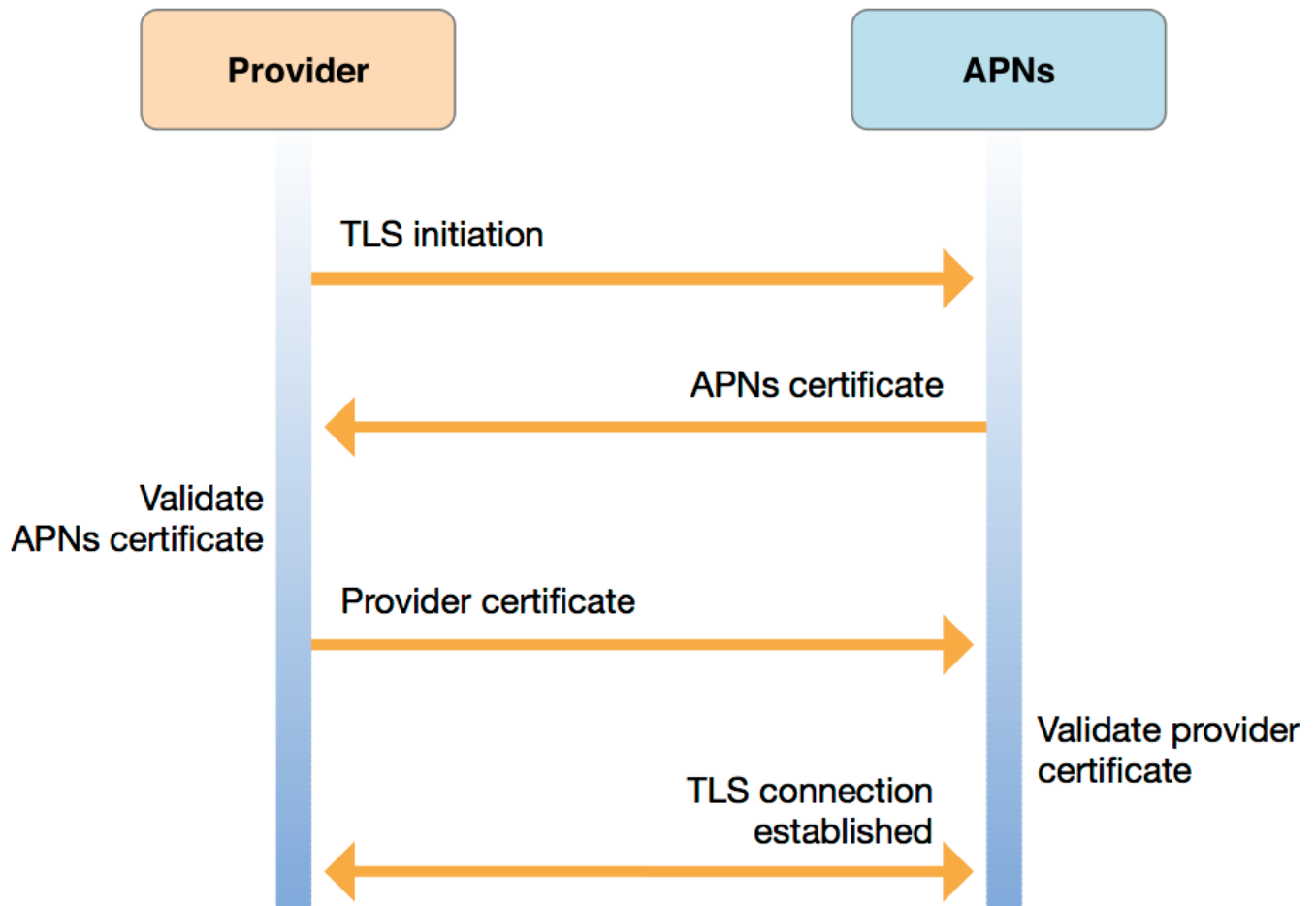
TsgcHTTP_JWT_Client oJWT = new TsgcHTTP_JWT_Client();
oHTTP.Authentication.Token.JWT = oJWT;
oJWT.JWTOptions.Header.alg = TsgcHTTP_JWT_Algorithm.jwtES256;
oJWT.JWTOptions.Header.kid = "apple key id";
oJWT.JWTOptions.Payload.iss = "issuer";
oJWT.JWTOptions.Payload.iat = unix time;
oJWT.JWTOptions.Algorithms.ES.PrivateKey.LoadFromFile("AuthKey_*.p8");
oJWT.JWTOptions.RefreshTokenAfter = 60*40;

oHTTP.Request.CustomHeaders = "apns-topic: com.example.application";
```

Connessione ad APNs basata su certificato

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/establishing_a_certificate-based_connection_to_apns

È possibile proteggere le comunicazioni con Apple Push Notification service (APNs) utilizzando un certificato ottenuto da Apple.



Prima accedere al proprio account sviluppatore e **creare un nuovo certificato** per il servizio Apple Push Notification

Una volta scaricato il certificato, il client HTTP/2 di `sgcWebSockets` consente di utilizzare 2 IOHandler di sicurezza (solo per Windows; per le altre piattaforme è supportato solo `openssl`).

- `OpenSSL`
- `SChannel` (solo per Windows)

OpenSSL

Se si utilizza `OpenSSL`, è necessario distribuire le librerie `OpenSSL` con l'applicazione. Prima di impostare il certificato con `TsgcHTTP2Client`, questo certificato deve essere prima convertito in formato PEM perché `OpenSSL` non consente l'importazione diretta di certificati P12.

Utilizzare i seguenti comandi per convertire un singolo certificato P12 in un certificato in formato PEM e un file di chiave privata

crea file di certificato PEM

```
openssl pkcs12 -in INFILE.p12 -out OUTFILE.crt -nokeys
```

Crea file di chiave privata

```
openssl pkcs12 -in INFILE.p12 -out OUTFILE.key -nodes -nocerts
```

Una volta ottenuto il certificato e la chiave privata in formato PEM, è possibile configurare il [TsgcHTTP2Client](#) come segue.

```
TsgcHTTP2Client oHTTP = new TsgcHTTP2Client();
oHTTP.TLSOptions.IOHandler = TwstLSIOHandler.iohOpenSSL;
oHTTP.TLSOptions.CertFile = "certificate_file.pem";
oHTTP.TLSOptions.KeyFile = "private_key.pem";
oHTTP.TLSOptions.Password = "certificate_password";
oHTTP.TLSOptions.Version = TwstLSVersions.tls1_2;
```

SChannel

Se si utilizza SChannel, non è necessario distribuire librerie e il certificato scaricato da Apple può essere importato direttamente senza necessità di una preventiva conversione in formato PEM.

Impostare la proprietà `UseLegacyCredentials` a `True` quando si utilizza SChannel come IOHandler.

```
TsgcHTTP2Client oHTTP = new TsgcHTTP2Client();
oHTTP.TLSOptions.IOHandler = TwstLSIOHandler.iohSChannel;
oHTTP.TLSOptions.IOHandler.SChannel_Options.UseLegacyCredentials = true;
oHTTP.TLSOptions.CertFile = "certificate_file.p12";
oHTTP.TLSOptions.Password = "certificate_password";
oHTTP.TLSOptions.Version = TwstLSVersions.tls1_2;
```

Errori

Se si riceve l'errore **"missing topic"**, molto probabilmente si sta utilizzando un certificato universale (certificati utilizzabili per notifiche push, voip...) che richiede di impostare il nome dell'argomento con il valore del bundle ID/app id dell'app (esempio: `com.example.application`). È sufficiente impostare l'intestazione `apns-topic` con il valore corretto nella proprietà `Request` del client HTTP/2.

```
oHTTP.Request.CustomHeaders = "apns-topic: com.example.application";
```


HTTP/1

TsgcHTTP1Client è un componente non visuale che eredita dal componente indy **TIdHTTP** e aggiunge alcune nuove proprietà.
Questo componente si trova nell'unità **sgcHTTP**.

TLSOptions

Consente di configurare la modalità di connessione a server sicuri SSL/TLS tramite il protocollo HTTP/1.

ALPNProtocols: elenco dei protocolli ALPN che verranno inviati al server.

RootCertFile: percorso del file del certificato radice.

CertFile: percorso del file del certificato.

KeyFile: percorso del file della chiave del certificato.

Password: se il certificato è protetto da una password, impostarla qui.

VerifyCertificate: se il certificato deve essere verificato, abilita questa proprietà. Utilizzi l'evento **OnSSLVerifyPeer** per personalizzare la verifica SSL.

VerifyDepth: è una proprietà Integer che rappresenta il numero massimo di link consentiti durante la verifica del certificato X.509.

Versione: per impostazione predefinita utilizza TLS 1.0. Se il server richiede una versione TLS superiore, è possibile selezionarla qui.

Proxy: qui è possibile definire se si desidera connettersi tramite un server Proxy; è possibile connettersi ai seguenti server proxy:

pxyHTTP: Server Proxy HTTP.

pxySocks4: Server proxy SOCKS4.

pxySocks4A: Server Proxy SOCKS4A.

pxySocks5: Server Proxy SOCKS5.

IOHandler: selezionare la libreria da utilizzare per connettersi tramite TLS.

iohOpenSSL: utilizza la libreria OpenSSL ed è il predefinito per i componenti Indy. Richiede il deployment delle librerie OpenSSL per win32/win64.

iohSChannel: utilizza Secure Channel, un protocollo di sicurezza implementato da Microsoft per Windows. Non richiede la distribuzione delle librerie OpenSSL. Funziona solo su Windows 32/64 bit.

OpenSSL_Options: configurazione delle librerie openssl.

APIVersion: consente di definire quale API OpenSSL verrà utilizzata.

oslAPI_1_0: utilizza l'API OpenSSL 1.0, l'ultima supportata da Indy

oslAPI_1_1: utilizza OpenSSL API 1.1, richiede la nostra libreria Indy personalizzata e consente l'utilizzo delle librerie OpenSSL 1.1.1 (con supporto TLS 1.3).

oslAPI_3_0: utilizza l'API 3.0 OpenSSL, richiede la nostra libreria Indy personalizzata e consente l'uso delle librerie OpenSSL 3.0.0 (con supporto TLS 1.3).

LibPath: qui è possibile configurare il percorso in cui si trovano le librerie OpenSSL

oslpNone: questa è l'impostazione predefinita. Le librerie OpenSSL devono trovarsi nella stessa cartella del binario o in un percorso noto.

oslpDefaultFolder: imposta automaticamente il percorso OpenSSL in cui dovrebbero trovarsi le librerie per tutte le personalità dell'IDE.

oslpCustomFolder: se questa è l'opzione selezionata, definire il percorso completo nella proprietà LibPathCustom.

LibPathCustom: quando LibPath = oslpCustomFolder definisca qui il percorso completo dove si trovano le librerie OpenSSL.

UnixSymLinks: abilita o disabilita il caricamento dei SymLink sui sistemi Unix (per impostazione predefinita è abilitato, tranne che su OSX64):

oslsSymLinksDefault: per impostazione predefinita, i symlink sono abilitati eccetto su OSX64 (macOS Monterey e versioni successive non riescono a caricare la libreria con una versione).

oslsSymLinksLoadFirst: carica prima i symlink, prima di tentare di caricare le librerie con versione.

oslsSymLinksLoad: carica i symlink dopo aver tentato di caricare le librerie con versione.

oslsSymLinksDontLoad: non caricare i SymLink.

MinVersion: impostare qui la versione minima che il client utilizzerà per connettersi a un server sicuro. Per impostazione predefinita, il valore è **tlsUndefined**, il che significa che la versione minima è la stessa impostata nella proprietà **Version**. Esempio: se si desidera che il client si connetta solo utilizzando TLS 1.2 o TLS 1.3, impostare i seguenti valori.

SSLOptions.Version := tls1_3;
SSLOptions.OpenSSL_Options.MinVersion := tls1_2;
X509Checks: utilizzare questa proprietà per abilitare ulteriori validazioni del certificato X509:
Mode: seleziona quali opzioni verranno validate
oslx509chHostName: verifica il certificato dell'hostname.
oslx509chIPAddress: verifica l'indirizzo IP del certificato.
HostName: impostare il nome host se è diverso dalla richiesta.
IPAddress: impostare l'indirizzo IP se è diverso da quello della richiesta.

SChannel_Options: consente di utilizzare un certificato dall'archivio certificati di Windows.
CertHash: è l'hash del certificato. È possibile trovare l'hash del certificato eseguendo un comando dir in PowerShell.
CipherList: qui è possibile impostare quali cifrari verranno utilizzati (separati da ":"). Esempio: CALG_AES_256:CALG_AES_128
CertStoreName: il nome dell'archivio in cui è memorizzato il certificato. Selezionare uno dei seguenti:
 scsnMY (il valore predefinito)
 scsnCA
 scsnRoot
 scsnTrust
CertStorePath: il percorso dello store in cui è memorizzato il certificato. Selezioni una delle seguenti opzioni:
 scspStoreCurrentUser (predefinito)
 scspStoreLocalMachine

Log

Se la proprietà Log è abilitata, salva i messaggi socket in un file di log specificato, utile per il debug.

LogOptions.FileName: percorso completo del file.

Autenticazione

Consente di autenticarsi utilizzando [OAuth2](#) o [JWT](#).

Richieste asincrone

Per impostazione predefinita, HTTP1Client utilizza richieste bloccanti, quindi dopo aver chiamato un metodo di richiesta HTTP, il client attende la risposta dal server. In alternativa, è possibile utilizzare metodi asincroni per eseguire queste richieste HTTP in un thread secondario, evitando di bloccare il thread in cui viene chiamata la richiesta. Sono implementati i seguenti metodi asincroni:

- **GetAsync**
- **PostAsync**
- **PutAsync**
- **OptionsAsync**
- **DeleteAsync**

Dopo aver chiamato questi metodi, invece di attendere la risposta, il codice continua alla riga successiva e la risposta può essere gestita tramite l'evento **OnAsyncResponse**.

```
void OnAsyncResultEvent(object Sender, TsgcHTTPAsyncRequest aRequest, TIdHTTPResponse aResponse)
```

Se si verifica un errore durante l'elaborazione della richiesta asincrona, l'eccezione verrà sollevata nell'evento **OnAsyncException**.

Esempi

Esegui una richiesta **GET** al server HTTPs utilizzando **TLS 1.2**

```
TsgcHTTP1Client oHTTP = new TsgcHTTP1Client();
oHTTP.TLSOptions.Version = tls1_2;
MessageBox.Show(oHTTP.Get("https://www.google.es"));
```

Richiede un metodo **GET** al server HTTPs usando **openssl 1.1** e **TLS 1.3**

```
TsgcHTTP1Client oHTTP = new TsgcHTTP1Client();
oHTTP.TLSOptions.OpenSSL_Options.APIVersion = oslAPI_1_1;
oHTTP.TLSOptions.Version = tls1_3;
MessageBox.Show(oHTTP.Get("https://www.google.es"));
```

Richiedere un metodo **POST asincrono** e leggere la risposta tramite **OnAsyncResultEvent**.

```
void OnAsyncExceptionEvent(object Sender, TsgcThread aThread, Exception E)
{
    Log(E.Message);
}
void OnAsyncResultEvent(object Sender, TsgcHTTPAsyncRequest aRequest, TIdHTTPResponse aResponse)
{
    if (aResponse.ResponseCode == 200)
        Log("ok", aRequest.Response);
    else
        Log("error", aRequest.Response);
}
TsgcHTTP1Client oHTTP = new TsgcHTTP1Client();
oHTTP.OnAsyncResult += OnAsyncResultEvent;
oHTTP.OnAsyncException += OnAsyncExceptionEvent;
TStringStream oRequest = new TStringStream("body");
TStringStream oResponse = new TStringStream("");
oHTTP.PostAsync("https://localhost/test", oRequest, oResponse);
```

Richiede un metodo GET a un server HTTPs utilizzando **SChannel per Windows**.

```
TsgcHTTP1Client oHTTP = new TsgcHTTP1Client();
oHTTP.TLSOptions.IOHandler = iohSChannel;
oHTTP.TLSOptions.Version = tls1_2;
MessageBox.Show(oHTTP.Get("https://www.google.es"));
```

Richiesta del metodo **SSE** per ottenere eventi dati

Eventi

OnSSEMessage

L'evento viene chiamato quando viene ricevuto un nuovo messaggio SSE.

OnSSLVerifyPeer

Se la verifica del certificato è abilitata, in questo evento è possibile verificare e decidere se accettare il certificato del server.

OnSSLGetHandler

Questo evento viene generato prima della creazione dell'handler SSL. Qui è possibile creare il proprio handler SSL (deve essere ereditato da TIdServerIOHandlerSSLBase o TIdIOHandlerSSLBase) e impostare le proprietà necessarie.

OnSSLAfterCreateHandler

Se non è stato creato un oggetto SSL personalizzato, ne viene creato uno predefinito utilizzando il gestore OpenSSL. È possibile accedere alle proprietà del gestore SSL e modificarle se necessario.

OnAsyncResult

L'evento viene chiamato dopo la richiesta di un metodo asincrono (tramite i metodi GetAsync, PutAsync...). Utilizzare il parametro Response per conoscere il risultato della richiesta.

OnAsyncException

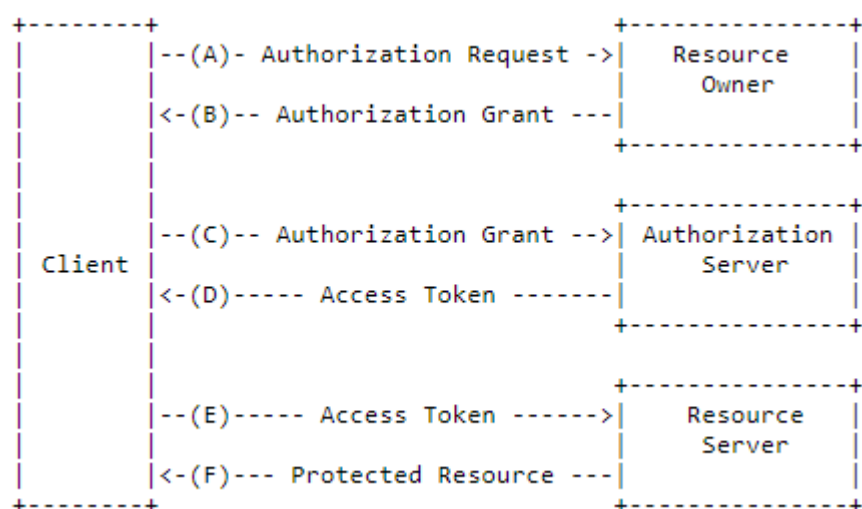
Se si verifica un errore durante l'elaborazione di una richiesta asincrona, questo evento viene chiamato con l'eccezione generata.

HTTP | OAuth2

OAuth2 consente ad applicazioni di terze parti di ricevere accesso limitato a un servizio HTTP, sia per conto di un proprietario della risorsa sia consentendo a un'applicazione di terze parti di ottenere accesso per proprio conto. Grazie a OAuth2, i fornitori di servizi e le applicazioni consumer possono interagire tra loro in modo sicuro.

In OAuth2, ci sono 4 ruoli:

- **Resource Owner:** l'utente.
- **Resource Server:** il server che ospita le risorse protette e ne consente l'accesso in base al token di accesso.
- **Client:** l'applicazione esterna che richiede l'autorizzazione.
- **Authorization Server:** emette l'access token dopo aver autenticato l'utente.



Componenti

- **TsgcHTTP_OAuth2_Client:** è un client con supporto per OAuth2, in grado di connettersi a server OAuth2 per richiedere un'autenticazione come Google, Facebook...
- **TsgcHTTP_OAuth2_Server:** è l'implementazione server del protocollo OAuth2, consente di proteggere le risorse del Server.

I componenti OAuth2 **Server** e **Client** supportano **PKCE** (Proof Key for Code Exchange), che è un'estensione al flusso Authorization Code per prevenire attacchi CSRF e iniezione del codice di autorizzazione (RFC 7636).

TsgcHTTP_OAuth2_Client

Quando un client necessita di un nuovo Access Token, avvia automaticamente un server HTTP per elaborare la risposta del server di autorizzazione.

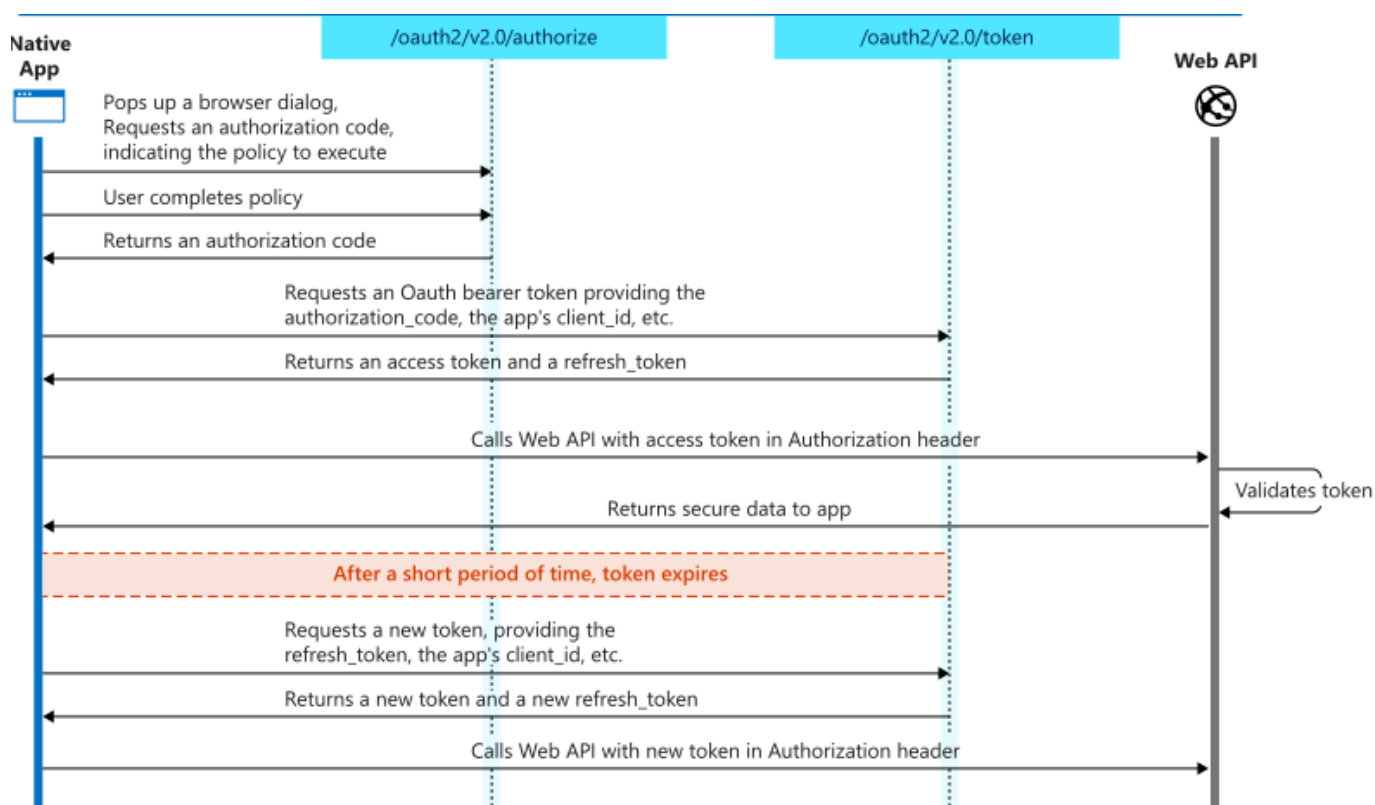
Introduzione

Questo componente Le consente di gestire il flusso tra il client e gli altri ruoli; in pratica, quando imposta `Active := True`, apre un nuovo Web Browser e richiede all'utente di concedere l'autorizzazione; se l'operazione ha esito positivo, il server di autorizzazione invia un token all'applicazione, che viene elaborato e con il quale il client può connettersi al resource server. Questo componente avvia un semplice server HTTP che gestisce le risposte del server di autorizzazione e utilizza un client HTTP per richiedere gli Access Token.

GrantType

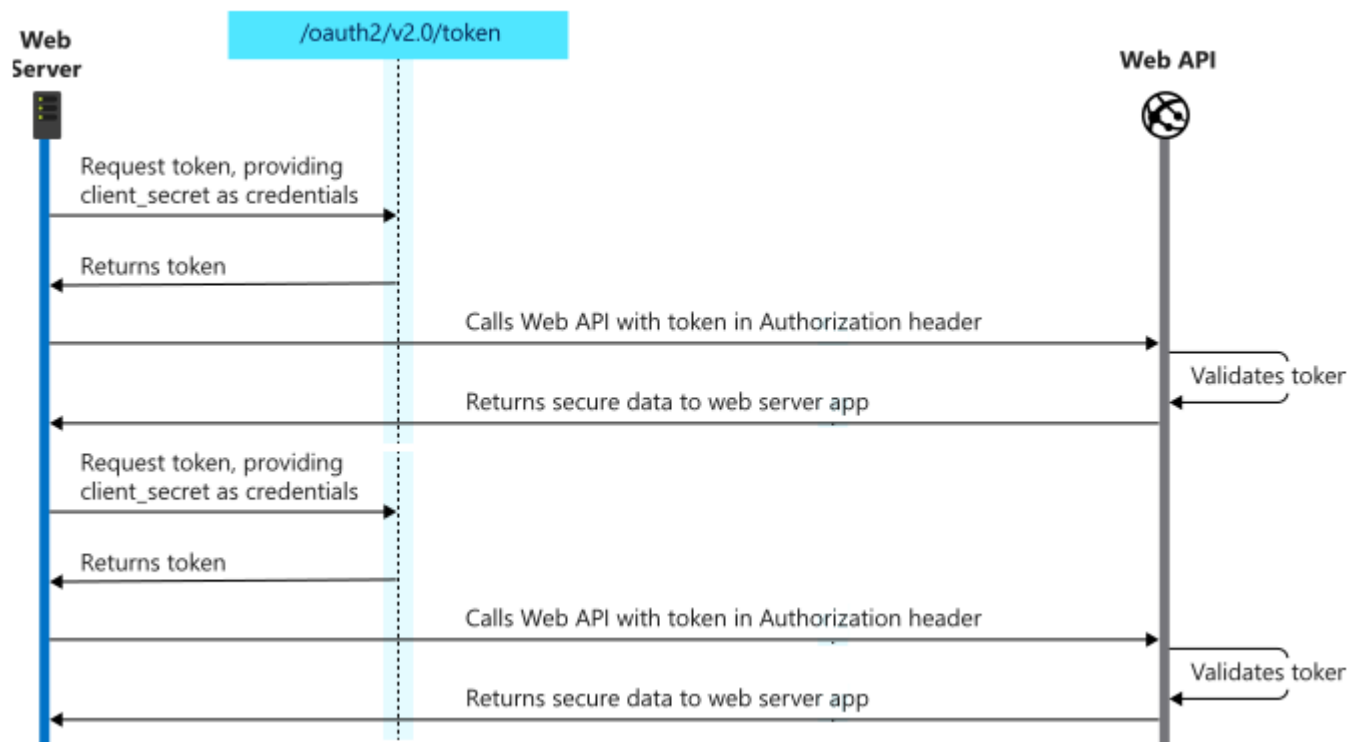
Il client supporta i seguenti tipi di autorizzazione:

auth2Code: Viene utilizzato per eseguire l'autenticazione e l'autorizzazione nella maggior parte dei tipi di applicazioni, incluse le applicazioni a pagina singola, le applicazioni Web e le applicazioni installate nativamente. Il flusso consente alle app di acquisire in modo sicuro `access_token` che possono essere utilizzati per accedere a risorse protette, nonché `refresh_token` per ottenere ulteriori `access_token` e ID token per l'utente connesso.



auth2CodePKCE: è lo stesso flusso di autenticazione di `auth2Code` con PKCE abilitato. PKCE (Proof Key for Code Exchange) è un'estensione di sicurezza per OAuth 2.0, progettata per migliorare la sicurezza dei flussi di autorizzazione per applicazioni native e single-page. Mitiga il rischio di attacchi di intercettazione, specialmente nei client pubblici in cui il codice di autorizzazione potrebbe essere esposto all'intercettazione durante il transito. Di solito questa opzione viene utilizzata nelle app native e mobili.

auth2ClientCredentials: Questo tipo di concessione è comunemente utilizzato per le interazioni server-to-server che devono essere eseguite in background, senza l'interazione immediata con un utente. Questi tipi di applicazioni vengono spesso definiti daemon o account di servizio.



auth2DeviceCode: Device Authorization Grant (RFC 8628) per dispositivi con input limitato come smart TV, console multimediali e dispositivi IoT privi di browser o con capacità di input limitate. Il dispositivo mostra un codice utente e un URI di verifica; l'utente visita l'URI su un dispositivo secondario (telefono o computer) e inserisce il codice per autorizzare.

LocalServerOptions

Quando un client ha bisogno di un nuovo Access Token, avvia automaticamente un server HTTP per elaborare la risposta dal server di autorizzazione. Questo server è trasparente per l'utente e di solito funziona su localhost. Per impostazione predefinita utilizza la porta 8080, ma può essere modificata se necessario.

- **IP:** IP server in ascolto, esempio: 127.0.0.1
- **Port:** per impostazione predefinita 8080. Quando si utilizza GrantType = auth2CodePKCE (per applicazioni desktop e native), è possibile impostare il valore della porta a zero e il server sceglierà una porta casuale.
- **RedirectURL:** (opzionale) consente di personalizzare l'URL di reindirizzamento, esempio: <http://localhost:8080/oauth/>.
- **SSL:** abilitare questa proprietà se il server locale è in esecuzione su una porta sicura (*supportato solo dalle edizioni Professional ed Enterprise).
- **SSLOptions:** consente di personalizzare le proprietà SSL del server (*supportato solo dalle edizioni Professional ed Enterprise).
- **LogOptions:** consente di salvare il log delle Richieste/Risposte ricevute e inviate dal Server HTTP Interno (*Solo per le edizioni Professional ed Enterprise).
 - **Enabled:** impostare su True per abilitare il log su file.
 - **FileName:** impostare il nome del file in cui archiviare il log.

AuthorizationServerOptions

Qui deve impostare l'URL per Authorization e Access Token, di solito questi sono forniti nella specifica dell'API. Scope è un elenco di tutti gli scope richiesti dal client. Esempio:

- **AuthURL:** <https://accounts.google.com/o/oauth2/auth>
- **TokenURL:** <https://accounts.google.com/o/oauth2/token>
- **Scope:** <https://mail.google.com/>
- **RevocationURL:** URL per l'endpoint di revoca del token (obbligatorio per il metodo Revoke). Esempio: <https://accounts.google.com/o/oauth2/revoke>
- **IntrospectionURL:** URL dell'endpoint di introspezione del token (richiesto per il metodo Introspect). Esempio: <https://accounts.google.com/o/oauth2/introspect>

OAuth2Options

ClientId è un campo obbligatorio che informa il server dell'identificazione del client. Consultare la specifica API per sapere come ottenere un ClientId. Lo stesso vale per il client secret.

A volte il server richiede nome utente e password per connettersi tramite autenticazione Basic. Se è questo il caso, è possibile configurarlo nei campi Username/Password. Esempio:

- **GrantType:** Tipo di flusso di autorizzazione
 - **auth2Code:** applicazioni attendibili, come un server web.
 - **auth2CodePKCE:** applicazioni native o mobili non attendibili.
 - **auth2ClientCredentials:** applicazioni automatizzate senza interazione dell'utente.
 - **auth2ResourceOwnerPassword:** consente a un'applicazione di autenticare l'utente gestendo direttamente la sua password
 - **auth2DeviceCode:** Device Authorization Grant (RFC 8628) per dispositivi con capacità di input limitate, privi di browser o con funzionalità di input ridotte.
- **ClientId:** 180803918307-eqjtm20gqfhcs6gjklibrreng022mqqc.apps.googleusercontent.com
- **ClientSecret:** _by1iYYrvVHxC2Z8TbtNEYJN
- **Username:**
- **Password:**

HTTPClientOptions

Qui può personalizzare le Client Options utilizzate per la connessione all'HTTP Server al fine di richiedere un nuovo token.

TLSOptions: se TLS è abilitato, qui è possibile personalizzare alcune proprietà TLS.

ALPNProtocols: elenco dei protocolli ALPN che verranno inviati al server.

RootCertFile: percorso del file del certificato radice.

CertFile: percorso del file del certificato.

KeyFile: percorso del file della chiave del certificato.

Password: se il certificato è protetto da una password, impostarla qui.

VerifyCertificate: se il certificato deve essere verificato, abilitare questa proprietà.

VerifyDepth: è una proprietà Integer che rappresenta il numero massimo di link consentiti durante la verifica del certificato X.509.

Version: per impostazione predefinita utilizza TLS 1.0; se il server richiede una versione TLS superiore, è possibile selezionarla qui.

IOHandler: selezionare la libreria da utilizzare per connettersi tramite TLS.

iohOpenSSL: utilizza la libreria OpenSSL ed è l'impostazione predefinita per i componenti Indy. Richiede il deployment delle librerie openssl per win32/win64.

iohSChannel: utilizza Secure Channel, un protocollo di sicurezza implementato da Microsoft per Windows, non richiede la distribuzione delle librerie openssl. Funziona solo in Windows a 32/64 bit.

OpenSSL_Options: consente di definire quale API OpenSSL verrà utilizzata.

APIVersion: consente di definire quale API OpenSSL verrà utilizzata.

oslAPI_1_0: utilizza l'API OpenSSL 1.0, l'ultima supportata da Indy

oslAPI_1_1: utilizza OpenSSL API 1.1, richiede la nostra libreria Indy personalizzata e consente l'utilizzo delle librerie OpenSSL 1.1.1 (con supporto TLS 1.3).

oslAPI_3_0: utilizza l'API 3.0 OpenSSL, richiede la nostra libreria Indy personalizzata e consente l'uso delle librerie OpenSSL 3.0.0 (con supporto TLS 1.3).

LibPath: qui è possibile configurare la posizione delle librerie openssl

oslpNone: questa è l'impostazione predefinita; le librerie openssl devono trovarsi nella stessa cartella del binario o in un percorso noto.

oslpDefaultFolder: imposta automaticamente il percorso openssl dove le librerie devono essere situate per tutte le personalità IDE.

oslpCustomFolder: se questa è l'opzione selezionata, definire il percorso completo nella proprietà LibPathCustom.

LibPathCustom: quando LibPath = oslpCustomFolder, definire qui il percorso completo dove si trovano le librerie openssl.

UnixSymLinks: abilita o disabilita il caricamento dei SymLink sui sistemi Unix (per impostazione predefinita è abilitato, tranne che su OSX64):

oslsSymLinksDefault: per impostazione predefinita sono abilitati, eccetto su OSX64 (dopo MacOS Monterey si verifica un errore nel tentativo di caricare la libreria senza versione.).

oslsSymLinksLoadFirst: Carica i SymLink e li elabora prima di tentare il caricamento delle librerie di versione.

oslsSymLinksLoad: Carica i SymLink dopo aver tentato di caricare le librerie di versione.

oslsSymLinksDontLoad: non caricare i SymLink.

SChannel_Options: consente di utilizzare un certificato dall'archivio certificati di Windows.

CertHash: è l'hash del certificato. È possibile trovare l'hash del certificato eseguendo un comando dir in PowerShell.

CertStoreName: il nome dello store in cui è memorizzato il certificato. Selezioni uno dei seguenti:

scsnMY (il valore predefinito)

scsnCA

scsnRoot

scsnTrust

CertStorePath: il percorso dello store in cui è memorizzato il certificato. Selezionare uno dei seguenti:

scspStoreCurrentUser (predefinito)

scspStoreLocalMachine

LogOptions: se viene impostato un nome file, verrà salvato un log delle richieste/risposte HTTP del client HTTP

Riferimento

- Proprietà — 6 proprietà disponibili nel componente.
- Metodi — 7 metodi pubblici.
- Eventi — 19 eventi che è possibile gestire.

OAuth2 Client for Web Applications

Quando il Client OAuth2 deve ottenere un Token Autorizzato connettendosi a un'Applicazione Web, il Server Locale (utilizzato per ottenere il token autorizzato) deve essere configurato con i Parametri dell'Applicazione Web. Impostare le seguenti proprietà di **LocalServerOptions**:

- **IP**: indirizzo IP precedentemente configurato nella configurazione OAuth2.
- **Port**: Porta di ascolto configurata in precedenza nella Configurazione OAuth2.
- **SSL**: se il server utilizza una connessione sicura, abilitare questa opzione.

Client OAuth2 per applicazioni desktop

Quando il Client OAuth2 è un'applicazione Desktop o nativa, il Local Server (utilizzato per ottenere il token autorizzato) può essere in ascolto su un indirizzo ip locale (esempio: 127.0.0.1) e la porta può essere scelta casualmente. Pertanto, le **LocalServerOptions** devono essere configurate come segue

- **IP:** 127.0.0.1
- **Port:** impostare su zero e il server sceglierà automaticamente una porta casuale.

Quando si crea un client OAuth2 per applicazioni Desktop o Native, impostare **OAuth2Options.GrantType** su **auth2CodePKCE** per aggiungere un ulteriore livello di sicurezza.

OAuth2 | TsgcHTTP_OAuth2_Client_Google

Questo componente consente di effettuare il login con il proprio Account Google in modo semplice.

Configurazione

Il modulo richiede prima di **configurare l'applicazione OAuth2** nel proprio account Google; una volta configurata, è sufficiente aggiungere alcune righe nell'applicazione per consentire agli utenti di accedere con qualsiasi account Google.

Il **Server locale** utilizzato per leggere la risposta da Google, per impostazione predefinita è in ascolto sull' **indirizzo IP 127.0.0.1** e sulla **porta 0** (porta casuale). È quindi necessario configurare il Callback URL nell'applicazione Google. Naturalmente è possibile modificare l'indirizzo IP e la porta.

Una volta configurata l'applicazione OAuth2 nell'account Google, è sufficiente creare un'istanza di **TsgcHTTP_OAuth2_Client_Google** e chiamare il metodo **Authenticate** passando come parametri il **Client_Id** e il **Client_Secret**. Questo **metodo attende** (per impostazione predefinita fino a 60 secondi) finché l'utente abbia **effettuato correttamente l'accesso**. Restituisce un oggetto con cui è possibile verificare se l'utente si è autenticato o meno, il Nome, l'Id... e altri dati del profilo utente.

Esempio

```
void GoogleSignIn()
{
    TsgcHTTP_OAuth2_Client_Google oClient = new TsgcHTTP_OAuth2_Client_Google();
    TsgcOAuth2_Google_Data oData = oClient.Authenticate("client_id", "client_secret");
    if (oData.Authenticated)
        MessageBox.Show(oData.UserProfile._Name);
}
```

TsgcHTTP_OAuth2_Client_Microsoft

Questo componente consente di accedere con il proprio Account Microsoft in modo semplice.

Configurazione

Il modulo richiede innanzitutto di **configurare la Sua Applicazione OAuth2** nel Suo Account Microsoft; una volta configurata, basta aggiungere un paio di righe alla Sua applicazione per consentire agli utenti di accedere con qualsiasi Account Microsoft.

Il **Server Locale** utilizzato per leggere la risposta da Microsoft, per impostazione predefinita è in ascolto sull' **indirizzo IP 127.0.0.1** e sulla **porta 8080** e utilizza SSL. Pertanto è necessario configurare l'URL di Callback come **https://localhost:8080** (Microsoft consente solo localhost come indirizzo IP locale) nell'applicazione Microsoft. Naturalmente è possibile modificare l'indirizzo IP e la porta.

Una volta configurata l'applicazione OAuth2 nell'Account Microsoft, è sufficiente creare un'istanza di **TsgcHTTP_OAuth2_Client_Microsoft** e chiamare il metodo **Authenticate** passando come parametri il **TenantId**, **Client_Id** e **Client_Secret**. Questo **metodo attende** (per impostazione predefinita fino a 60 secondi) finché l'utente non ha **effettuato l'accesso con successo**. Restituisce un oggetto con cui è possibile verificare se l'utente si è autenticato o meno, il Nome, l'Id... e altri dati del profilo utente.

Esempio

```
void MicrosoftSignIn()
{
    TsgcHTTPComponentClient_OAuth2_Microsoft oClient = new TsgcHTTPComponentClient_OAuth2_Microsoft();
    TsgcOAuth2_Microsoft_Data oData = oClient.Authenticate("tenant_id", "client_id", "client_secret");
    if (oData.Authenticated)
        MessageBox.Show(oData.UserProfile.DisplayName);
}
```

OAuth2 | Authorization Code Grant (RFC 6749)

Panoramica

Il grant Authorization Code è il flusso OAuth2 più comune. È progettato per applicazioni web, desktop e mobili in cui il client può memorizzare in modo sicuro un segreto client. Il flusso prevede il reindirizzamento dell'utente al server di autorizzazione, dove si autentica e concede il permesso. Il server restituisce quindi un codice di autorizzazione a un URI di reindirizzamento e il client scambia quel codice con un token di accesso.

Flow

1. Il client reindirizza l'utente all'**AuthURL** con i parametri `client_id`, `redirect_uri`, `response_type=code`, `scope` e `state`.
2. L'utente si autentica con il server di autorizzazione e concede le autorizzazioni al client.
3. Il server di autorizzazione reindirizza l'utente all'URI di reindirizzamento con un parametro `code`.
4. Il client invia tramite POST il codice di autorizzazione al **TokenURL** insieme a `client_id`, `client_secret` e `redirect_uri` per scambiarlo con un token di accesso.
5. Il server restituisce `access_token`, `token_type`, `expires_in` e opzionalmente un `refresh_token`.

Configurazione

Proprietà	Descrizione
<code>OAuth2Options.GrantType</code>	Impostare su auth2Code .
<code>OAuth2Options.ClientId</code>	L'identificatore del client emesso dal server di autorizzazione.
<code>OAuth2Options.ClientSecret</code>	Il client secret emesso dal server di autorizzazione.
<code>AuthorizationServerOptions.AuthURL</code>	L'URL dell'endpoint di autorizzazione a cui l'utente viene reindirizzato per l'autenticazione.
<code>AuthorizationServerOptions.TokenURL</code>	L'URL dell'endpoint del token in cui il codice di autorizzazione viene scambiato con un token di accesso.
<code>AuthorizationServerOptions.Scope</code>	L'ambito della richiesta di accesso (ad es. openid, profile, email).
<code>LocalServerOptions.IP</code>	L'indirizzo IP del server di redirect locale (ad es., 127.0.0.1).
<code>LocalServerOptions.Port</code>	La porta del server di reindirizzamento locale (ad es., 8080).

Eventi

Evento	Descrizione

<code>OnBeforeAuthorizeCode</code>	Generato prima che il client reindirizzi l'utente all'endpoint di autorizzazione. Consente la personalizzazione della richiesta di autorizzazione.
<code>OnAfterAuthorizeCode</code>	Si genera dopo che il codice di autorizzazione è ricevuto dal server.
<code>OnErrorAuthorizeCode</code>	Generato quando si verifica un errore durante la fase del codice di autorizzazione.
<code>OnBeforeAccessToken</code>	Generato prima che il client scambi il codice di autorizzazione con un token di accesso.
<code>OnAfterAccessToken</code>	Si attiva dopo la ricezione del token di accesso. Utilizzare questo evento per recuperare il valore del token.
<code>OnErrorAccessToken</code>	Generato quando si verifica un errore durante la fase di scambio del token.

Esempio

```
OAuth2.OAuth2Options.GrantType = TsgcOAuth2GrantType.auth2Code;
OAuth2.OAuth2Options.ClientId = "your-client-id";
OAuth2.OAuth2Options.ClientSecret = "your-client-secret";
OAuth2.AuthorizationServerOptions.AuthURL = "https://provider.com/oauth2/authorize";
OAuth2.AuthorizationServerOptions.TokenURL = "https://provider.com/oauth2/token";
OAuth2.AuthorizationServerOptions.Scope.Text = "openid profile";
OAuth2.LocalServerOptions.IP = "127.0.0.1";
OAuth2.LocalServerOptions.Port = 8080;
OAuth2.Start();
```

Aggiornamento Token

Quando un access token scade, può utilizzare il refresh token per ottenerne uno nuovo senza richiedere l'interazione dell'utente. Chiami il metodo `Refresh` con il valore del refresh token:

L'evento `OnAfterAccessToken` si attiverà con il nuovo token di accesso. Se il rinnovo fallisce, si attiverà l'evento `OnErrorAccessToken`.

Quando Utilizzarlo

Utilizzi il grant Authorization Code per:

- Applicazioni web con un backend lato server in grado di memorizzare in modo sicuro il client secret.
- Applicazioni renderizzate lato server (ad es., ASP.NET, PHP, Java).
- Applicazioni desktop in cui il segreto client può essere ragionevolmente protetto.

Per i client pubblici (app native, SPA) dove il client secret non può essere memorizzato in modo sicuro, valuti l'utilizzo di [Authorization Code con PKCE](#).

OAuth2 | Authorization Code con PKCE (RFC 7636)

Panoramica

Il grant Authorization Code con PKCE (Proof Key for Code Exchange) estende il flusso Authorization Code standard con un ulteriore livello di sicurezza. È progettato per applicazioni native, applicazioni mobili e applicazioni a pagina singola (SPA) in cui il segreto client non può essere archiviato in modo sicuro. Anziché affidarsi a un segreto client, il client genera un verificatore di codice crittografico e una sfida che dimostrano che l'entità che scambia il codice di autorizzazione è la stessa entità che ha avviato il flusso.

Come funziona PKCE

1. Il client genera un **code_verifier** casuale (32 byte di dati casuali crittograficamente sicuri, codificati in Base64URL).
 2. Il client calcola il **code_challenge** come hash SHA-256 del **code_verifier**, codificato in Base64URL.
 3. Il client invia **code_challenge** e **code_challenge_method=S256** con la richiesta di autorizzazione.
 4. Dopo l'autorizzazione dell'utente, il client invia il **code_verifier** originale insieme alla richiesta di scambio del token.
 5. Il server di autorizzazione verifica che $\text{SHA256}(\text{code\_verifier}) == \text{code\_challenge}$ prima di emettere il token.
- Questo previene gli attacchi di intercettazione del codice di autorizzazione, poiché un aggressore che intercetta il codice non può scambiarlo senza il **code_verifier** originale.

Configurazione

Proprietà	Descrizione
<code>OAuth2Options.GrantType</code>	Impostare su auth2CodePKCE .
<code>OAuth2Options.ClientId</code>	L'identificatore del client emesso dal server di autorizzazione.
<code>OAuth2Options.ClientSecret</code>	Opzionale. Alcuni provider lo richiedono anche con PKCE; altri no.
<code>AuthorizationServerOptions.AuthURL</code>	L'URL dell'endpoint di autorizzazione a cui l'utente viene reindirizzato per l'autenticazione.
<code>AuthorizationServerOptions.TokenURL</code>	L'URL dell'endpoint del token in cui il codice di autorizzazione viene scambiato con un token di accesso.
<code>AuthorizationServerOptions.Scope</code>	L'ambito della richiesta di accesso.
<code>LocalServerOptions.IP</code>	L'indirizzo IP del server di redirect locale (ad es., 127.0.0.1).
<code>LocalServerOptions.Port</code>	La porta del server di reindirizzamento locale. Impostare a 0 per una porta disponibile casuale.

Esempio

```
OAuth2.OAuth2Options.GrantType = TsgcOAuth2GrantType.auth2CodePKCE;
OAuth2.OAuth2Options.ClientId = "your-client-id";
OAuth2.AuthorizationServerOptions.AuthURL = "https://provider.com/oauth2/authorize";
OAuth2.AuthorizationServerOptions.TokenURL = "https://provider.com/oauth2/token";
OAuth2.AuthorizationServerOptions.Scope.Text = "openid profile";
OAuth2.LocalServerOptions.IP = "127.0.0.1";
OAuth2.LocalServerOptions.Port = 0;
OAuth2.Start();
```

Porta casuale

Quando `LocalServerOptions.Port` è impostato su **0**, il componente seleziona automaticamente una porta disponibile casuale. Questo è consigliato per le applicazioni desktop e mobile perché evita conflitti di porta. La porta selezionata è inclusa nell'URI di reindirizzamento inviato al server di autorizzazione.

Aggiornamento Token

I flussi PKCE tipicamente restituiscono un refresh token. Utilizzi il metodo `Refresh` per ottenere un nuovo access token:

Quando Utilizzarlo

Utilizzare il grant Authorization Code con PKCE per:

- Applicazioni desktop (Windows nativo, macOS, Linux).
- Applicazioni mobile (iOS, Android).
- Applicazioni a pagina singola (SPA) in esecuzione nel browser.
- Qualsiasi client pubblico che non riesce a memorizzare in modo sicuro un segreto client.
- Come alternativa più sicura alla concessione standard tramite [Codice di Autorizzazione](#), anche per i client riservati.

OAuth2 | Client Credentials Grant (RFC 6749)

Panoramica

Il grant Client Credentials viene utilizzato per l'autenticazione machine-to-machine (M2M) quando non è richiesta alcuna interazione da parte dell'utente. Il client si autentica direttamente con il server di autorizzazione utilizzando le proprie credenziali (client_id e client_secret) e riceve un token di accesso. Non esiste un passaggio con codice di autorizzazione né un login utente.

Questo tipo di grant è appropriato quando il client agisce per proprio conto anziché per conto di un utente.

Flow

1. Il client invia tramite POST client_id, client_secret, grant_type=client_credentials e scope all'endpoint del token.
2. Il server di autorizzazione valida le credenziali del client.
3. Il server restituisce access_token, token_type e expires_in.

Nota che i token di aggiornamento in genere non vengono emessi con questo tipo di concessione, poiché il client può semplicemente richiedere un nuovo token di accesso utilizzando le proprie credenziali.

Configurazione

Proprietà	Descrizione
<code>OAuth2Options.GrantType</code>	Impostare su auth2ClientCredentials .
<code>OAuth2Options.ClientId</code>	L'identificatore del client emesso dal server di autorizzazione.
<code>OAuth2Options.ClientSecret</code>	Il client secret emesso dal server di autorizzazione.
<code>AuthorizationServerOptions.TokenURL</code>	L'URL dell'endpoint token. Non è necessario un AuthURL poiché non è previsto alcun passaggio di autorizzazione utente.
<code>AuthorizationServerOptions.Scope</code>	L'ambito della richiesta di accesso.

Esempio

```
OAuth2.OAuth2Options.GrantType = TsgcOAuth2GrantType.auth2ClientCredentials;  
OAuth2.OAuth2Options.ClientId = "your-client-id";  
OAuth2.OAuth2Options.ClientSecret = "your-client-secret";  
OAuth2.AuthorizationServerOptions.TokenURL = "https://provider.com/oauth2/token";  
OAuth2.AuthorizationServerOptions.Scope.Text = "api.read api.write";  
OAuth2.Start();
```

Quando Utilizzarlo

Utilizzare il grant Client Credentials per:

- Servizi backend e daemon che vengono eseguiti senza interazione dell'utente.

- Comunicazione API da server a server.
- Job pianificati, processi batch e attività cron.
- Autenticazione da microservizio a microservizio.
- Qualsiasi scenario in cui l'applicazione agisce per proprio conto, non per conto di un utente.

OAuth2 | Resource Owner Password Credentials Grant (RFC 6749)

Panoramica

La concessione Resource Owner Password Credentials (ROPC) consente al client di raccogliere direttamente nome utente e password dell'utente e scambiarli con un access token. Le credenziali dell'utente vengono inviate all'endpoint del token e il server restituisce un access token se sono valide.

Questo tipo di grant dovrebbe essere utilizzato solo quando vi è un elevato grado di fiducia tra il proprietario della risorsa e il client, ad esempio quando il client è un'applicazione first-party sviluppata dalla stessa organizzazione che gestisce il server di autorizzazione.

Avviso di sicurezza

Questo tipo di concessione è considerato meno sicuro rispetto ad altri flussi OAuth2. Espone le credenziali dell'utente direttamente all'applicazione client, violando il principio dell'autorizzazione delegata. La specifica OAuth 2.1 ha deprecato questo tipo di concessione. Utilizzare [Authorization Code con PKCE](#) ogni volta che è possibile.

Flow

1. L'utente fornisce nome utente e password direttamente all'applicazione client.
2. Il client invia tramite POST username, password, client_id, client_secret, grant_type=password e scope all'endpoint del token.
3. Il server di autorizzazione convalida le credenziali.
4. Il server restituisce access_token, token_type, expires_in e opzionalmente un refresh_token.

Configurazione

Proprietà	Descrizione
<code>OAuth2Options.GrantType</code>	Impostare su auth2ResourceOwnerPassword .
<code>OAuth2Options.ClientId</code>	L'identificatore del client emesso dal server di autorizzazione.
<code>OAuth2Options.ClientSecret</code>	Il client secret emesso dal server di autorizzazione.
<code>OAuth2Options.UserName</code>	Il nome utente del proprietario della risorsa.
<code>OAuth2Options.Password</code>	La password del resource owner.
<code>AuthorizationServerOptions.TokenURL</code>	L'URL dell'endpoint token. Non è necessario un AuthURL poiché le credenziali utente vengono fornite direttamente.
<code>AuthorizationServerOptions.Scope</code>	L'ambito della richiesta di accesso.

Esempio

```
OAuth2.OAuth2Options.GrantType = TsgcOAuth2GrantType.auth2ResourceOwnerPassword;  
OAuth2.OAuth2Options.ClientId = "your-client-id";  
OAuth2.OAuth2Options.ClientSecret = "your-client-secret";  
OAuth2.OAuth2Options.UserName = "user@example.com";  
OAuth2.OAuth2Options.Password = "user-password";  
OAuth2.AuthorizationServerOptions.TokenURL = "https://provider.com/oauth2/token";  
OAuth2.AuthorizationServerOptions.Scope.Text = "openid profile";  
OAuth2.Start();
```

Aggiornamento Token

Se il server restituisce un token di aggiornamento, è possibile utilizzarlo per ottenere un nuovo token di accesso senza reinserire le credenziali dell'utente:

Quando Utilizzarlo

Utilizzare la concessione Resource Owner Password Credentials solo per:

- Applicazioni legacy che non possono essere migrate a un flusso più sicuro.
- Applicazioni proprietarie in cui si controlla sia il client che il server di autorizzazione.
- Scenari di migrazione in cui si sta passando dall'autenticazione diretta con nome utente/password a OAuth2.

Per tutti gli altri casi, utilizzare [Codice di Autorizzazione con PKCE](#) o [Codice di Autorizzazione](#).

OAuth2 | Device Authorization Grant (RFC 8628)

Panoramica

Il grant Device Authorization (noto anche come flusso Device Code) è progettato per dispositivi con input limitato che non possono facilmente visualizzare un browser o accettare input da tastiera. Esempi includono smart TV, dispositivi IoT, media player, strumenti CLI e chioschi. Il dispositivo mostra un breve codice che l'utente inserisce su un altro dispositivo (come un telefono o un computer) per completare l'autorizzazione.

Flow

1. Il client invia via POST `client_id` e `scope` al **DeviceAuthorizationURL**.
2. Il server restituisce `device_code`, `user_code`, `verification_uri`, `expires_in` e `interval`.
3. Il client genera l'evento **OnDeviceCode** con il codice utente e l'URI di verifica. L'applicazione mostra questi all'utente (ad esempio, "Andare su `https://provider.com/device` e inserire il codice: ABCD-1234").
4. L'utente accede all'URI di verifica su un altro dispositivo e inserisce il codice utente.
5. L'utente si autentica e concede l'autorizzazione sul dispositivo secondario.
6. Nel frattempo, il client esegue automaticamente il polling dell'endpoint token ogni `interval` secondi utilizzando il codice dispositivo.
7. Quando l'utente completa l'autorizzazione, il polling successivo restituisce l'`access_token`.
8. Se il device code scade prima che l'utente autorizzi, si attiva l'evento **OnDeviceCodeExpired**.

Configurazione

Proprietà	Descrizione
<code>OAuth2Options.GrantType</code>	Impostare su auth2DeviceCode .
<code>OAuth2Options.ClientId</code>	L'identificatore del client emesso dal server di autorizzazione.
<code>OAuth2Options.ClientSecret</code>	Facoltativo. Alcuni provider lo richiedono, altri no.
<code>AuthorizationServerOptions.TokenURL</code>	L'URL dell'endpoint token utilizzato per il polling.
<code>AuthorizationServerOptions.DeviceAuthorizationURL</code>	L'endpoint di autorizzazione del dispositivo dove il client richiede un device code.
<code>AuthorizationServerOptions.Scope</code>	L'ambito della richiesta di accesso.

Eventi

Evento	Descrizione
<code>OnDeviceCode</code>	Attivato quando il codice dispositivo e il codice utente vengono ricevuti dal server. Mostrare il codice utente e l'URI di verifica all'utente.

<code>OnDeviceCodeExpired</code>	Generato quando il codice del dispositivo scade prima che l'utente completi l'autorizzazione. L'applicazione dovrebbe invitare l'utente a riavviare il flusso.
<code>OnAfterAccessToken</code>	Viene generato quando l'utente completa l'autorizzazione e viene ricevuto l'access token.
<code>OnErrorAccessToken</code>	Si attiva quando si verifica un errore non recuperabile durante il flusso del dispositivo.

Esempio

Gestione dell'evento Device Code

```
OAuth2.OAuth2Options.GrantType = TsgcOAuth2GrantType.auth2DeviceCode;
OAuth2.OAuth2Options.ClientId = "your-client-id";
OAuth2.AuthorizationServerOptions.TokenURL = "https://provider.com/oauth2/token";
OAuth2.AuthorizationServerOptions.DeviceAuthorizationURL = "https://provider.com/oauth2/device/code";
OAuth2.AuthorizationServerOptions.Scope.Text = "openid profile";
OAuth2.OnDeviceCode += (sender, userCode, verificationUri) =>
{
    Console.WriteLine("Go to " + verificationUri + " and enter code: " + userCode);
};
OAuth2.Start();
```

Comportamento di polling

Dopo aver chiamato `start`, il componente interroga automaticamente l'endpoint dei token all'intervallo specificato dal server (tipicamente 5 secondi). Se il server risponde con un errore `slow_down`, il componente aumenta l'intervallo di polling di 5 secondi. Il polling continua finché non si verifica una delle seguenti condizioni:

- L'utente completa l'autorizzazione e viene restituito il token.
- Il codice dispositivo scade (viene attivato `OnDeviceCodeExpired`).
- Si verifica un errore irreversibile (si attiva `OnErrorAccessToken`).

Quando Utilizzarlo

Utilizzare la concessione Autorizzazione Dispositivo per:

- Smart TV e dispositivi di streaming.
- Strumenti dell'interfaccia a riga di comando (CLI).
- Dispositivi IoT con capacità di input limitata.
- Chioschi e segnaletica digitale.
- Qualsiasi dispositivo in cui digitare un URL completo o le credenziali non è pratico.

OAuth2 | DPoP - Demonstrating Proof of Possession (RFC 9449)

Panoramica

DPoP (Demonstrating Proof of Possession) è un meccanismo di sicurezza che lega i token di accesso a un client specifico richiedendo la prova di possesso di una chiave privata. A differenza dei bearer token, che possono essere utilizzati da chiunque li ottenga, i token vincolati a DPoP sono inutilizzabili senza la corrispondente chiave privata. Ciò fornisce protezione contro il furto di token e gli attacchi di replay.

DPoP non è un tipo di grant in sé, ma un miglioramento della sicurezza che può essere utilizzato con qualsiasi tipo di grant OAuth2 (Authorization Code, Client Credentials, ecc.).

Come Funziona

1. Il client genera una coppia di chiavi asimmetriche (ES256 o RS256).
2. Il client crea un **JWT di prova DPoP** con la seguente struttura:
 - Intestazione: typ: dpop+jwt, alg: ES256, jwk: {public key}
 - Payload: htm (metodo HTTP), htu (URI HTTP), iat (emesso a), jti (identificatore univoco)
3. Il client invia la prova DPoP come intestazione HTTP (DPoP: <proof>) con la richiesta di token.
4. Il server di autorizzazione lega il token emesso all'impronta digitale JWK della chiave pubblica del client.
5. Il server restituisce token_type: DPoP invece di token_type: Bearer.
6. Per le successive chiamate API, il client invia sia l'intestazione Authorization: DPoP <token> sia una nuova intestazione DPoP: <proof> specifica per quella richiesta.

Generazione della chiave

È possibile generare una coppia di chiavi ES256 direttamente in Delphi utilizzando il metodo **GenerateDPoPKeyPair**. Questo utilizza OpenSSL internamente per creare la coppia di chiavi e imposta automaticamente sia PrivateKey che PublicKeyJWK:

```
OAuth2.DPoPOptions.Enabled = true;
OAuth2.DPoPOptions.Algorithm = TsgcHTTPOAuth2_DPoP_Algorithm.dpopES256;
OAuth2.GenerateDPoPKeyPair(); // Generates PrivateKey + PublicKeyJWK automatically
OAuth2.Start();
```

PublicKeyToJWK — Convertire qualsiasi chiave o certificato in JWK

Se si dispone già di una chiave pubblica, di una chiave privata o di un certificato X.509 (formato PEM), è possibile convertirlo in JWK utilizzando la funzione di classe **PublicKeyToJWK**. Accetta uno qualsiasi di questi input e restituisce il JSON JWK corretto:

Input	PEM Header
Chiave pubblica EC o RSA	-----BEGIN PUBLIC KEY-----
Chiave privata EC	-----BEGIN EC PRIVATE KEY-----
Chiave privata RSA	-----BEGIN RSA PRIVATE KEY-----
Certificato X.509	-----BEGIN CERTIFICATE-----

Da un file di chiave PEM

```
// From a public key file
OAuth2.DPoPOptions.PublicKeyJWK =
    OAuth2.PublicKeyToJWK(
        File.ReadAllText("public_key.pem"));

// From a private key file (extracts the public part)
OAuth2.DPoPOptions.PublicKeyJWK =
    OAuth2.PublicKeyToJWK(
        File.ReadAllText("private_key.pem"));
```

Da un certificato smart card

Le smart card e gli HSM proteggono la chiave privata — non può essere letta o esportata. Tuttavia, il **certificato** (che contiene la chiave pubblica) è sempre esportabile. Esportare il certificato come PEM e passarlo a `PublicKeyToJWK`:

```
// Export the certificate from the smart card
string certPEM = mySmartCard.ExportCertificatePEM();

// Convert the certificate to JWK
OAuth2.DPoPOptions.PublicKeyJWK =
    OAuth2.PublicKeyToJWK(certPEM);

// No PrivateKey needed - use OnDPoPSign for signing via the card
OAuth2.DPoPOptions.Enabled = true;
OAuth2.OnDPoPSign += MyDPoPSignHandler;
```

Questo approccio funziona con qualsiasi card o HSM in grado di esportare il proprio certificato: token PKCS#11, smart card Windows CNG, chiavi di sicurezza USB (YubiKey, Feitian) e HSM cloud (Azure Key Vault, AWS Cloud-HSM).

Utilizzo degli strumenti a riga di comando OpenSSL

È anche possibile generare una coppia di chiavi esternamente:

DPoP Options

Proprietà	Descrizione
<code>DPoPOptions.Enabled</code>	Impostare su True per abilitare la generazione della prova DPoP.
<code>DPoPOptions.Algorithm</code>	L'algoritmo di firma. Valori supportati: ES256 (consigliato), RS256 .
<code>DPoPOptions.PrivateKey</code>	La chiave privata codificata in PEM utilizzata per firmare il JWT di prova DPoP.
<code>DPoPOptions.PublicKeyJWK</code>	La chiave pubblica in formato JWK (JSON Web Key), inclusa nell'header della prova DPoP.

Metodi

Metodo	Descrizione
<code>GenerateDPoPKeyPair</code>	Genera una coppia di chiavi ES256 utilizzando OpenSSL e imposta automaticamente sia <code>DPoPOptions.PrivateKey</code> (PEM) che <code>DPoPOptions.PublicKeyJWK</code> (JSON).

<code>GetDPoPPProof(Method, URL, AccessToken)</code>	Genera un JWT di prova DPoP per il metodo HTTP, l'URL e il token di accesso specificati. Restituisce la stringa JWT firmata. La prova include il claim <code>ath</code> (hash del token di accesso) per associare la prova al token.
<code>GetDPoPJWKThumbprint</code>	Restituisce il thumbprint JWK (SHA-256) RFC 7638 della chiave pubblica, utilizzato dal server per associare il token.

Gestione del Nonce

Alcuni server di autorizzazione richiedono un nonce fornito dal server nella prova DPoP. Quando il server risponde con un'intestazione `DPoP-Nonce` e uno stato 400 o 401, il componente automaticamente:

1. Estrae il nonce dall'intestazione di risposta `DPoP-Nonce`.
2. Include il nonce nel claim `nonce` del prossimo JWT di prova DPoP.
3. Ripete la richiesta con la prova aggiornata.

Firma personalizzata con OnDPoPSign (HSM / Smart Card)

In alcuni ambienti, la chiave privata è memorizzata su un modulo di sicurezza hardware (HSM) o una smart card (ad esempio tramite PKCS#11 o Windows CNG) e **non può essere estratta**. L'evento `OnDPoPSign` consente di eseguire la firma esternamente mentre il componente gestisce tutto il resto (struttura della proof, nonce, intestazioni). Quando `OnDPoPSign` è assegnato, il componente lo chiama prima di tentare la firma interna. Se si imposta `Handled := True` e si fornisce la `Signature`, il componente utilizza la firma fornita e salta la firma interna. La proprietà `DPoPOptions.PrivateKey` non è necessaria in questo caso.

Firma dell'evento

Parametro	Direzione	Descrizione
<code>SigningInput</code>	in	I dati da firmare: <code>Base64URL(header) + "." + Base64URL(payload)</code> . Questo è l'input di firma JWS standard.
<code>Algorithm</code>	in	Il nome dell'algoritmo: <code>"ES256"</code> o <code>"RS256"</code> .
<code>Signature</code>	out	Impostare questo valore con i byte della firma codificati in Base64URL prodotti dal dispositivo di firma.
<code>Handled</code>	out	Impostare su <code>True</code> per utilizzare la propria firma personalizzata. Se <code>False</code> , il componente utilizza la firma interna tramite <code>PrivateKey</code> .

Esempio Smart Card

Il seguente esempio mostra come utilizzare DPoP con una smartcard. La chiave privata non abbandona mai la carta — solo l'hash viene inviato alla carta per la firma.

```
// Configure DPoP - no PrivateKey needed when using OnDPoPSign
OAuth2.DPoPOptions.Enabled = true;
OAuth2.DPoPOptions.Algorithm = TsgcHTTPOAuth2_DPoP_Algorithm.dpopES256;
OAuth2.DPoPOptions.PublicKeyJWK =
    "{\"kty\":\"EC\",\"crv\":\"P-256\",\"x\":\"...\",\"y\":\"...\"}";
OAuth2.OnDPoPSign += OnDPoPSignHandler;
OAuth2.Start();

private void OnDPoPSignHandler(TsgcHTTPOAuth2_Client Sender,
    string SigningInput, string Algorithm,
    ref string Signature, ref bool Handled)
{
    // Hash and sign with smart card
    byte[] hash = SHA256.Create().ComputeHash(Encoding.UTF8.GetBytes(SigningInput));
```

```
byte[] sig = mySmartCard.SignHash(hash);
Signature = Base64UrlEncode(sig);
Handled = true;
}
```

Questo approccio funziona con qualsiasi backend di firma: token PKCS#11, Windows CNG (CryptoAPI Next Generation), Azure Key Vault, AWS CloudHSM o qualsiasi hardware personalizzato in grado di produrre firme ECDSA/RSA.

Esempio

```
// Configure the grant type
OAuth2.OAuth2Options.GrantType = TsgcOAuth2GrantType.auth2CodePKCE;
OAuth2.OAuth2Options.ClientId = "your-client-id";
OAuth2.AuthorizationServerOptions.AuthURL = "https://provider.com/oauth2/authorize";
OAuth2.AuthorizationServerOptions.TokenURL = "https://provider.com/oauth2/token";
OAuth2.AuthorizationServerOptions.Scope.Text = "openid profile";
// Enable DPoP
OAuth2.DPoPOptions.Enabled = true;
OAuth2.DPoPOptions.Algorithm = "ES256";
OAuth2.DPoPOptions.PrivateKey = File.ReadAllText("dpop_private.pem");
OAuth2.DPoPOptions.PublicKeyJWK = "{\"kty\":\"EC\",\"crv\":\"P-256\",\"x\":\"...\",\"y\":\"...\"}";
OAuth2.Start();
// Generate DPoP proof for API calls
string proof = OAuth2.GetDPoPPProof("GET", "https://api.provider.com/resource");
// Use: Authorization: DPoP <access_token>
// Use: DPoP: <proof>
```

TsgcHTTP_OAuth2_Server

Questo componente fornisce l'implementazione del protocollo OAuth2 nei componenti lato server.

Introduzione

Questo componente fornisce l'implementazione del protocollo OAuth2 nei componenti lato server. I componenti server dispongono di una proprietà denominata `Authorization.OAuth.OAuth2` dove è possibile assegnare un'istanza di `TsgcHTTP_OAuth2_Server`; se l'autenticazione è abilitata e la proprietà `OAuth2` è collegata al componente `Server OAuth2`, le richieste WebSocket e HTTP richiedono un Bearer Token per essere elaborate, altrimenti la connessione verrà chiusa automaticamente.

```
OAuth2 = new TsgcHTTP_OAuth2_Server();
Server.Authentication.Enabled = true;
Server.Authentication.OAuth.OAuth2 = OAuth2;
```

Il server supporta i seguenti tipi di autorizzazione:

- **auth2Code:** Viene utilizzato per eseguire l'autenticazione e l'autorizzazione nella maggior parte dei tipi di applicazioni, incluse le applicazioni a pagina singola, le applicazioni Web e le applicazioni installate nativamente. Il flusso consente alle app di acquisire in modo sicuro `access_token` che possono essere utilizzati per accedere a risorse protette, nonché `refresh token` per ottenere ulteriori `access_token` e ID token per l'utente connesso.
- **auth2ClientCredentials:** Questo tipo di grant è comunemente utilizzato per interazioni server-to-server che devono essere eseguite in background, senza interazione immediata con un utente. Questi tipi di applicazioni sono spesso indicati come daemon o account di servizio.
- **password** (Resource Owner Password Credentials): Consente a un'applicazione di far accedere l'utente gestendo direttamente le sue credenziali. Il client invia il nome utente e la password dell'utente all'endpoint del token.
- **urn:ietf:params:oauth:grant-type:device_code** (Device Code): Device Authorization Grant secondo l'[RFC 8628](#). Consente ai dispositivi con input limitato (smart TV, dispositivi IoT) di ottenere l'autorizzazione dell'utente facendo sì che l'utente effettui l'autorizzazione su un dispositivo secondario.

Il tipo di autorizzazione può essere personalizzato durante la registrazione dell'app; per impostazione predefinita, tutti i tipi di autorizzazione sono supportati.

Endpoints

Per impostazione predefinita, il componente è configurato con i seguenti endpoint per gestire le richieste di autorizzazione e token

Authorization: `/sgc/oauth2/auth`

Token: `/sgc/oauth2/token`

Revocation: `/sgc/oauth2/revoke`

Introspection: `/sgc/oauth2/introspect`

Autorizzazione dispositivo: `/sgc/oauth2/device`

Verifica del dispositivo: `/sgc/oauth2/device/verify`

Quindi se il server è in ascolto sulla porta 443 e il dominio è `www.esegece.com`, gli EndPoint saranno:

Authorization: <https://www.esegece.com/sgc/oauth2/auth>

Token: <https://www.esegece.com/sgc/oauth2/token>

Revocation: <https://www.esegece.com/sgc/oauth2/revoke>

Introspezione: <https://www.esegece.com/sgc/oauth2/introspect>

Device Authorization: <https://www.esegece.com/sgc/oauth2/device>

Verifica dispositivo: <https://www.esegece.com/sgc/oauth2/device/verify>

Gli endpoint possono essere configurati nella proprietà `OAuth2Options`.

Per impostazione predefinita, **PKCE** (un'estensione al flusso Authorization Code per prevenire attacchi CSRF e injection del codice di autorizzazione) è abilitato.

Configurazione

Prima di poter avviare il processo OAuth2, è necessario registrare le app che saranno disponibili; ciò avviene tramite la proprietà Apps del componente server OAuth2.

Registra app

Utilizzare Apps.AddApp per aggiungere una nuova applicazione al server OAuth2; è necessario impostare i seguenti parametri:

- **App Name:** è il nome dell'applicazione. Esempio: MyApp
- **RedirectURI:** è il punto dove verranno reindirizzate le risposte. Esempio: http://127.0.0.1:8080
- **ClientId:** è un'informazione pubblica ed è l'ID del client.
- **ClientSecret:** deve essere mantenuto riservato.

Facoltativamente è possibile impostare i seguenti parametri:

- **ExpiresIn:** per impostazione predefinita è 3600 secondi, quindi il token scadrà in 1 ora; è possibile impostare un valore maggiore se necessario.
- **RefreshToken:** per impostazione predefinita i token di aggiornamento sono supportati; se non lo sono, impostare questo parametro su false.
- **AllowedGrantTypes:** per impostazione predefinita tutti i tipi di grant sono supportati (auth2Code e auth2ClientCredentials), ma il server può essere configurato per consentire solo l'Authorization Code o solo le Client Credentials.

Elimina app

Utilizzare Apps.RemoveApp per eliminare un'App esistente.

AddToken

Se il server è stato riavviato mentre erano stati emessi alcuni token, è possibile recuperare questi token utilizzando il metodo AddToken prima di avviare il Server OAuth2 e dopo aver registrato le App

- **AppName:** il nome dell'applicazione.
- **Token:** token di accesso.
- **Expires:** quando il token scade.
- **RefreshToken:** token di aggiornamento.

RemoveToken

Rimuove un Token già emesso.

OAuth2Options

La proprietà OAuth2Options consente di configurare gli endpoint del server e le funzionalità opzionali.

Revoca

Revoca del token secondo [RFC 7009](#). Se abilitata, i client possono revocare token di accesso o refresh precedentemente emessi.

- **OAuth2Options.Revocation.Enabled:** impostare su True per abilitare l'endpoint di revoca.
- **OAuth2Options.Revocation.URL:** il percorso URL dell'endpoint. Predefinito: /sgc/oauth2/revoke

Introspezione

Introspezione del token per [RFC 7662](#). Quando abilitata, i resource server possono interrogare il server di autorizzazione per determinare lo stato attivo e i metadati di un token.

- **OAuth2Options.Introspection.Enabled:** impostare su True per abilitare l'endpoint di introspezione.
- **OAuth2Options.Introspection.URL:** il percorso URL dell'endpoint. Predefinito: /sgc/oauth2/introspect

DeviceAuthorization

Device Authorization Grant secondo [RFC 8628](#). Quando abilitato, i dispositivi con input limitato possono richiedere l'autorizzazione facendo autorizzare l'utente su un dispositivo secondario.

- **OAuth2Options.DeviceAuthorization.Enabled:** impostare su True per abilitare l'endpoint di autorizzazione del dispositivo.
- **OAuth2Options.DeviceAuthorization.URL:** il percorso URL dell'endpoint per le richieste di device code. Predefinito: /sgc/oauth2/device
- **OAuth2Options.DeviceAuthorization.VerificationURL:** il percorso URL dell'endpoint per la pagina di verifica utente. Valore predefinito: /sgc/oauth2/device/verify
- **OAuth2Options.DeviceAuthorization.ExpiresIn:** la durata in secondi del codice dispositivo. Predefinito: 600 (10 minuti).
- **OAuth2Options.DeviceAuthorization.Interval:** l'intervallo di polling minimo in secondi che il client dovrebbe utilizzare quando esegue il polling dell'endpoint token. Predefinito: 5.

Riferimento

- Proprietà — 2 proprietà disponibili sul componente.
- Metodi — 11 metodi pubblici.
- Eventi — 13 eventi da gestire.

Guide

- **QuickStart**
 - [Esempio Server OAuth2](#)
 - [OAuth2 Personalizza HTML di Accesso](#)
 - [Endpoint Server OAuth2](#)
 - [OAuth2 Registra App](#)
 - [OAuth2 Recupera Token di Accesso](#)
- **Authenticate**
 - [OAuth2 Server Authentication](#)
 - [OAuth2 None Authenticate some URLs](#)

OAuth2 | Esempio server

Facciamo un semplice esempio di server OAuth2, utilizzando un [TsgcWebSocketHTTPServer](#).

Prima di tutto, creare un nuovo `TsgcWebSocketHTTPServer` in ascolto sulla porta 443 e utilizzando un certificato autofirmato nel file `sgc.pem`.

```
oServer = new TsgcWebSocketHTTPServer();
oServer.Port = 80;
oServer.SSLOptions.Port = 443;
oServer.SSLOptions.CertFile = "sgc.pem";
oServer.SSLOptions.KeyFile = "sgc.pem";
oServer.SSLOptions.RootCertFile = "sgc.pem";
oServer.SSL = true;
```

Quindi crei una nuova istanza di `TsgcHTTP_OAuth2_Server` e la assegni al server creato in precedenza. Registra una nuova applicazione con i seguenti valori:

Name: MyApp
 RedirectURI: http://127.0.0.1:8080
 ClientId: client-id
 ClientSecret: client-secret

```
OAuth2 = new TsgcHTTP_OAuth2_Server.Create();
OAuth2.Apps.AddApp("MyApp", "http://127.0.0.1:8080", "client-id", "client-secret");
oServer.Authentication.Enabled = true;
oServer.Authentication.OAuth.OAuth2 = OAuth2;
```

Gestisca quindi l'evento `OnOAuth2Authentication` del componente server OAuth2 e implementi il proprio metodo per l'accesso degli utenti. Utilizzerò la coppia "user/secret" per accettare un accesso.

```
void OnOAuth2Authentication(TsgcWSConnection Connection, TsgcHTTPOAuth2Request OAuth2, string aUser,
    string aPassword, ref bool Authenticated)
{
    if ((aUser == "user") and (aPassword == "secret"))
    {
        Authenticated = true;
    }
}
```

Infine, avviare il server e utilizzare un client OAuth2 per effettuare il login; ad esempio è possibile utilizzare il componente [TsgcHTTP_OAuth2_Client](#) incluso nella libreria `sgcWebSockets`.

 OAuth2

Configuration

Auth. URL

Token URL

Scope

Authorization Server Options

IP

Redirect URL

Port

Local Server Options

ClientId

Secret

Username

Password

OAuth2 Options

New Access Token

Access Token

Token Type

Expires In

Refresh Token

Scope

Refresh Token

Richiede un nuovo Access Token; verrà visualizzata una nuova sessione del browser web e l'utente dovrà consentire la connessione e quindi effettuare il login.

OAuth2 Authorization

Sign in

Introduce your username and password.

GO BACK

Sign in

SIGN IN

Se il login ha esito positivo, al client viene restituito un nuovo Token. Tutte le richieste successive devono includere questo token bearer nelle intestazioni HTTP.

 OAuth2

Configuration

Gmail

Auth. URL

https://127.0.0.1/sgc/oauth2/auth

Token URL

https://127.0.0.1/sgc/oauth2/token

Scope

scope

Authorization Server Options

IP

127.0.0.1

Port

8080

Redirect URL

Local Server Options

ClientId

client-id

Secret

client-secret

Username

Password

OAuth2 Options

New Access Token

Access Token

be4d115a9e6a44f0a859cb303d7f03890d3bf290fc4342c1a08befa550b738bd

Token Type

Bearer

Expires In

3600

Refresh Token

b5ec418745ef4c9e94d3b1a90b34324826152b7edbf040a6a55271813aac5849

Scope

scope

Refresh Token

After Authorize Code: code=4b387ffb4255412083057f29ca35933a
 state=EB2FD5EB11B346BFB9CE14F7974DBEE7

After Access Token:


```
{
  "token_type": "Bearer",
  "access_token": "be4d115a9e6a44f0a859cb303d7f03890d3bf290fc4342c1a08befa550b738bd",
  "expires_in": 3600,
  "refresh_token": "b5ec418745ef4c9e94d3b1a90b34324826152b7edbf040a6a55271813aac5849",
  "scope": "scope"
}
```

OAuth2 | Personalizza HTML di accesso

Quando un client OAuth2 effettua una richiesta per ottenere un nuovo Access Token, viene visualizzata una pagina Web nel browser per consentire la connessione e accedere con nome utente e password.

La pagina HTML è inclusa per impostazione predefinita nel componente server, ma questo codice può essere personalizzato utilizzando l'evento **OnAuth2BeforeDispatchPage**.

```
void OnAuth2BeforeDispatchPage(TObject Sender; TsgcHTTPOAuth2Request OAuth2; ref string HTML)
{
    HTML = "your custom html";
}
```

Se si personalizza l'HTML con un codice HTML completamente nuovo, è necessario mantenere almeno il form in cui vengono inviati Username e password:

```
<form action="">
<input type="hidden" name="request_type" value="sign-in" />
<input type="username" name="username" placeholder="Username" />
<input type="password" name="password" placeholder="Password" />
<input type="hidden" name="id" value="" />
<p></p>
<button>Sign In</button>
</form>
```

Il parametro id, che è nascosto, deve mantenere lo stesso valore del modulo originale per consentire al server di identificare la richiesta.

OAuth2 | Endpoint del Server

Per impostazione predefinita, il Server OAuth2 utilizza i seguenti Endpoint:

Authorization: /sgc/oauth2/auth

Token: /sgc/oauth2/token

Revocation: /sgc/oauth2/revoke (POST) - Revoca i token secondo [RFC 7009](#)

Introspection: /sgc/oauth2/introspect (POST) - Restituisce i metadati del token secondo [RFC 7662](#)

Device Authorization: /sgc/oauth2/device (POST) - Emette codici dispositivo secondo [RFC 8628](#)

Verifica dispositivo: /sgc/oauth2/device/verify (GET/POST) - Pagina di verifica utente

Il che significa che se il server è in ascolto sull'IP 80.54.41.30 e sulla porta 8443, gli endpoint OAuth2 completi saranno:

Authorization: https://80.54.41.30:8443/sgc/oauth2/auth

Token: https://80.54.41.30:8443/sgc/oauth2/token

Revocation: https://80.54.41.30:8443/sgc/oauth2/revoke

Introspection: https://80.54.41.30:8443/sgc/oauth2/introspect

Device Authorization: https://80.54.41.30:8443/sgc/oauth2/device

Verifica Dispositivo: https://80.54.41.30:8443/sgc/oauth2/device/verify

Questi endpoint possono essere modificati facilmente, accedendo alla proprietà OAuth2Options del componente e modificando gli URL di autorizzazione e token.

Esempio: se i propri endpoint devono essere

Authorization: https://80.54.41.30:8443/authentication/auth

Token: https://80.54.41.30:8443/authentication/token

Impostare la proprietà OAuth2Options con i seguenti valori:

OAuth2Options.Authorization.URL = /authentication/auth

OAuth2Options.Token.URL = /authentication/token

Lo stesso approccio si applica agli altri endpoint:

OAuth2Options.Revocation.URL = /authentication/revoke

OAuth2Options.Introspection.URL = /authentication/introspect

OAuth2Options.DeviceAuthorization.URL = /authentication/device

OAuth2Options.DeviceAuthorization.VerificationURL = /authentication/device/verify

OAuth2 | Registrazione delle app

Prima che un nuovo OAuth2 venga richiesto da un client, l'App deve essere registrata nel server. La registrazione di una nuova App richiede le seguenti informazioni:

- **App Name:** è il nome dell'applicazione. Esempio: MyApp
- **RedirectURI:** è il punto dove verranno reindirizzate le risposte. Esempio: `http://127.0.0.1:8080`
- **ClientId:** è un'informazione pubblica ed è l'ID del client.
- **ClientSecret:** deve essere mantenuto riservato.

Facoltativamente è possibile impostare i seguenti parametri:

- **ExpiresIn:** per impostazione predefinita è 3600 secondi, quindi il token scadrà in 1 ora; è possibile impostare un valore maggiore se necessario.
- **RefreshToken:** per impostazione predefinita i token di aggiornamento sono supportati; se non lo sono, impostare questo parametro su false.

Se un nuovo client vuole autenticarsi tramite OAuth2, prima l'App deve essere registrata nel server, è possibile utilizzare:

1. RegisterApp

Questo metodo richiede il nome dell'app e il RedirectURI e restituirà un ClientId e un ClientSecret.

2. Apps.AddApp

Questo metodo richiede AppName, RedirectURI, ClientId e ClientSecret. Di solito è possibile utilizzare questo metodo quando un server dispone di alcune App già create e si desidera caricarle prima dell'avvio.

Entrambi i metodi fanno la stessa cosa, registrano l'applicazione nel server, ma il primo è più utile quando l'app viene registrata per la prima volta e il secondo metodo quando si vogliono caricare tutte le app registrate prima di avviare il server (perché sono salvate nel database, ad esempio).

OAuth2 | Recupera token di accesso

Se il server OAuth2 viene distrutto (perché è stato riavviato) e sono stati emessi Access Token validi, questi vengono persi per impostazione predefinita. È possibile recuperare questi Access Token utilizzando il metodo **AddToken**. Questo metodo memorizza i token nel server OAuth2.

Per aggiungere un token sono necessarie le seguenti informazioni:

- **AppName:** il nome dell'applicazione.
- **Token:** token di accesso.
- **Expires:** quando il token scade.
- **RefreshToken:** token di aggiornamento.

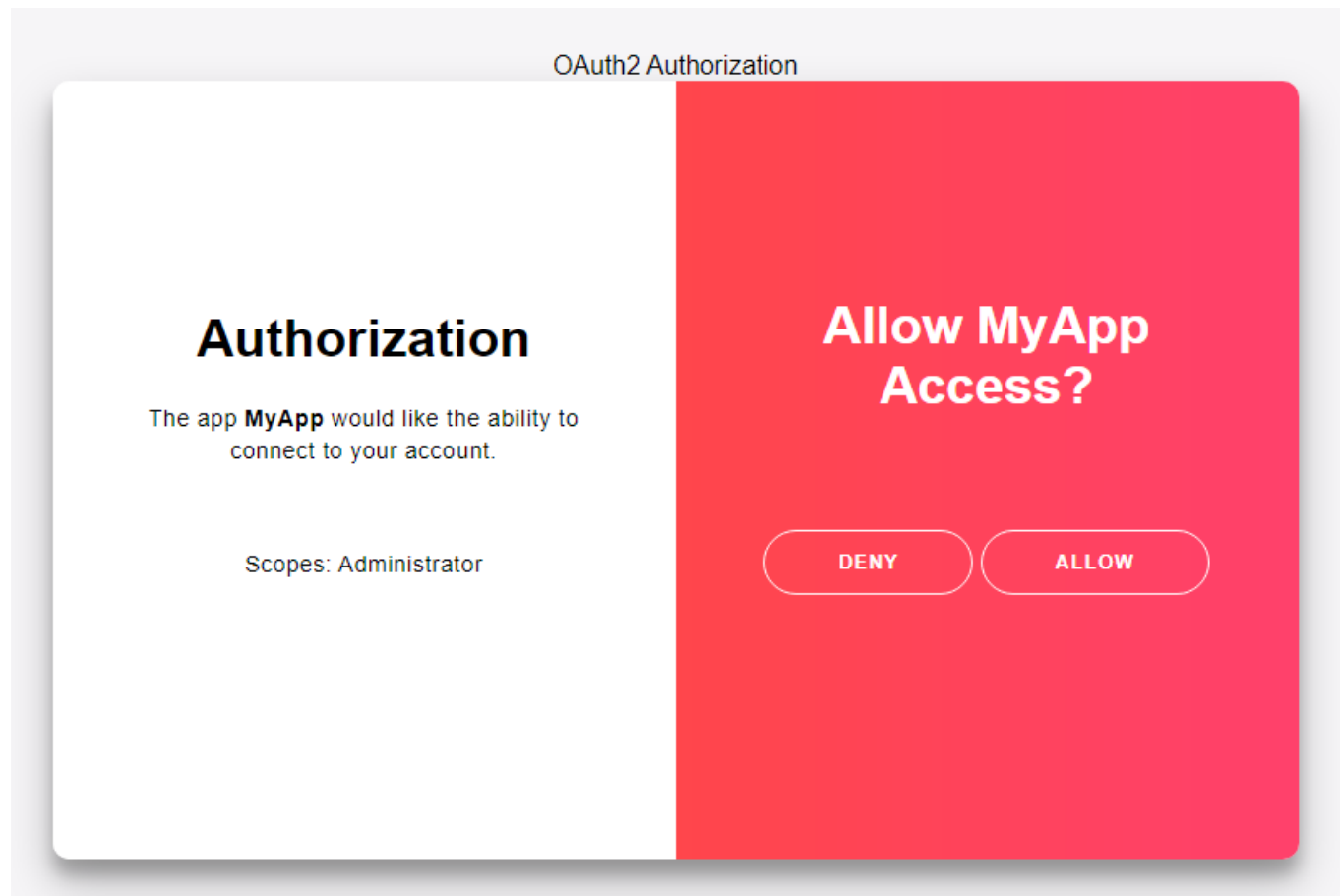
È possibile salvare i token emessi gestendo l'evento **OAuth2AfterAccessToken**.

```
private void OnOAuth2AfterAccessToken(TObject Sender, TsgcWSConnection Connection, TsgcHTTPOAuth2Request OAuth2,
    string aResponse)
{
    // ... store OAuth2 Token data
}

OAuth2 = new TsgcHTTP_OAuth2_Server.Create();
OAuth2.Apps.AddApp("MyApp", "http://127.0.0.1:8080", "client-id", "client-secret");
OAuth2.AddToken("MyApp", "12146ce12b0e4813987f2794f768905cefc39da6fbd54f6d9b38387489280608", 1622796714,
    "ef3e3dfa56ec44109c3d345b1541f08e539ce21432d9433099b48a3d08d34bc0");
oServer.Authentication.Enabled = true;
oServer.Authentication.OAuth.OAuth2 = OAuth2;
```

OAuth2 | Autenticazione Server

Quando un client OAuth2 richiede una nuova autorizzazione, il server mostra una pagina web in cui l'utente deve consentire la connessione e quindi effettuare il login. Questa pagina è fornita dalla libreria `sgcWebSockets` e viene distribuita automaticamente quando un client richiede un'autorizzazione.



Se l'utente consente l'accesso, verrà mostrato un modulo di login in cui l'utente dovrà inserire nome utente e password.

Questi dati verranno ricevuti nell'evento `OnOAuth2Authentication`; è quindi necessario verificare che l'utente/password sia corretto e, in caso affermativo, impostare il parametro `Authenticated` su `true`.

```
void OnOAuth2Authentication(TsgcWSConnection Connection, TsgcHTTPOAuth2Request OAuth2, string aUser,
    string aPassword, ref bool Authenticated)
{
    if ((aUser == "user") and (aPassword == "secret"))
    {
        Authenticated = true;
    }
}
```

OAuth2 | URL di autenticazione None

Per impostazione predefinita, quando OAuth2 è abilitato lato server, tutte le richieste HTTP richiedono l'autenticazione tramite Bearer Token.

Se si desidera consentire l'accesso ad alcuni URL senza la necessità di utilizzare un Bearer Token, è possibile utilizzare l'evento **OnOAuth2BeforeRequest**

```
procedure OnOAuth2BeforeRequest(TObject Sender; TsgcWSConnection aConnection; TStringList aHeaders;  
    ref bool Cancel)  
{  
    if (DecodeGETFullPath(aHeaders) == "/Public.html")  
    {  
        Cancel = true;  
    }  
}
```


OAuth2 Server | Authorization Code Grant

Panoramica

Il grant Authorization Code è il flusso OAuth2 più comune per le applicazioni lato server. Il componente [TsgcHTTP_OAuth2_Server](#) fornisce due endpoint per gestire questo flusso: un endpoint di autorizzazione che mostra all'utente una pagina di accesso e un endpoint Token che scambia il codice di autorizzazione con un token di accesso.

Questo flusso è adatto per applicazioni web, applicazioni desktop e applicazioni mobili in cui il client può interagire con un browser.

Flow

1. Il client reindirizza l'utente a `/sgc/oauth2/auth?response_type=code&client_id=...&redirect_uri=...&scope=...&state=...`
2. Il server genera **OnOAuth2BeforeDispatchPage** per consentire la personalizzazione della pagina HTML di accesso.
3. L'utente invia le credenziali. Il server genera **OnOAuth2Authentication** per convalidare il nome utente e la password.
4. Se autenticato, il server reindirizza l'utente al `redirect_uri` con `?code=...&state=...` aggiunto.
5. Il client esegue un POST su `/sgc/oauth2/token` con `grant_type=authorization_code&code=...&redirect_uri=...&client_id=...&client_secret=...`
6. Il server convalida il codice di autorizzazione e le credenziali del client, genera **OnOAuth2AfterAccessToken** e restituisce una risposta JSON contenente l'`access_token`, il `token_type`, l'`expires_in` e, facoltativamente, un `refresh_token`.

Configurazione

Proprietà	Descrizione
<code>OAuth2Options.Authorization.Enabled</code>	Imposti a True per abilitare l'endpoint di autorizzazione. Predefinito: True.
<code>OAuth2Options.Authorization.URL</code>	Il percorso URL dell'endpoint di autorizzazione. Predefinito: <code>/sgc/oauth2/auth</code>
<code>OAuth2Options.Token.Enabled</code>	Impostare a True per abilitare l'endpoint token. Predefinito: True.
<code>OAuth2Options.Token.URL</code>	Il percorso URL dell'endpoint del token. Predefinito: <code>/sgc/oauth2/token</code>
<code>OAuth2Options.PKCE</code>	Se True, il server richiede PKCE (<code>code_challenge</code> e <code>code_verifier</code>) per le richieste di codice di autorizzazione. Predefinito: True.

App Registration

Prima che i client possano utilizzare il flusso Authorization Code, deve registrare l'applicazione sul server. Utilizzi `Apps.AddApp` e si assicuri che `AllowedGrantTypes` includa `auth2Code`.

```
OAuth2Server.Apps.AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2Code });
```

Eventi

Evento	Descrizione
<code>OnOAuth2BeforeDispatchPage</code>	Generato prima che la pagina HTML di accesso venga inviata all'utente. Consente la personalizzazione del contenuto HTML.
<code>OnOAuth2Authentication</code>	Si genera quando l'utente invia le credenziali. Validare qui il nome utente e la password e impostare <code>Authenticated</code> a True o False.
<code>OnOAuth2AfterAccessToken</code>	Generato dopo che il server emette con successo un token di accesso. Utile per la registrazione dei log.
<code>OnOAuth2AfterValidateAccessToken</code>	Viene attivato quando un client effettua una richiesta con un token valido e non scaduto. Utile per la registrazione e il controllo.

Esempio

L'esempio seguente mostra una configurazione completa del server con il grant Authorization Code, inclusa la registrazione dell'app e i gestori di eventi.

```
// Create and configure OAuth2 server
var OAuth2Server = new TsgcHTTP_OAuth2_Server();

// Register an application
OAuth2Server.Apps.AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2Code });

// Attach to HTTP server
Server.Authentication.Enabled = true;
Server.Authentication.OAuth.OAuth2 = OAuth2Server;

OAuth2Server.OnOAuth2Authentication += (sender, user, password, ref authenticated) =>
{
    authenticated = (user == "admin") && (password == "secret");
};

OAuth2Server.OnOAuth2AfterAccessToken += (sender, token, refreshToken, expiresIn) =>
{
    Console.WriteLine("Access token issued: " + token);
};
```

Supporto PKCE

Quando `OAuth2Options.PKCE` è impostato su True (il valore predefinito), il server applica Proof Key for Code Exchange. Il client deve includere un parametro `code_challenge` nella richiesta di autorizzazione e un corrispondente `code_verifier` nella richiesta di token. Il server verifica che il `code_verifier` corrisponda al `code_challenge` originale prima di emettere un token.

PKCE è consigliato per tutti i client, in particolare per i client pubblici (app native e SPA) che non possono archiviare in modo sicuro un client secret.

OAuth2 Server | Concessione tramite credenziali client

Panoramica

La concessione Client Credentials viene utilizzata per la comunicazione machine-to-machine (M2M) senza interazione dell'utente. Il client si autentica direttamente con [TsgcHTTP_OAuth2_Server](#) utilizzando le proprie credenziali (client_id e client_secret) e riceve un token di accesso. Non è previsto alcun passaggio con codice di autorizzazione, nessun reindirizzamento e nessuna pagina di login utente.

Questo tipo di concessione è appropriato quando l'applicazione client agisce per conto proprio piuttosto che per conto di un utente specifico.

Flow

1. Il client invia una POST all'endpoint token con
`grant_type=client_credentials&client_id=...&client_secret=...&scope=...`
2. Il server convalida le credenziali del client rispetto all'applicazione registrata.
3. Il server attiva **OnOAuth2AfterAccessToken** e restituisce una risposta JSON con `access_token`, `token_type` ed `expires_in`.

Si noti che nessun evento `OnOAuth2Authentication` viene attivato per questo tipo di grant, poiché non c'è alcun utente da autenticare. Il server convalida automaticamente `client_id` e `client_secret` rispetto all'app registrata.

Configurazione

Registrare l'applicazione con `AllowedGrantTypes` che includa `auth2ClientCredentials`. Per questo flusso è necessario solo l'endpoint del token.

Proprietà	Descrizione
<code>OAuth2Options.Token.Enabled</code>	Impostare a True per abilitare l'endpoint token. Predefinito: True.
<code>OAuth2Options.Token.URL</code>	Il percorso URL dell'endpoint del token. Predefinito: <code>/sgc/oauth2/token</code>

App Registration

```
OAuth2Server.Apps.AddApp("MyServiceApp", "",  
    "service-client-id", "service-client-secret", 3600, false,  
    new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2ClientCredentials });
```

Il `RedirectURI` può essere lasciato vuoto poiché non è coinvolto alcun redirect. `RefreshToken` è tipicamente impostato su False perché il client può richiedere un nuovo token in qualsiasi momento utilizzando le sue credenziali.

Eventi

Evento	Descrizione
<code>OnOAuth2AfterAccessToken</code>	Generato dopo che il server emette con successo un token di accesso. Utile per la registrazione dei log.

OnOAuth2AfterValidateAccessToken

Attivato quando un client effettua una richiesta con un token valido. Utile per l'auditing.

Esempio

```
var OAuth2Server = new TsgcHTTP_OAuth2_Server();
OAuth2Server.Apps.AddApp("BackendService", "",
    "service-client-id", "service-client-secret", 7200, false,
    new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2ClientCredentials });
Server.Authentication.Enabled = true;
Server.Authentication.OAuth.OAuth2 = OAuth2Server;
OAuth2Server.OnOAuth2AfterAccessToken += (sender, token, refreshToken, expiresIn) =>
{
    Console.WriteLine("Service token issued: " + token);
};
```

Quando Utilizzarlo

Utilizzare il grant Client Credentials sul server per:

- Servizi backend e daemon che si autenticano senza interazione dell'utente.
- Comunicazione API da server a server.
- Autenticazione da microservizio a microservizio.
- Processi automatizzati, job pianificati e operazioni batch.

OAuth2 Server | Resource Owner Password Credentials Grant

Panoramica

Il grant Resource Owner Password Credentials (ROPC) consente al client di inviare direttamente le credenziali dell'utente (username e password) all'endpoint token. Il [TsgcHTTP_OAuth2_Server](#) valida le credenziali tramite l'evento **OnOAuth2Authentication** e rilascia un token di accesso se l'autenticazione ha esito positivo.

Questo flusso non prevede un reindirizzamento del browser o una pagina di accesso. Il client raccoglie le credenziali direttamente dall'utente e le invia al server.

Avviso di sicurezza

Questo tipo di grant è deprecato in OAuth 2.1. Deve essere utilizzato solo quando altri flussi (come Authorization Code con PKCE) non sono fattibili, ad esempio nelle applicazioni legacy o nei client di prima parte altamente attendibili. Il grant Authorization Code con PKCE è l'alternativa raccomandata.

Flow

1. Il client invia una richiesta POST all'endpoint token con
grant_type=password&username=...&password=...&client_id=...&client_secret=...&scope=...
2. Il server genera **OnOAuth2Authentication** per convalidare nome utente e password.
3. Se autenticato, il server genera **OnOAuth2AfterAccessToken** e restituisce una risposta JSON con access_token, token_type, expires_in e facoltativamente un refresh_token.
4. Se l'autenticazione fallisce, il server restituisce una risposta di errore.

Configurazione

Registrare l'applicazione con AllowedGrantTypes che include auth2ResourceOwnerPassword.

Proprietà	Descrizione
<code>OAuth2Options.Token.Enabled</code>	Impostare a True per abilitare l'endpoint token. Predefinito: True.
<code>OAuth2Options.Token.URL</code>	Il percorso URL dell'endpoint del token. Predefinito: /sgc/oauth2/token

App Registration

```
OAuth2Server.Apps.AddApp("MyApp", "",
    "my-client-id", "my-client-secret", 3600, true,
    new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2ResourceOwnerPassword });
```

Eventi

Evento	Descrizione

<code>OnOAuth2Authentication</code>	Viene generato quando il server riceve nome utente e password. Validare le credenziali qui e impostare <code>Authenticated</code> su True o False.
<code>OnOAuth2AfterAccessToken</code>	Generato dopo che il server emette con successo un token di accesso. Utile per la registrazione dei log.
<code>OnOAuth2AfterValidateAccessToken</code>	Generato quando un client effettua una richiesta con un token valido.

Esempio

```
var OAuth2Server = new TsgcHTTP_OAuth2_Server();
OAuth2Server.Apps.AddApp("MyApp", "",
    "my-client-id", "my-client-secret", 3600, true,
    new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2ResourceOwnerPassword });
Server.Authentication.Enabled = true;
Server.Authentication.OAuth.OAuth2 = OAuth2Server;
OAuth2Server.OnOAuth2Authentication += (sender, user, password, ref authenticated) =>
{
    authenticated = ValidateUser(user, password);
};
OAuth2Server.OnOAuth2AfterAccessToken += (sender, token, refreshToken, expiresIn) =>
{
    Console.WriteLine("Token issued for user: " + token);
};
```

Quando Utilizzarlo

Utilizzare il grant Resource Owner Password Credentials solo quando:

- Il client è un'applicazione first-party con un elevato livello di attendibilità.
- Altri tipi di grant (Authorization Code con PKCE) non sono praticabili.
- Migrazione di applicazioni legacy che raccolgono già le credenziali dell'utente direttamente.

Per le nuove applicazioni, preferire sempre la concessione [Authorization Code](#) con PKCE.

OAuth2 Server | Concessione token di aggiornamento

Panoramica

Il grant Refresh Token consente a un client di scambiare un refresh token precedentemente emesso con un nuovo token di accesso senza richiedere l'interazione dell'utente. Il [TsgcHTTP_OAuth2_Server](#) convalida il refresh token ed emette un nuovo token di accesso (e opionalmente un nuovo refresh token).

Ciò è utile per le sessioni di lunga durata in cui il token di accesso ha una scadenza breve ma il client necessita di accesso continuo alle risorse protette.

Flow

1. Il client invia una POST all'endpoint del token con
`grant_type=refresh_token&refresh_token=...&client_id=...&client_secret=...&scope=...`
2. Il server convalida il refresh token rispetto ai token memorizzati.
3. Il server genera **OnOAuth2AfterRefreshToken** e restituisce una risposta JSON con un nuovo `access_token`, `token_type`, `expires_in` e, facoltativamente, un nuovo `refresh_token`.

Configurazione

I token di aggiornamento sono abilitati per applicazione al momento della registrazione dell'app. Impostare il parametro `RefreshToken` a `True` in `Apps.AddApp`.

Proprietà	Descrizione
<code>OAuth2Options.Token.Enabled</code>	Impostare a <code>True</code> per abilitare l'endpoint token. Predefinito: <code>True</code> .
<code>OAuth2Options.Token.URL</code>	Il percorso URL dell'endpoint del token. Predefinito: <code>/sgc/oauth2/token</code>
<code>RefreshToken</code> (AddApp parameter)	Impostare su <code>True</code> per emettere token di aggiornamento per questa applicazione. Valore predefinito: <code>True</code> .

App Registration

Durante la registrazione dell'app, imposti il parametro `RefreshToken` su `True` in modo che il server emetta refresh token insieme agli access token.

```
// RefreshToken = true (6th parameter)
OAuth2Server.Apps.AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2Code });
```

Eventi

Evento	Descrizione

<code>OnOAuth2AfterRefreshToken</code>	Viene attivato dopo che il server emette con successo un nuovo token di accesso da un token di aggiornamento. Utile per la registrazione della rotazione dei token.
<code>OnOAuth2AfterAccessToken</code>	Generato dopo l'emissione del nuovo token di accesso.

Esempio

```
var OAuth2Server = new TsgcHTTP_OAuth2_Server();
OAuth2Server.Apps.AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2Code });
Server.Authentication.Enabled = true;
Server.Authentication.OAuth.OAuth2 = OAuth2Server;
OAuth2Server.OnOAuth2AfterRefreshToken += (sender, oldToken, newToken, newRefreshToken, expiresIn) =>
{
    Console.WriteLine("Token refreshed. New token: " + newToken);
};
```

Rotazione dei token

Quando viene utilizzato un token di aggiornamento, il server può emettere un nuovo token di aggiornamento insieme al nuovo token di accesso. Il vecchio token di aggiornamento viene invalidato. Questo meccanismo è noto come rotazione del token di aggiornamento e contribuisce a prevenire gli attacchi di replay del token.

Ripristino dei token dopo il riavvio

Se il server viene riavviato mentre i token sono ancora validi, può recuperarli usando `Apps.AddToken` prima di avviare il server. Per i dettagli veda [OAuth2 Recover Access Tokens](#).

OAuth2 Server | Device Authorization Grant (RFC 8628)

Panoramica

Il Device Authorization Grant è progettato per dispositivi con input limitato (smart TV, dispositivi IoT, strumenti CLI) che non possono facilmente visualizzare un browser o accettare input da tastiera. Il componente [TsgcHTTP_OAuth2_Server](#) fornisce un endpoint di autorizzazione del dispositivo e una pagina di verifica in modo che gli utenti possano autorizzare il dispositivo da un dispositivo secondario come un telefono o un computer.

Flow

1. Il dispositivo invia una POST a `/sgc/oauth2/device` con `client_id=...&scope=...`
2. Il server genera **OnOAuth2DeviceAuthorization** e restituisce una risposta JSON con `device_code`, `user_code`, `verification_uri`, `expires_in` e `interval`.
3. Il dispositivo visualizza all'utente il `user_code` e il `verification_uri` (ad es. "Andare su <https://example.com/sgc/oauth2/device/verify> e inserire il codice: ABCD-1234").
4. L'utente naviga verso l'URI di verifica su un dispositivo secondario e inserisce il `user_code`.
5. Il server attiva **OnOAuth2DeviceCodeVerification** per validare lo `user_code` e autorizzare il dispositivo.
6. Nel frattempo, il dispositivo esegue il polling dell'endpoint token con `grant_type=urn:ietf:params:oauth:grant-type:device_code&device_code=...&client_id=...`
7. Il server restituisce `authorization_pending` finché l'utente non autorizza. Una volta autorizzato, restituisce l'access token.

Configurazione

Proprietà	Descrizione
<code>OAuth2Options.DeviceAuthorization.Enabled</code>	Impostare su True per abilitare l'endpoint di autorizzazione del dispositivo. Predefinito: False.
<code>OAuth2Options.DeviceAuthorization.URL</code>	Il percorso URL dell'endpoint di autorizzazione del dispositivo. Predefinito: <code>/sgc/oauth2/device</code>
<code>OAuth2Options.DeviceAuthorization.VerificationURL</code>	Il percorso URL della pagina di verifica in cui l'utente inserisce il codice. Predefinito: <code>/sgc/oauth2/device/verify</code>
<code>OAuth2Options.DeviceAuthorization.ExpiresIn</code>	La durata in secondi del codice dispositivo. Predefinito: 600 (10 minuti).
<code>OAuth2Options.DeviceAuthorization.Interval</code>	L'intervallo minimo di polling in secondi che il dispositivo deve utilizzare. Valore predefinito: 5.

Eventi

Evento	Descrizione
<code>OnOAuth2DeviceAuthorization</code>	Generato quando un dispositivo richiede un codice dispositivo. Consente di personalizzare la risposta o registrare la richiesta.
<code>OnOAuth2DeviceCodeVerification</code>	Generato quando un utente invia un codice di verifica nella pagina di verifica del dispositivo. Convalidare lo <code>user_code</code> e autorizzare o negare il dispositivo.
<code>OnOAuth2AfterAccessToken</code>	Si attiva dopo che il server ha emesso il token di accesso al dispositivo.

Esempio

```
var OAuth2Server = new TsgcHTTP_OAuth2_Server();
OAuth2Server.OAuth2Options.DeviceAuthorization.Enabled = true;
OAuth2Server.OAuth2Options.DeviceAuthorization.ExpiresIn = 600;
OAuth2Server.OAuth2Options.DeviceAuthorization.Interval = 5;
OAuth2Server.Apps.AddApp("DeviceApp", "",
    "device-client-id", "device-client-secret", 3600, true,
    new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2Code });
Server.Authentication.Enabled = true;
Server.Authentication.OAuth.OAuth2 = OAuth2Server;
OAuth2Server.OnOAuth2DeviceCodeVerification += (sender, userCode, ref authorized) =>
{
    authorized = true;
};
OAuth2Server.OnOAuth2AfterAccessToken += (sender, token, refreshToken, expiresIn) =>
{
    Console.WriteLine("Device authorized. Token: " + token);
};
```

Comportamento di polling

Finché l'utente non ha ancora autorizzato il dispositivo, il dispositivo interroga l'endpoint token all'intervallo configurato. Il server risponde con:

- `authorization_pending` -- l'utente non ha ancora completato l'autorizzazione.
- `slow_down` -- il dispositivo sta effettuando il polling troppo frequentemente. Il dispositivo dovrebbe aumentare l'intervallo di 5 secondi.
- `expired_token` -- il codice del dispositivo è scaduto. Il dispositivo deve riavviare il flusso.
- `access_denied` -- l'utente ha negato l'autorizzazione.

Una volta che l'utente autorizza il dispositivo, il successivo polling restituisce il token di accesso.

Server OAuth2 | Revoca del token (RFC 7009)

Panoramica

La revoca del token consente ai client di notificare a [TsgcHTTP_OAuth2_Server](#) che un token di accesso o un token di aggiornamento precedentemente emesso non è più necessario. Il server invalida il token in modo che non possa più essere utilizzato per accedere alle risorse protette.

Ciò è utile quando un utente effettua il logout, quando un'applicazione viene disinstallata o quando un token potrebbe essere stato compromesso.

Endpoint

POST a `/sgc/oauth2/revoke` con i seguenti parametri:

Parametro	Descrizione
<code>token</code>	Il token da revocare (obbligatorio).
<code>token_type_hint</code>	Suggerimento facoltativo sul tipo di token: <code>access_token</code> o <code>refresh_token</code> .

Comportamento

Come previsto dall'[RFC 7009](#), il server restituisce sempre HTTP 200 OK indipendentemente dal fatto che il token sia stato trovato o revocato con successo. Ciò impedisce la divulgazione dell'esistenza del token -- un client non può determinare se un token era valido osservando lo stato della risposta.

Quando un refresh token viene revocato, anche l'access token associato può essere invalidato (dipende dall'implementazione). Quando un access token viene revocato, il refresh token associato rimane valido a meno che non venga revocato esplicitamente.

Configurazione

Proprietà	Descrizione
<code>OAuth2Options.Revocation.Enabled</code>	Impostare su True per abilitare l'endpoint di revoca. Predefinito: False.
<code>OAuth2Options.Revocation.URL</code>	Il percorso URL dell'endpoint di revoca. Predefinito: <code>/sgc/oauth2/revoke</code>

Eventi

Evento	Descrizione
<code>OnOAuth2AfterRevokeToken</code>	Attivato dopo un tentativo di revoca del token. Fornisce il valore del token, il <code>token_type_hint</code> e un parametro <code>Revoked</code> che indica se il token è stato invalidato con successo. Utile per registrare l'attività di revoca.

Esempio

```
var OAuth2Server = new TsgcHTTP_OAuth2_Server();
OAuth2Server.OAuth2Options.Revocation.Enabled = true;

OAuth2Server.Apps.AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2Code });

Server.Authentication.Enabled = true;
Server.Authentication.OAuth.OAuth2 = OAuth2Server;

OAuth2Server.OnOAuth2AfterRevokeToken += (sender, token, tokenTypeHint, ref revoked) =>
{
    Console.WriteLine("Token revoked: " + token + " (type: " + tokenTypeHint + ")");
    revoked = true;
};
```

Esempio di richiesta client

Un client revoca un token inviando una richiesta POST all'endpoint di revoca:

```
// Client-side: revoke an access token
HttpClient.Post("https://example.com/sgc/oauth2/revoke",
    "token=eyJhbGciOiI...&token_type_hint=access_token");
```

OAuth2 Server | Token Introspection (RFC 7662)

Panoramica

L'introspezione dei token consente ai server delle risorse di interrogare il [TsgcHTTP_OAuth2_Server](#) per determinare lo stato attivo e i metadati di un token di accesso o di aggiornamento. Questo è utile quando un server delle risorse deve convalidare un token senza doverlo decodificare localmente.

Endpoint

POST a `/sgc/oauth2/introspect` con i seguenti parametri:

Parametro	Descrizione
<code>token</code>	Il token da esaminare (obbligatorio).
<code>token_type_hint</code>	Suggerimento facoltativo sul tipo di token: <code>access_token</code> o <code>refresh_token</code> .

Risposta

L'endpoint di introspezione restituisce un oggetto JSON con i seguenti campi:

Campo	Descrizione
<code>active</code>	Booleano. True se il token è attualmente attivo (valido e non scaduto né revocato).
<code>scope</code>	Lo scope associato al token.
<code>client_id</code>	L'identificatore client dell'applicazione che ha richiesto il token.
<code>token_type</code>	Il tipo del token (es. Bearer).
<code>exp</code>	Il tempo di scadenza del token (Unix timestamp).
<code>username</code>	Il nome utente associato al token (se applicabile).

Se il token non è valido, è scaduto o è stato revocato, la risposta contiene solo `{"active": false}`.

Configurazione

Proprietà	Descrizione
<code>OAuth2Options.Introspection.Enabled</code>	Impostare su True per abilitare l'endpoint di introspezione. Predefinito: False.

`OAuth2Options.Introspection.URL`

Il percorso URL dell'endpoint di introspezione. Valore predefinito: `/sgc/oauth2/introspect`

Eventi

Evento	Descrizione
<code>OnOAuth2AfterIntrospectToken</code>	Attivato dopo una richiesta di introspezione del token. Fornisce il valore del token e un parametro <code>IsActive</code> che può essere modificato per sovrascrivere il risultato di validazione predefinito. Utile per logiche di validazione personalizzate e per l'auditing.

Esempio

```
var OAuth2Server = new TsgcHTTP_OAuth2_Server();
OAuth2Server.OAuth2Options.Introspection.Enabled = true;
OAuth2Server.Apps.AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2Code });
Server.Authentication.Enabled = true;
Server.Authentication.OAuth.OAuth2 = OAuth2Server;

OAuth2Server.OnOAuth2AfterIntrospectToken += (sender, token, ref isActive) =>
{
    Console.WriteLine("Token introspected: " + token + " Active: " + isActive);
};
```

Esempio di richiesta al Resource Server

Un resource server valida un token inviando una richiesta POST all'endpoint di introspezione:

```
// Resource server: introspect a token
string response = HTTPClient.Post("https://example.com/sgc/oauth2/introspect",
    "token=eyJhbGciOiI...&token_type_hint=access_token");
// Response: {"active":true,"scope":"read write","client_id":"my-client-id",...}
```

OAuth2 Server | Validazione DPoP (RFC 9449)

Panoramica

DPoP (Demonstrating Proof of Possession) per [RFC 9449](#) consente a [TsgcHTTP_OAuth2_Server](#) di richiedere token vincolati al mittente. Quando DPoP è abilitato, il server convalida i JWT di prova DPoP, vincola i token all'impronta JWK del client e restituisce `token_type: DPoP` anziché `Bearer`.

Questo garantisce che i token di accesso possano essere utilizzati solo dal client che li ha ottenuti originariamente, prevenendo il furto e gli attacchi di replay dei token.

Come Funziona

1. Il client include un'intestazione `DPoP` con una prova JWT firmata nelle richieste di token.
2. Il server convalida il proof JWT:
 - `typ` deve essere `dpop+jwt`
 - L'algoritmo (`alg`) deve essere asimmetrico (ad es., ES256, RS256)
 - `htm` (metodo HTTP) e `htu` (URI HTTP) devono corrispondere alla richiesta
 - `iat` (issued at) deve essere recente (controllo di freschezza)
 - La firma viene verificata rispetto alla JWK incorporata nell'header del proof
3. Il server estrae il thumbprint JWK e lo associa al token emesso.
4. La risposta del token include `token_type: DPoP` invece di `Bearer`.
5. Nelle successive richieste di risorse, il server convalida la prova DPoP incluso il claim `ath` (hash del token di accesso) per verificare che la prova sia vincolata al token corretto.

Configurazione

Proprietà	Descrizione
<code>OAuth2Options.DPoP</code>	Boolean. Imposti a True per abilitare la validazione dei token vincolati DPoP. Predefinito: False.

Verifica della firma

Il server ricostruisce la chiave pubblica EC o RSA dal JWK incluso nell'intestazione della prova DPoP e verifica la firma del JWT. Vengono accettati solo algoritmi asimmetrici (ES256, ES384, ES512, RS256, RS384, RS512, PS256, PS384, PS512). Gli algoritmi simmetrici (HS256, ecc.) vengono rifiutati.

Supporto Nonce

Il server può emettere un'intestazione `DPoP-Nonce` nelle risposte token. Quando presente, il client deve includere il valore del nonce nel claim `nonce` dei successivi JWT di prova DPoP. Ciò fornisce un ulteriore livello di protezione contro il replay.

Eventi

Evento	Descrizione

OnOAuth2ValidateDPoP

Generato al termine della validazione DPoP standard. Consente una logica di validazione personalizzata, come il controllo di claim aggiuntivi o l'applicazione di criteri. Impostare **Valid** su False per rifiutare la prova.

Esempio

```
var OAuth2Server = new TsgcHTTP_OAuth2_Server();
OAuth2Server.OAuth2Options.DPoP = true;
OAuth2Server.Apps.AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2Code });
Server.Authentication.Enabled = true;
Server.Authentication.OAuth.OAuth2 = OAuth2Server;
OAuth2Server.OnOAuth2ValidateDPoP += (sender, dpopProof, accessToken, method, url, ref valid) =>
{
    Console.WriteLine("DPoP proof validated for: " + method + " " + url);
    valid = true;
};
```

Rifiuto Proof Non Valida

Quando la convalida della prova DPoP fallisce, il server restituisce HTTP 400 con una risposta di errore:

```
{"error": "invalid_dpop_proof", "error_description": "..."}

```

I motivi comuni di rifiuto includono:

- Intestazione DPoP mancante o malformata.
- Firma JWT non valida.
- Algoritmo simmetrico utilizzato (sono consentiti solo algoritmi asimmetrici).
- Claim `htm` o `htu` non corrispondenti.
- Claim `iat` scaduto o futuro.
- `ath` (access token hash) non corrispondente nelle richieste di risorse.
- `nonce` non valido o mancante quando richiesto dal server.

TsgcHTTP_OAuth2_Server_Provider

Questo componente Le consente di integrare provider OAuth2 esterni (come Azure AD, Google, Facebook).

Introduzione

Questo componente consente di integrare provider OAuth2 esterni (come Azure AD, Google, Facebook...) nel componente server (come un server HTTP), in modo che un utente possa effettuare l'accesso utilizzando le credenziali di Azure AD e, se l'autenticazione ha esito positivo, il server HTTP possa fornire accesso alle risorse protette.

I componenti server dispongono di una proprietà denominata `Authorization.OAuth.OAuth2Provider` a cui è possibile assegnare un'istanza di `TsgcHTTP_OAuth2_Server_Provider`; se `Authentication` è abilitato e la proprietà `OAuth2Provider` è collegata al componente `OAuth2 Provider Server`, le richieste `WebSocket` e `HTTP` richiedono un `Cookie/Bearer Token` per essere elaborate, altrimenti la connessione viene chiusa automaticamente.

```
OAuth2Provider = new TsgcHTTP_OAuth2_ServerProvider();
Server.Authentication.Enabled = true;
Server.Authentication.OAuth.OAuth2Provider = OAuth2Provider;
```

Registra Provider OAuth2

Prima di avviare il server, è necessario configurare i Provider OAuth2 che il server utilizzerà per l'autenticazione. Utilizzare il metodo **RegisterProvider** per configurare i Provider OAuth2; questo metodo ha i seguenti parametri:

- **Name:** è il nome del provider, può essere qualsiasi nome, serve solo a identificare il provider in seguito.
- **ClientId:** è il Client Id pubblico, questo valore viene fornito dal Provider OAuth2.
- **ClientSecret:** è il client secret privato (deve essere mantenuto riservato), questo valore è fornito dal Provider OAuth2.
- **AuthorizeURL:** è l'URL verso cui il client OAuth2 eseguirà il reindirizzamento per il login (la connessione utilizza un browser web).
- **TokenURL:** è l'URL che il server utilizzerà per convalidare il token fornito dopo un'autorizzazione riuscita (la connessione è da server a server).
- **Scope:** è il valore degli ambiti/scope.
- **URL:** è l'URL del server HTTP che verrà utilizzato per reindirizzare all'URL di autorizzazione.
- **CallbackURL:** è l'URL configurato nel provider OAuth2 che elaborerà la risposta inviata dal server OAuth2 dopo un'autorizzazione riuscita.

Esempio: per configurare Azure AD, è necessario un `tenant-id` che viene aggiunto agli URL OAuth2, `ClientId`, `ClientSecret`, `Scope` e un `CallbackURL`.

```
RegisterProvider(
'azure',
'90945b8d-f6b7-4b97-b2bd-21c3c90b5f3x',
'PN67Q~5m06c~X_GMyMf9zMntmm5l2dt~3jVq',
'https://login.microsoftonline.com/a0ca2055-5dd1-467f-bf13-291f6fd715c6/oauth2/v2.0/authorize',
'https://login.microsoftonline.com/a0ca2055-5dd1-467f-bf13-291f6fd715c6/oauth2/v2.0/token',
'user.read',
'/login',
'https://localhost/callback'
);
```

Per eliminare un Provider esistente, utilizzare il metodo **UnRegisterProvider**.

Riferimento

- Proprietà — 2 proprietà disponibili sul componente.
- Metodi — 15 metodi pubblici.
- Events — 4 eventi gestibili.

Guide

- **QuickStart**
 - [OAuth2 Provider Azure AD](#)
- **Autenticare**
 - [Endpoint privati del provider OAuth2](#)
 - [Autenticazione del provider OAuth2](#)
 - [Richieste del provider OAuth2](#)

Provider OAuth2 | Azure AD

Azure AD utilizza i seguenti URL di autorizzazione OAuth2

Autorizzazione: `https://login.microsoftonline.com/<tenant-id>/oauth2/v2.0/authorize`

Token: `https://login.microsoftonline.com/<tenant-id>/oauth2/v2.0/token`

Il **<tenant-id>** deve essere sostituito con i propri valori.

Durante la creazione della configurazione OAuth2, è necessario configurare un URL di callback del server; questo URL verrà utilizzato da Azure per inviare una risposta al server dopo un'autorizzazione riuscita.

Esempio: trovare di seguito un semplice esempio di come registrare il provider Azure AD.

Valori forniti da Azure AD

ClientId: 90945b8d-f6b7-4b97-b2bd-21c3c90b5f3x
ClientSecret: PN67Q~5m06c~X_GMyMf9zMntmm5l2dt~3jVq
tenant: a0ca2055-5dd1-467f-bf13-291f6fd715c6
scope: user.read
CallbackURL: `https://localhost/callback`

Come Registrare Azure AD

```
RegisterProvider(  
    'azure',  
    '90945b8d-f6b7-4b97-b2bd-21c3c90b5f3x',  
    'PN67Q~5m06c~X_GMyMf9zMntmm5l2dt~3jVq',  
    'https://login.microsoftonline.com/a0ca2055-5dd1-467f-bf13-291f6fd715c6/oauth2/v2.0/authorize',  
    'https://login.microsoftonline.com/a0ca2055-5dd1-467f-bf13-291f6fd715c6/oauth2/v2.0/token',  
    'user.read',  
    '/login',  
    'https://localhost/callback'  
);
```

OAuth2 Provider | Private Endpoints

Ogni volta che il Server riceve una richiesta HTTP, viene chiamato l'evento **OnOAuth2IsPrivateEndpoint** per chiedere se l'Endpoint è privato o meno. Per impostazione predefinita, non è privato.

```
void OnOAuth2IsPrivateEndpoint(TObject *ender, const string aEndpoint, ref bool IsPrivate)
{
    if (aEndpoint == "/private")
    {
        IsPrivate = True;
    }
}
```

Provider OAuth2 | Autenticazione

Il componente OAuth2 Provider Server consente di autenticarsi utilizzando un provider OAuth2 esterno (come Azure AD o Google) per accedere alle risorse protette del Suo server. **Esempio:** può configurare il Suo server HTTP per consentire agli utenti di accedere utilizzando le credenziali Azure; se l'accesso ha esito positivo, tali utenti potranno accedere alle risorse protette del Suo server.

Il processo di autenticazione avviene lato server e i token OAuth2 non vengono condivisi con i client. Ciò significa che quando l'utente effettua l'accesso tramite Azure ad esempio, se l'autenticazione ha esito positivo, Azure restituisce un Access Token che consente di inviare richieste al server Azure per ottenere alcune informazioni (a seconda dello scope) sul profilo utente, email ecc. Questo Access Token **NON VIENE CONDIVISO** con il client (ad esempio un browser web). Invece di restituire l'Access Token al client, il server crea un ID casuale collegato internamente all'Access Token, in modo che ogni volta che il Client (browser web) vuole effettuare una chiamata al server OAuth2, utilizzi l'ID pubblico e il server usi questo ID per ottenere l'Access Token OAuth2 e fare da proxy alle richieste HTTP.

Di seguito è riportato un esempio di come funziona l'autenticazione OAuth2. L'esempio utilizzerà la configurazione Azure AD descritta nel seguente link [OAuth2 Provider Azure AD](#).

Avvia il Server

Il server inizia l'ascolto su localhost e sulla porta 443. Il server HTTP di sgcWebSockets è collegato al componente OAuth2 Server Provider e la proprietà Authentication è abilitata.

Prima che il server venga avviato, il provider Azure OAuth2 viene registrato tramite la seguente chiamata al metodo.

```
RegisterProvider(
'azure',
'90945b8d-f6b7-4b97-b2bd-21c3c90b5f3x',
'PN67Q~5m06c~X_GMyMf9zMntmm5l2dt~3jVq',
'https://login.microsoftonline.com/a0ca2055-5dd1-467f-bf13-291f6fd715c6/oauth2/v2.0/authorize',
'https://login.microsoftonline.com/a0ca2055-5dd1-467f-bf13-291f6fd715c6/oauth2/v2.0/token',
'user.read',
'/login',
'https://localhost/callback'
);
```

Accessi utente

L'utente apre un nuovo browser web e accede all'endpoint '/login'.

Il server rileva che l'endpoint '/login' viene utilizzato per accedere tramite il provider Azure, quindi reindirizza a

<https://login.microsoftonline.com/a0ca2055-5dd1-467f-bf13-291f6fd715c6/oauth2/v2.0/authorize>

E il flusso di autenticazione OAuth2 ha inizio.

Autenticazione OAuth2

L'utente viene reindirizzato all'Authentication Endpoint del server OAuth2 e ora deve effettuare l'accesso utilizzando le credenziali e accettare i termini dell'applicazione OAuth2.

Se l'autorizzazione ha esito positivo, Azure AD invia un Codice all'url

<https://localhost/callback>

Convalida il codice OAuth2

Ora, il server ha ricevuto un codice da Azure ed effettuerà una connessione interna ad Azure (da server a server) per verificare che questo token sia corretto (ed evitare che qualcuno stia cercando di violare il server).

Il server si connette a

`https://login.microsoftonline.com/a0ca2055-5dd1-467f-bf13-291f6fd715c6/oauth2/v2.0/token`

Passando alcuni parametri come il codice ricevuto e il clientsecret, se la validazione ha esito positivo, Azure restituisce il Token di Accesso che può essere utilizzato per accedere alle Risorse Protette Azure come leggere il profilo, l'email...

Token di Accesso Valido

Quando il server riceve un AccessToken valido, viene chiamato l'evento **OnOAuth2ProviderTokenValid**; qui è possibile configurare come viene memorizzato l'AccessToken (se viene memorizzato) accedendo alla classe di parametri `TsgcHTTPOAuth2ProviderToken`.

AccessToken: è il Token OAuth2 restituito da Azure

ID: è l'identificatore pubblico memorizzato come cookie.

In questo evento è possibile configurare le azioni da eseguire dopo una autenticazione riuscita; ad esempio: se si desidera reindirizzare l'utente all'URL privato, utilizzare quanto segue

`Response.Redirect.URL := 'https://localhost/private';`

Invia Richieste ad Azure

Ora è possibile inviare richieste al server Azure utilizzando l'ID pubblico memorizzato come cookie.

Esempio: se si desidera leggere i dati del profilo, utilizzare il seguente metodo.

`Get('ID', 'https://graph.microsoft.com/v1.0/me');`

Dove ID è l'identificatore pubblico.

OAuth2 Provider | Richieste

Una volta completata con successo l'autenticazione, è possibile inviare richieste al server protetto OAuth2 utilizzando il token ID pubblico memorizzato come cookie.

Il componente server provider OAuth2 dispone di diversi metodi per inviare richieste HTTP: GET, POST, DELETE...

È possibile passare il token come parametro oppure passare la classe RequestInfo se si utilizzano i componenti server Indy.

```
void OnCommandGet(TidHTTPRequestInfo ARequestInfo, TidHTTPResponseInfo AResponseInfo)
{
    if (ARequestInfo.Document == "/private"
    {
        // return OAuth2 profile data
        AResponseInfo.ContentText = OAuth2Provider.Get("ID Token", "https://graph.microsoft.com/v1.0/me");
        AResponseInfo.ContentType = "application/json";
        AResponseInfo.ResponseNo = 200;
    }
    else
    {
        AResponseInfo.ResponseNo = 404;
    }
}
```

HTTP | JWT

JWT (JSON Web Token) è tipicamente composto da intestazione + payload + firma.

Intestazione

Contiene le informazioni sui metadati del JWT.

- **alg**: l'algoritmo utilizzato per firmare il token.
- **typ**: il tipo del token, sempre JWT.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Potrebbero essere presenti ulteriori intestazioni, ma quelle precedenti saranno sempre presenti.

Payload

Il payload contiene i claim del JWT. Gli header standard sono i seguenti:

- **iss**: l'emittente, l'entità che genera e rilascia il JWT.
- **sub**: il soggetto, l'entità identificata da questo token.
- **aud**: il destinatario, il pubblico di destinazione per questo JWT.
- **exp**: la scadenza, il timestamp in formato UNIX dopo il quale il token non deve essere accettato.
- **iat**: issued at, specifica la data in cui il token è stato emesso.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

Signature

La firma viene creata utilizzando l'Header Codificato, il Payload Codificato, un Segreto e un Algoritmo Crittografico.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4uRG9lIiwiaWF0IjoxNTE2MjM5MjM5Lm50

DIyf0.Sf1KxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_ad0ssw5c

Algoritmi supportati

I seguenti algoritmi sono supportati sia dal componente JWT Client che Server.

- HS256
- HS384
- HS512
- RS256
- RS384
- RS512
- ES256
- ES384
- ES512

Le librerie **OpenSSL** sono **richieste** per firmare e verificare il JWT.

Componenti

- **TsgcHTTP_JWT_Client**: Client JWT che consente di codificare e firmare JWT e di inviarli come intestazione Authorization nei protocolli **HTTP** e **WebSocket**.
- **TsgcHTTP_JWT_Server**: Server JWT che consente di decodificare e validare i JWT ricevuti come Authorization Header nei protocolli **HTTP** e **WebSocket**.

TsgcHTTP_JWT_Client

Configuri quali librerie openSSL utilizzerà quando si usa il client JWT.

Introduzione

Il componente TsgcHTTP_JWT_Client consente di codificare e firmare token JWT, collegato a un [client WebSocket](#) o a un [client HTTP/2](#); il token verrà inviato automaticamente come Authorization Bearer Token Header.

Configurazione

È possibile configurare i valori JWT nelle proprietà **JWTOptions**, ci sono 2 proprietà principali: **Header** e **Payload**; impostare i valori per ogni proprietà richiesta.

Se la firma è crittografata con una chiave privata (algoritmi RS ed ES), impostare il valore nella proprietà **Private-Key** dell'algoritmo.

Se la firma è cifrata utilizzando un Secret (algoritmi HS), impostare il valore nella proprietà **Secret** dell'Algorithm.

Opzioni OpenSSL

Configuri quali librerie openSSL utilizzerà quando si usa il client JWT.

OpenSSL_Options: configurazione delle librerie openSSL.

APIVersion: consente di definire quale API OpenSSL verrà utilizzata.

osIAPI_1_0: utilizza l'API OpenSSL 1.0, l'ultima supportata da Indy

osIAPI_1_1: utilizza le API OpenSSL 1.1, richiede la nostra libreria Indy personalizzata e consente di utilizzare le librerie OpenSSL 1.1.1 (con supporto TLS 1.3).

osIAPI_3_0: utilizza OpenSSL API 3.0, richiede la nostra libreria Indy personalizzata e consente l'uso delle librerie OpenSSL 3.0.0 (con supporto TLS 1.3).

LibPath: qui è possibile configurare la posizione delle librerie openSSL

oslpNone: questa è l'impostazione predefinita; le librerie openSSL devono trovarsi nella stessa cartella del binario o in un percorso noto.

oslpDefaultFolder: imposta automaticamente il percorso openSSL dove le librerie devono essere situate per tutte le personalità IDE.

oslpCustomFolder: se questa è l'opzione selezionata, definire il percorso completo nella proprietà LibPathCustom.

LibPathCustom: quando LibPath = oslpCustomFolder, definire qui il percorso completo dove si trovano le librerie openSSL.

UnixSymLinks: abilita o disabilita il caricamento dei SymLink sui sistemi Unix (per impostazione predefinita è abilitato, tranne che su OSX64):

oslsSymLinksDefault: per impostazione predefinita sono abilitati, eccetto su OSX64 (dopo MacOS Monterey si verifica un errore nel tentativo di caricare la libreria senza versione.).

oslsSymLinksLoadFirst: Carica i SymLink e li elabora prima di tentare il caricamento delle librerie di versione.

oslsSymLinksLoad: Carica i SymLink dopo aver tentato di caricare le librerie di versione.

oslsSymLinksDontLoad: non caricare i SymLink.

Intestazioni Personalizzate

Le proprietà Header e Payload contengono le intestazioni più comuni utilizzate per generare un JWT, ma è possibile aggiungere altre intestazioni chiamando il metodo **AddKeyValue** e passando Key e Value come parametri.

Esempio: se si desidera aggiungere un nuovo record nell'intestazione JWT con il proprio nome, utilizzare il seguente metodo

```
Header.AddKeyValue("name", "John Smith");
```

Dopo aver configurato le proprietà, per generare il JWT è sufficiente chiamare il metodo **Sign** che restituirà il valore del JWT.

Client WebSocket e JWT

[TsgcWebSocketClient](#) consente l'uso di JWT durante la connessione ai server WebSocket; è sufficiente creare un nuovo client JWT e assegnarlo alla proprietà **Authentication.Token.JWT** property.

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
oClient.URL = "ws://www.esegece.com:2052";

TsgcHTTP_JWT_Client oJWT = new TsgcHTTP_JWT_Client();
oJWT.JWTOptions.Header.alg = jwtRS256;
oJWT.JWTOptions.Payload.sub = "1234567890";
oJWT.JWTOptions.Payload.iat = 1516239022;

oClient.Authentication->Enabled = true;
oClient.Authentication.URL.Enabled = false;
oClient.Authentication.Token.Enabled = true;
oClient.Authentication.Token.JWT = oJWT;
oClient.Active = true;
```

Client HTTP e JWT

[TsgcHTTP2Client](#) e [TsgcHTTP1Client](#) consentono l'utilizzo di JWT durante la connessione ai server HTTP/2; è sufficiente creare un nuovo client JWT e assegnarlo alla proprietà **Authentication.Token.JWT** property.

```
TsgcHTTP2Client oHTTP = new TsgcHTTP2Client();

TsgcHTTP_JWT_Client oJWT = new TsgcHTTP_JWT_Client();
oJWT.JWTOptions.Header.alg = jwtRS256;
oJWT.JWTOptions.Payload.sub = "1234567890";
oJWT.JWTOptions.Payload.iat = 1516239022;

oHTTP.Authentication.Token.JWT = oJWT;
oHTTP.Get("https://your.api.com");
```

Scadenza

L'Authorization Token può essere **ricreato ogni volta** che si invia una richiesta HTTP utilizzando un client HTTP oppure può essere **riutilizzato più volte fino alla sua scadenza**.

Esempio: la chiamata ad Apple APNs tramite Token richiede che il token venga riutilizzato per almeno 20 minuti e per un massimo di 1 ora. Utilizzare la proprietà **RefreshTokenAfter** per impostare i secondi dopo i quali il token scadrà, ad esempio dopo 30 minuti.

```
RefreshTokenAfter = 60 * 40.
```

Crea firma JWT

È possibile **creare Firme JWT manualmente** da utilizzare in applicazioni che non fanno uso del protocollo Web-Socket o HTTP, o se si utilizzano componenti di applicazioni di terze parti e si ha bisogno solo del Token JWT.

Per ottenere la firma JWT, è sufficiente creare una nuova istanza del client JWT e compilare manualmente le proprietà; quando tutte le proprietà sono impostate, chiamare il metodo **Sign** che restituirà il Token JWT.

```
TsgcHTTP_JWT_Client oJWT = new TsgcHTTP_JWT_Client();  
// ... header  
oJWT.JWTOptions.Header.alg = jwtHS256;  
oJWT.JWTOptions.Algorithms.HS.Secret = "79F66F1E-E998-436B-8A0A-3E5DEFA4FD9E";  
// ... payload  
oJWT.JWTOptions.Payload.jti = "9B66FB94-B761-42B1-A1AF-3C44233DBE87";  
oJWT.JWTOptions.Payload.iat = 1630925658;  
oJWT.JWTOptions.Payload.iss = "2886EC7547B7BA6A9009";  
oJWT.JWTOptions.Payload.exp = 1630933158;  
// ... custom payload values  
oJWT.JWTOptions.Payload.ClearKeyValues;  
oJWT.JWTOptions.Payload.AddKeyValue("origin", "www.yourwebsite.com");  
oJWT.JWTOptions.Payload.AddKeyValue("ip", "69.39.46.178");  
// ... get JWT Token  
MessageBox.Show(oJWT.Sign());
```

Riferimento

- Proprietà — 2 proprietà disponibili sul componente.
- Metodi — 2 metodi pubblici.
- Eventi — 0 eventi gestibili.

TsgcHTTP_JWT_Server

Componente *TsgcHTTP_JWT_Server*.

Introduzione

Il componente *TsgcHTTP_JWT_Server* consente di **decodificare** e **validare** i token JWT ricevuti nell'**handshake WebSocket** quando si utilizza il protocollo WebSocket, oppure come **HTTP Header** quando si utilizza il protocollo HTTP.

Configurazione

È possibile configurare le seguenti proprietà nella proprietà **JWTOptions** del componente:

Se la firma viene validata utilizzando una chiave pubblica (algoritmi RS ed ES), impostare il valore nella proprietà **PublicKey** dell'algoritmo.

Se la firma viene convalidata utilizzando un segreto (algoritmi HS), impostare il valore nella proprietà **Secret** dell'algoritmo.

Per validare i token JWT, è sufficiente **collegare un'istanza di *TsgcHTTP_JWT_Server*** alla proprietà **Authentication.JWT.JWT** del server WebSocket/HTTP.

```
TsgcWebSocketHTTPServer oServer = new TsgcWebSocketHTTPServer();
oServer.Port = 80;
TsgcHTTP_JWT_Server oJWT = new TsgcHTTP_JWT_Server();
oJWT.JWTOptions.Algorithms.RS.PublicKey = "public key here";
oServer.Authorization.Enabled = true;
oServer.Authorization.JWT.JWT = oJWT;
oServer.Active = true;
```

Checks la proprietà consente di abilitare alcune verifiche nel Payload del JWT; per impostazione predefinita controlla che le date di emissione siano valide.

Riferimento

- Proprietà — 2 proprietà disponibili nel componente.
- Metodi — 3 metodi pubblici.
- Events — 7 eventi che è possibile gestire.

WebAuthn

WebAuthn (Web Authentication) è uno standard web sviluppato dal World Wide Web Consortium (W3C) e dalla FIDO Alliance per consentire un'autenticazione sicura, senza password sul web. Fa parte del più ampio framework FIDO2 e mira a ridurre la dipendenza dalle password tradizionali, che sono spesso vulnerabili al phishing, al credential stuffing e ad altri attacchi.

Alla base, **WebAuthn consente agli utenti di autenticarsi utilizzando la crittografia a chiave pubblica. Invece di un nome utente e una password, gli utenti registrano una coppia di chiavi pubblica-privata univoca con un'applicazione web** (la Relying Party). La chiave privata è archiviata in modo sicuro su un autenticatore — come una chiave di sicurezza hardware, uno smartphone o un dispositivo biometrico integrato — mentre la chiave pubblica è archiviata sul server.

Durante l'autenticazione, il server emette una sfida che deve essere firmata dalla chiave privata dell'utente. La sfida firmata viene restituita e verificata tramite la chiave pubblica memorizzata, garantendo l'integrità e l'origine della risposta. Questo approccio impedisce l'intercettazione o il riutilizzo delle credenziali.

WebAuthn supporta una gamma di autenticator e dispositivi, rendendolo flessibile sia per gli sviluppatori che per gli utenti. Consente inoltre l'autenticazione a più fattori (MFA) se combinato con altri fattori come PIN o dati biometrici, migliorando significativamente la sicurezza senza sacrificare la facilità d'uso.

Server

- **TsgcWSAPIServer_WebAuthn:** Il componente fornisce una soluzione semplice ma potente per implementare il server Relying Party WebAuthn, abilitando l'autenticazione senza password nella propria applicazione web. Il server ha superato il **Test di Conformità FIDO** utilizzando lo strumento di test di auto-validazione della conformità.

Client

- **Client Javascript:** funge da ponte tra l'utente, il browser e l'autenticatore durante la registrazione e l'autenticazione

TsgcWSAPIServer_WebAuthn

Solo OpenSSL 3.

Introduzione

Il componente **TsgcWSAPIServer_WebAuthn** fornisce una soluzione semplice ma potente per implementare il server Relying Party WebAuthn, abilitando l'autenticazione senza password nella propria applicazione web. Un'applicazione WebAuthn è composta da un server WebAuthn che gestisce la registrazione e l'autenticazione lato server e da un'applicazione lato client che di solito è un'applicazione javascript.

WebAuthn richiede l'uso di connessioni sicure (SSL/TLS), quindi le librerie [OpenSSL](#) devono essere distribuite e configurate con il server.

È supportata solo l'API OpenSSL 3.0.0+, pertanto le versioni precedenti di OpenSSL potrebbero non funzionare.

Configurazione

Il **TsgcWSAPIServer_WebAuthn** deve essere collegato a un server HTTP, [TsgcWebSocketHTTP_Server](#) o [TsgcWebSocketServer_HTTPAPI](#) utilizzando la proprietà `Server`. Può configurare gli endpoint del server che gestiranno le opzioni di registrazione e autenticazione, e le opzioni WebAuthn come gli algoritmi supportati, le origini e altro.

Opzioni degli endpoint

Qui è possibile configurare gli endpoint del server che gestiranno le richieste HTTP/JavaScript per utilizzare WebAuthn come autenticatore. Il componente è già configurato con endpoint predefiniti, ma è possibile modificarli tutti per adattarli alle proprie esigenze.

- **AuthenticationOptions:** per impostazione predefinita è `/sgcWebAuthn/Authentication/Options`
- **AuthenticationVerify:** per impostazione predefinita è `/sgcWebAuthn/Authentication/Verify`
- **RegistrationOptions:** per impostazione predefinita è `/sgcWebAuthn/Registration/Options`
- **RegistrationVerify:** per impostazione predefinita è `/sgcWebAuthn/Registration/Verify`
- **Webauthn:** include la libreria javascript utilizzata per impostazione predefinita. È possibile disabilitare questa proprietà e utilizzare la propria libreria webauthn.
- **Test:** per impostazione predefinita è disabilitato; usarlo solo per testare la funzionalità WebAuthn.

Esempio: se il server è in ascolto sul dominio `www.test.com`, la richiesta alle opzioni di autenticazione per impostazione predefinita sarà `http://www.test.com/sgcWebAuthn/Authentication/Options`

Opzioni WebAuthn

In questa proprietà può configurare le opzioni principali del componente WebAuthn Server.

- **RelyingParty:** una proprietà **obbligatoria** in cui deve essere definito il **nome DNS del server**. Esempio: se il server è in esecuzione sul dominio `www.test.com`, impostare questa proprietà su `"www.test.com"`.

WebAuthn utilizza le origini per applicare i vincoli della politica same-origin, essenziali per prevenire phishing e attacchi cross-site. Durante i processi di registrazione e autenticazione WebAuthn, l'origine viene rigorosamente validata dal browser e dall'autenticatore.

- **Origins:** Se le richieste possono provenire da origini diverse, utilizzare la proprietà `Origin` per impostare le origini aggiuntive. Esempio: se le richieste possono provenire da `login.test.com` e `www.test.co.uk`, configurare la proprietà `Origins` con i valori: `https://login.test.com` e `https://www.test.co.uk`
- **TopOrigins:** Normalmente, `WebAuthn` si basa sull'origine del frame chiamante (ovvero quello che richiama `navigator.credentials.create()` o `navigator.credentials.get()`). Tuttavia, le pagine web possono essere incorporate in `iframe`, che potrebbero provenire da un'origine diversa rispetto alla pagina di primo livello. Questo apre potenziali rischi di abuso o attacchi di tipo `clickjacking`. Per mitigare questo problema, la specifica `WebAuthn Level 2` introduce `TopOrigin`, dove è possibile definire i `TopOrigins`.

In `WebAuthn`, `crossOrigin` è un parametro booleano che indica se l'operazione `WebAuthn` viene eseguita da un contesto `cross-origin`, ad esempio un `iframe` incorporato da un'origine diversa rispetto al contesto di navigazione di primo livello.

Questo parametro è stato introdotto per aiutare browser e autenticatori a gestire in modo sicuro le richieste di autenticazione in ambienti incorporati, uno scenario comune nelle moderne applicazioni web.

- **AllowCrossOrigins:** se `true`, indica che le richieste provenienti da un `iframe cross-origin` (ad esempio un `iframe` su `https://auth.example.com` incorporato in una pagina su `https://app.example.org`) sono consentite. Per impostazione predefinita è disabilitato.

`WebAuthn` supporta una varietà di **algoritmi crittografici** per la generazione e la verifica delle credenziali a chiave pubblica. Questi algoritmi vengono utilizzati durante la registrazione delle credenziali (con `navigator.credentials.create()`) e l'autenticazione (con `navigator.credentials.get()`), e garantiscono la firma e la validazione sicure delle challenge utilizzando coppie di chiavi asimmetriche. Il server è configurato per impostazione predefinita con **ES256 e RS256**, che sono gli algoritmi più comuni. È possibile modificare in qualsiasi momento quali algoritmi sono supportati dalla proprietà **Algorithms**. Sono supportati i seguenti algoritmi:

- ES256
- ES384
- ES512
- RS256
- RS384
- RS512
- PS256
- PS384
- PS512
- RS1
- EdDSA

In `WebAuthn`, **l'attestazione è un meccanismo opzionale** che consente all'autenticatore (ad esempio dispositivo o chiave di sicurezza) di fornire informazioni sul proprio produttore, modello e caratteristiche di sicurezza durante la creazione delle credenziali. Queste informazioni aiutano la `Relying Party (RP)` a decidere se fidarsi dell'autenticatore.

Diversi formati di attestazione definiscono come questi dati sono strutturati e verificati. Tre formati comunemente utilizzati sono `android-key`, `packed` e altri come `fido-u2f`, `apple` o `none`. Per impostazione predefinita, tutti i formati di attestazione sono abilitati. Di seguito è riportato l'elenco dei formati di attestazione supportati:

- **NoneAttestation:** in questo caso non vengono restituiti dati di attestazione. Privilegia la privacy dell'utente evitando l'esposizione degli identificatori del dispositivo. Comune nelle applicazioni che non si preoccupano della provenienza del dispositivo.
- **PackedAttestation:** è un formato flessibile e compatto utilizzato da molti autenticatori. L'autenticatore restituisce un certificato di attestazione e una firma. Può essere: **Full attestation:** Firmato con una chiave e un certificato forniti dal produttore, oppure **Self attestation:** Firmato utilizzando la chiave privata della credenziale. Il più utilizzato su diverse piattaforme (es. `YubiKey`, `Windows Hello`).
- **TPMAttestation:** Utilizzato da dispositivi con un `Trusted Platform Module (TPM)`. L'attestazione è firmata utilizzando le chiavi del `TPM` e include una catena di certificati. Utilizzato da desktop/laptop aziendali con chip `TPM` (ad esempio, macchine `Windows`).
- **AndroidKeyAttestation:** Utilizzato dai dispositivi `Android` con l'`Android Keystore`. La chiave viene generata in hardware e l'attestazione include informazioni firmate da una catena di certificati emessa dal produttore del dispositivo. Utilizzato dai telefoni `Android` con `keystore` basati su hardware (`TEE` o `StrongBox`).

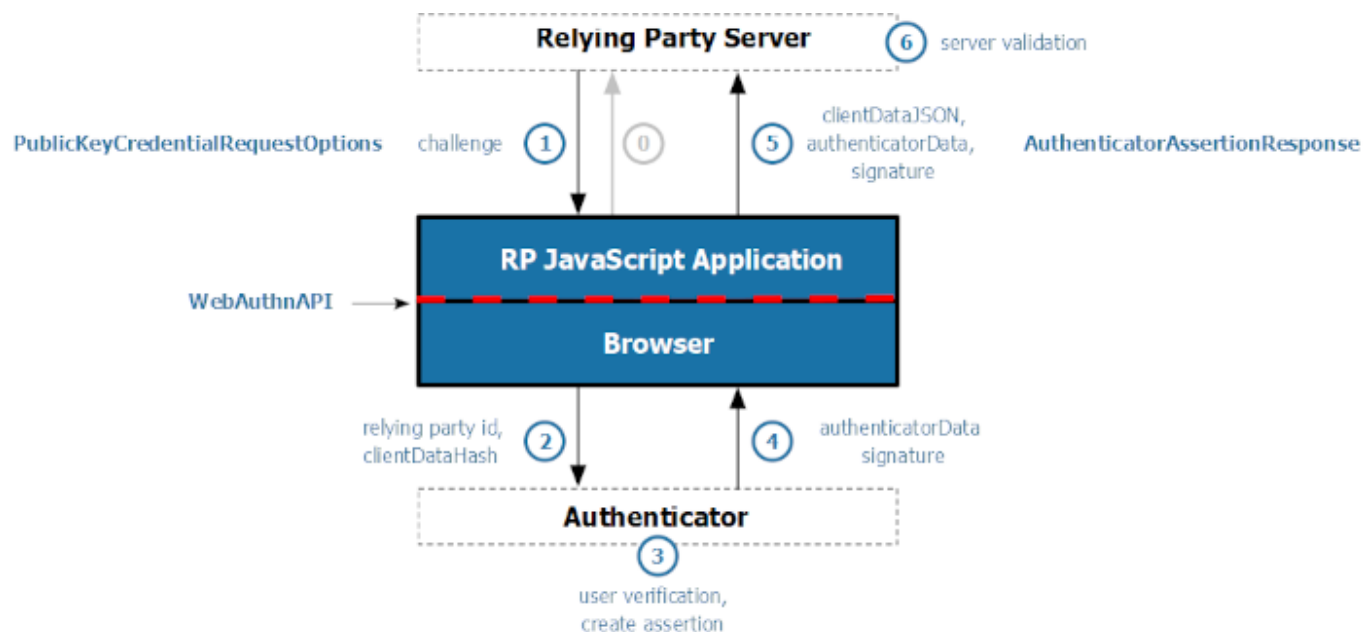
- **AppleAttestation:** Utilizzato dagli autenticatori di piattaforma Apple, come Touch ID e Face ID. L'attestazione viene generata dalle API interne di Apple e include un formato di certificato speciale. Utilizzato su Safari tramite la biometria Apple.
- **FidoU2FAttestation:** Formato di attestazione legacy utilizzato dagli autenticatori FIDO U2F. Restituisce un certificato e una firma compatibili con U2F. Utilizzato da chiavi di sicurezza più vecchie (ad es. i primi Yubi-Key) che supportano FIDO U2F.

Nell'API WebAuthn, **AllowCredentials** è un **campo facoltativo** utilizzato durante il processo di autenticazione (tramite `navigator.credentials.get()`). **Specifica un elenco di ID credenziale autorizzati ad autenticare l'utente** per una determinata Relying Party (RP). Questo meccanismo consente alla RP di controllare quali credenziali sono considerate valide per un tentativo di accesso. La proprietà `credentials` contiene i seguenti campi:

- **AllowCredentials:** se abilitato (false per impostazione predefinita) specifica un elenco di ID credenziali autorizzati ad autenticare l'utente
- **ExcludeCredentials:** dato un nome utente, mostra tutte le credenziali esistenti già memorizzate nel componente server.
- **Limit:** il numero massimo di credenziali che verranno inviate quando `ExcludeCredentials` è true.

Protocollo WebAuthn

- **Registrazione WebAuthn:** Il server genera una sfida e la invia al client, che utilizza un autenticatore (ad esempio una chiave di sicurezza o un dispositivo biometrico) per creare una coppia di chiavi. La chiave pubblica viene inviata e memorizzata dal server per le autenticazioni future. Di seguito sono riportate ulteriori informazioni sugli eventi del flusso di registrazione:
 - [Richiesta di registrazione WebAuthn](#)
 - [Risposta di registrazione WebAuthn](#)
 - [Risultato di registrazione WebAuthn](#)
- **Autenticazione WebAuthn:** Il server invia una sfida al client, che la firma usando la chiave privata precedentemente registrata memorizzata nell'autenticatore. La risposta firmata viene verificata dal server usando la chiave pubblica memorizzata per confermare l'identità dell'utente. Di seguito ulteriori informazioni sugli eventi del flusso di autenticazione:
 - [Richiesta di Autenticazione WebAuthn](#)
 - [Risposta di Autenticazione WebAuthn](#)
 - [Risultato di Autenticazione WebAuthn](#)
- **MDS:** Il FIDO Alliance Metadata Service (MDS) è un repository centralizzato di dichiarazioni di metadati utilizzato dalle parti di riferimento per validare l'attestazione degli autenticatori e verificare l'autenticità del modello del dispositivo.



- **Autorizzazione:** Il client può richiedere un token bearer dal server durante il flusso di autenticazione. Questo token può essere utilizzato in seguito per aprire una nuova connessione WebSocket o HTTP senza dover effettuare nuovamente il login tramite passkey.
 - **Autorizzazione WebAuthn WebSocket**
 - **Autorizzazione WebAuthn HTTP**

Riferimento

- Properties — 4 proprietà disponibili sul componente.
- Metodi — 8 metodi pubblici.
- Eventi — 15 eventi gestibili.

WebAuthn | Registrazione

La **Registrazione WebAuthn** coinvolge il **client** (browser), l'**authenticator** (dispositivo di sicurezza) e il **relying party** (RP; il server delphi/cbuilder TsgcWSAPIServer_WebAuthn).

Opzioni di Registrazione

La creazione di una nuova credenziale utente richiede solitamente che il client avvii il flusso di registrazione utilizzando una nuova richiesta HTTP all'**endpoint Registration Options** configurato nel server TsgcWSAPIServer_WebAuthn. Per impostazione predefinita, l'endpoint è /sgcWebAuthn/Registration/Options, quindi se il Suo server è in ascolto sul dominio https://www.test.com, il client dovrebbe effettuare una nuova richiesta all'url https://www.test.com/sgcWebAuthn/Registration/Options.

Il client invia una nuova richiesta passando come payload il seguente json (usando test come nome utente)

```
{"username":"test","algorithms":[]}
```

Il server legge la richiesta e restituisce una risposta con una nuova sfida.

```
{
  "rp": {
    "name": "localhost",
    "id": "localhost"
  },
  "user": {
    "id": "36b9d6a84204487382fee62e7e67a80d",
    "name": "test",
    "displayName": "test"
  },
  "challenge": "6c6c468c99f24bf29a85a15b661f75f385654f97309c46bab2909c926e17ccbe",
  "pubKeyCredParams": [
    {
      "type": "public-key",
      "alg": "-7"
    },
    {
      "type": "public-key",
      "alg": "-257"
    }
  ],
  "timeout": 60000,
  "excludeCredentials": [],
  "authenticatorSelection": {
    "residentKey": "preferred",
    "requireResidentKey": false,
    "userVerification": "preferred"
  },
  "attestation": "direct",
  "hints": [],
  "extensions": {
    "credProps": true
  }
}
```

La risposta restituisce i seguenti dati:

- **challenge:** Dati binari casuali, codificati in base64.
- **rp:** Informazioni sulla Relying Party (la propria applicazione/sito web).
- **user:** Identificatore utente e nome visualizzato.

- **pubKeyCredParams:** Algoritmi consentiti (es. ES256, RS256).
- **authenticatorSelection:** Preferenze per il tipo di autenticatore (platform o cross-platform, opzioni di verifica dell'utente).
- **timeout:** Valore di timeout suggerito.
- **attestation:** Preferenza di trasmissione dell'attestazione ("none", "direct", "indirect").

Verifica registrazione

Ora il client ha la risposta dal server, legge la risposta e l'autenticator restituisce i dati crittografici al web-browser del client. Ora il client invia una nuova HTTP Request con i seguenti dati al Registration Verify Endpoint configurato nel server TsgcWSAPIServer_WebAuthn, che per impostazione predefinita è /sgcWebAuthn/Registration/Verify, quindi se il Suo server è in ascolto nel dominio <https://www.test.com>, il client dovrebbe effettuare una nuova richiesta all'url <https://www.test.com/sgcWebAuthn/Registration/Verify>.

L'autenticatore risponde al JavaScript nel browser con:

- **id:** ID credenziale (ID univoco per la coppia di chiavi generata dall'autenticatore).
- **rawId:** Forma binaria dell'ID credenziale.
- **type:** Tipo di credenziale ("public-key").
- **response:** Contiene attestationObject e clientDataJSON.

Di seguito è riportato un esempio:

```
{
  "id": "yeA4BVRlrAfLG-KzqsL_rll4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
  "rawId": "yeA4BVRlrAfLG-KzqsL_rll4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
  "response": {
    "attestationObject": "o2NmbXRmcGFja2VkZ2F0dFN0....",
    "clientDataJSON": "eyJ0eXBlljoid2ViYXV0aG4uY3Jl....",
    "transports": [
      "nfc",
      "usb"
    ],
    "publicKeyAlgorithm": -7,
    "publicKey": "MFkwEwYHKoZIzj0CAQYIKoZIzj....",
    "authenticatorData": "SZYN5YgOjGh0NBcPZHgW4_k...."
  },
  "type": "public-key",
  "clientExtensionResults": {
    "credProps": {
      "rk": true
    }
  },
  "authenticatorAttachment": "cross-platform"
}
```

Il server legge la richiesta JSON dal client e decodifica, verifica e memorizza la chiave pubblica e l'ID credenziale.

Decodifica attestationObject e clientDataJSON:

- Estrarre la chiave pubblica dall'attestationObject.
- Verificare che la challenge in clientDataJSON corrisponda a quella originale inviata dal server.
- Assicurarsi di eseguire una corretta validazione dell'origine (myapp.example.com).

Store Credential:

- Se la verifica ha esito positivo, viene chiamato l'evento **OnWebAuthnRegistrationSuccessful** in modo da poter memorizzare in modo sicuro nel database l'ID della credenziale e la chiave pubblica estratta per le autenticazioni future.
- In caso di errore, viene chiamato l'evento **OnWebAuthnRegistrationError** ed è possibile ottenere i dettagli dell'errore.

Flusso di Registrazione

Ottenga maggiori informazioni sul processo di Registration Flow utilizzando i seguenti link:

- [Richiesta di registrazione WebAuthn](#)
- [Risposta di Registrazione WebAuthn](#)
- [Risultato Registrazione WebAuthn](#)

Registrazione WebAuthn | Richiesta

Questa richiesta di opzioni di registrazione è essenziale per avviare in modo sicuro la registrazione WebAuthn. Chiede al server di:

- Genera una sfida,
- Cercare o creare l'ID utente,
- Restituisce un oggetto `PublicKeyCredentialCreationOptions` valido (con RP, utente, algoritmi, ecc.).

Il browser deve ottenere le opzioni di creazione delle credenziali dal server per avviare una registrazione sicura delle credenziali WebAuthn. Questo viene tipicamente fatto tramite una richiesta HTTPS POST dal browser a un endpoint del server (ad esempio, `/sgcWebAuthn/Registration/Options`), che prepara parametri come la challenge, l'RP ID e le informazioni sull'utente.

Il corpo della richiesta contiene dati che identificano l'utente ed eventualmente forniscono preferenze di configurazione. Spesso si presenta così:

```
{
  "username": "alice@example.com",
  "displayName": "Alice Smith",
  "authenticatorSelection": {
    "authenticatorAttachment": "platform", // opzionale: platform o cross-platform
    "userVerification": "preferred" // opzionale: required | preferred | discouraged
  },
  "attestation": "none", // oppure "direct", "indirect"
  "residentKey": "discouraged" // opzionale
}
```

Il componente `Server WebAuthn` rimane in ascolto sull'endpoint configurato nella proprietà **`EndpointOptions.RegistrationOptions`** per la richiesta iniziale del browser di ottenere un `PublicKeyCredentialCreationOptions`. Quando il server riceve una nuova richiesta HTTP, viene chiamato l'evento **`OnWebAuthnRegistrationOptionsRequest`** ed è possibile accedere alla richiesta inviata dal client e annullarla impostando il parametro `Accept` su `False`.

```
public void OnWebAuthnRegistrationOptionsRequest(object sender, TsgcWebAuthn_RegistrationOptions_Request aRequest)
{
    if (aRequest.Username == "anonymous")
        accept = false;
}
```

WebAuthn Registrazione | Risposta

Il server risponde alla richiesta di opzioni di registrazione del client (ad esempio, POST /sgcWebAuthn/Registration/Options) con un payload JSON simile al seguente (dopo la codifica base64url dei campi binari):

```
{
  "publicKey": {
    "rp": {
      "name": "esegece software",
      "id": "esegece.com"
    },
    "user": {
      "id": "c3ViamVjdC1pZA", // ArrayBuffer codificato in base64url (user handle)
      "name": "webauthn@esegece.com",
      "displayName": "Delphi Developer"
    },
    "challenge": "Xz8x2K6nY3gZ...", // challenge codificata in base64url
    "pubKeyCredParams": [
      { "type": "public-key", "alg": -7 }, // ES256
      { "type": "public-key", "alg": -257 } // RS256
    ],
    "timeout": 60000,
    "attestation": "none",
    "authenticatorSelection": {
      "authenticatorAttachment": "platform",
      "userVerification": "preferred",
      "residentKey": "discouraged"
    }
  }
}
```

Di seguito è riportata una descrizione dei campi:

- **rp:** Informazioni sul Relying Party:
 - **name:** Nome visualizzato (ad es. "Example Inc.").
 - **id:** ID del Relying Party (deve corrispondere all'origine del sito o essere un suffisso registrabile).
- **user:** Informazioni che identificano l'account utente:
 - **id:** Array di byte univoco e stabile (ad esempio, l'UUID dell'utente).
 - **name:** Identificatore univoco dell'utente (nome utente o email).
 - **displayName:** Nome visualizzato rivolto all'utente.
- **challenge:** Un nonce crittograficamente casuale (16–64 byte), codificato in base64url.
- **pubKeyCredParams:** Algoritmi che l'RP è disposto ad accettare:
 - **Comuni:** -7 (ES256), -257 (RS256)
- **timeout:** Timeout facoltativo per l'operazione di creazione (in ms).
- **attestation:** Uno tra "none", "indirect", "direct".
- **authenticatorSelection:**
 - **authenticatorAttachment:** "platform" (integrato) o "cross-platform" (es., YubiKey).
 - **userVerification:** "required", "preferred" o "discouraged".
 - **residentKey:** "required", "preferred" o "discouraged".

Prima che la risposta venga inviata al client, viene chiamato l'evento **OnWebAuthnRegistrationOptionsResponse**, che Le consente di personalizzare la risposta.

```
void OnWebAuthnRegistrationOptionsResponse(object sender,
    WebAuthnRegistrationOptionsRequest request,
    WebAuthnRegistrationOptionsResponse response)
{
    if (request.Username == "esegece.com")
```

```
{  
    response.ExcludeCredentials.AddCredentialRecordFromJSON("json1.txt");  
    response.ExcludeCredentials.AddCredentialRecordFromJSON("json2.txt");  
}
```


WebAuthn Registration | Risultato

L'obiettivo è verificare l'autenticità e l'integrità dei dati restituiti dal client, garantire che la credenziale sia associata all'utente previsto e registrare in modo sicuro una credenziale a chiave pubblica per l'autenticazione futura.

Il server deve validare la risposta del client seguendo questi passaggi:

- Analizza la risposta del client: Il client invia una risposta come questa al server.

```
{
  "id": "base64url-encoded credential ID",
  "rawId": "base64url-encoded ID bytes",
  "response": {
    "clientDataJSON": "base64url",
    "attestationObject": "base64url"
  },
  "type": "public-key"
}
```

- Il server valida clientDataJSON e attestationObject
- Verifica l'Attestation Statement (se definito)
- Se tutto è valido, memorizza le credenziali e può considerare attendibile la chiave pubblica per i successivi accessi.

Registrazione Riuscita

Se la risposta inviata dal client è valida, l'evento **OnWebAuthnRegistrationSuccessful** viene chiamato e il Credential Record può essere memorizzato in modo sicuro in un database per le future validazioni di login.

```
void OnWebAuthnRegistrationSuccessful(
    object sender,
    WebAuthnRegistration registration,
    WebAuthnCredentialRecord credentialRecord,
    ref bool accept)
{
    // Store in a database
    using (var cmd = dbConnection.CreateCommand())
    {
        cmd.CommandText = "INSERT INTO Credentials (Credentials) VALUES (@json)";
        var param = cmd.CreateParameter();
        param.ParameterName = "@json";
        param.Value = credentialRecord.AsJson(); // or .AsJSON depending on naming
        cmd.Parameters.Add(param);
        cmd.ExecuteNonQuery();
    }
}
```

Errore di Registrazione

Se si verifica un errore durante la validazione della risposta del client, viene chiamato l'evento **OnWebAuthnRegistrationError** ed è possibile accedere alla causa dell'errore nel parametro `aError`.

```
void OnWebAuthnRegistrationError(
    object sender,
    WebAuthnRegistrationVerifyRequest request,
    WebAuthnRegistration registration,
    string error)
{
    Log("#webauthn_registration_error: " + error);
}
```

```
}
```

WebAuthn | Autenticazione

L'**autenticazione WebAuthn** consente agli utenti di accedere utilizzando credenziali a chiave pubblica precedentemente registrate. Implica la validazione di una sfida firmata utilizzando la chiave pubblica memorizzata dell'utente dalla registrazione.

Opzioni di autenticazione

L'autenticazione richiede che il client avvii il flusso di autenticazione tramite una nuova richiesta HTTP all'**Authentication Options Endpoint** configurato nel server TsgcWSAPIServer_WebAuthn. Per impostazione predefinita, l'endpoint è /sgcWebAuthn/Authentication/Options, quindi se il server è in ascolto sul dominio https://www.test.com, il client deve effettuare una nuova richiesta all'URL https://www.test.com/sgcWebAuthn/Authentication/Options.

Il client invia l'assertion (risposta di autenticazione) al server tramite POST

```
{"username": "test", "user_verification": "preferred"}
```

Il server genera PublicKeyCredentialRequestOptions

```
{
  "challenge": "9d0d61edf30b45f8b88aef7087f9117716e2b7d8b0ee4460b06142f39dd0ec9f",
  "timeout": 60000,
  "rpId": "localhost",
  "allowCredentials": [
    {
      "id": "yeA4BVRlrAfLG-KzqsL_rlI4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
      "type": "public-key",
      "transports": [
        "nfc",
        "usb"
      ]
    }
  ],
  "userVerification": "preferred",
  "hints": [],
  "attestation": "none",
  "attestationFormats": [],
  "extensions": {}
}
```

Verifica dell'autenticazione

L'autenticazione richiede che il client avvii il flusso di autenticazione utilizzando una nuova richiesta HTTP all'**Authentication Verify Endpoint** configurato nel server TsgcWSAPIServer_WebAuthn. Per impostazione predefinita, l'endpoint è /sgcWebAuthn/Authentication/Verify, quindi se il suo server è in ascolto sul dominio https://www.test.com, il client dovrebbe effettuare una nuova richiesta all'url https://www.test.com/sgcWebAuthn/Authentication/Verify.

Il browser chiede all'utente di utilizzare il proprio autenticatore (ad es. impronta digitale, YubiKey). L'autenticatore firma la challenge con la chiave privata collegata all'ID credenziale. La credenziale restituita include:

- **rawId**: ID credenziale (base64url).
- **response.authenticatorData**: Metadati dell'autenticatore (binario).
- **response.clientDataJSON**: Contiene la sfida originale, l'origine e il tipo.
- **response.signature**: Firma su authenticatorData + hash(clientDataJSON).

Trovi di seguito un esempio json della richiesta del client:

```
{
  "id": "yeA4BVRlrAfLG-KzqsL_rli4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
  "rawId": "yeA4BVRlrAfLG-KzqsL_rli4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
  "response": {
    "authenticatorData": "SZYN5YgOjGh0NBcPZHZgW4_krrmihjLHmVzzuoMdl2MFAAABw",
    "clientDataJSON": "eyJ0eXB1Ijoid2ViYXV0aG4uZ.....",
    "signature": "MEQCIAJRqvvy8....",
    "userHandle": "36b9d6a84204487382fee62e7e67a80d"
  },
  "type": "public-key",
  "clientExtensionResults": {},
  "authenticatorAttachment": "cross-platform"
}
```

Il server legge la richiesta dal client, verifica che la credenziale sia archiviata e controlla la firma. Se la firma è valida, viene chiamato l'evento **OnWebAuthnAuthenticationSuccessful**.

Flusso di autenticazione

Ulteriori informazioni sul processo del flusso di autenticazione sono disponibili nei seguenti link:

- [Richiesta di autenticazione WebAuthn](#)
- [Risposta di autenticazione WebAuthn](#)
- [Risultato dell'autenticazione WebAuthn](#)

WebAuthn Authentication | Richiesta

Quando un utente tenta di accedere, il browser invia una richiesta al server chiedendo le opzioni di autenticazione (chiamate anche "assertion options").

```
{  
  "username": "alice@example.com"  
}
```

Questa richiesta consente al server di:

- Genera una challenge (un valore casuale e sicuro).
- Verifica quali credenziali (ID di chiave pubblica) sono valide per questo utente.
- Risponde con parametri crittografici che il browser utilizzerà in `navigator.credentials.get()`.

Quando il componente server WebAuthn riceve questa richiesta, tipicamente effettuata tramite una richiesta HTTPS POST dal browser a un endpoint server (ad es. `/sgcWebAuthn/Authentication/Options`), viene chiamato l'evento **OnWebAuthnAuthenticationOptionsRequest**, così è possibile aggiungere le credenziali associate al nome utente (se presenti).

```
public void OnWebAuthnAuthenticationOptionsRequest(  
    object sender,  
    TsgcWebAuthn_AuthenticationOptions_Request aRequest,  
    ref TsgcWebAuthn_CredentialRecords credentialRecords,  
    ref bool accept)  
{  
    if (UserExistsInDB(aRequest.Username))  
    {  
        while (!EOF())  
        {  
            credentialRecords.AddCredentialRecordFromJSON(RecordFromDB());  
            Next();  
        }  
    }  
}
```

WebAuthn Authentication | Response

Una volta che il browser (client) invia una richiesta al server con il nome utente dell'utente (o un identificatore simile), il server risponde con le opzioni di autenticazione che il client utilizzerà per avviare il processo di autenticazione tramite l'API `navigator.credentials.get()` del browser.

Questa risposta fornisce i parametri e i vincoli necessari al browser per generare un'asserzione di autenticazione WebAuthn utilizzando l'autenticatore dell'utente (ad es. chiave di sicurezza, dispositivo biometrico).

Esempio di risposta JSON:

```
{
  "challenge": "Z3lVbWV5YXBpbmdvZG90IQ",
  "timeout": 60000,
  "rpId": "example.com",
  "allowCredentials": [
    {
      "type": "public-key",
      "id": "dXNlckNyZWJZA",
      "transports": ["usb", "nfc", "ble"]
    }
  ],
  "userVerification": "preferred"
}
```

Di seguito è riportata una descrizione dei campi:

- **challenge:** Una sfida crittografica (generata casualmente) per prevenire gli attacchi di replay. Deve essere univoca per ogni richiesta.
- **timeout:** Opzionale. Per quanto tempo (in ms) il browser deve attendere prima di annullare l'operazione.
- **rpId:** Identificatore della Relying Party — di solito il nome del proprio dominio.
- **allowCredentials:** Un elenco di credenziali accettabili (ID credenziale) precedentemente registrate dall'utente.
- **userVerification:** Informa il browser su come verificare l'utente: "required", "preferred" o "discouraged".
- **extensions (facoltativo):** Funzionalità o dati aggiuntivi richiesti all'autenticatore.

Prima che la risposta venga inviata al client, viene chiamato l'evento **OnWebAuthnAuthenticationOptionsResponse**, consentendo di personalizzare la risposta.

WebAuthn Authentication | Risultato

L'obiettivo è validare l'asserzione firmata fornita dal browser, che dimostra che l'utente possiede la chiave privata originariamente registrata. Questo è ciò che autentica in modo sicuro l'utente. Dopo che l'utente interagisce con il proprio autenticatore (ad esempio, impronta digitale, chiave di sicurezza), il browser invia una richiesta POST al server con il risultato dell'autenticazione. Di seguito è riportato un esempio in formato JSON:

```
{
  "id": "credential-id",
  "rawId": "base64url-encoded-credential-id",
  "type": "public-key",
  "response": {
    "clientDataJSON": "base64url",
    "authenticatorData": "base64url",
    "signature": "base64url",
    "userHandle": "optional"
  }
}
```

Quando il server riceve questa richiesta all'endpoint configurato (ad es., /sgcWebAuthn/Authentication/Verify), deve validare i seguenti passaggi:

- Recupera la credenziale dal DB: Utilizzare l'ID della credenziale per cercare la chiave pubblica registrata dell'utente e i metadati (ad esempio, il contatore delle firme).
- Decodifica & Analisi di clientDataJSON
- Convalida i dati dell'autenticatore
- Verifica firma

Se tutte le validazioni sono corrette, l'autenticazione ha esito positivo e viene chiamato l'evento **OnWebAuthnAuthenticationSuccessful**.

Se uno qualsiasi dei controlli fallisce, viene chiamato l'evento **OnWebAuthnAuthenticationError** con il motivo dell'errore.

WebAuthn | MDS

Il **Metadata Service (MDS)** è un **servizio centralizzato fornito dalla FIDO Alliance** che aggrega e pubblica **Metadata Statements** sugli autenticator certificati attraverso i programmi di certificazione FIDO. Queste dichiarazioni contengono informazioni dettagliate sulla sicurezza, conformità e operatività degli autenticator.

- L'endpoint del servizio è spesso indicato come **MDS3** (la terza versione del protocollo Metadata Service).
- Le Relying Party (ad esempio siti web o applicazioni che implementano WebAuthn) **recuperano dichiarazioni di metadati** dall'MDS per prendere decisioni di fiducia informate sugli autenticator.

Il MDS aggiunge un livello cruciale di fiducia e convalida della sicurezza per le relying party che utilizzano WebAuthn:

- **Validazione Autenticatore:** Consente la verifica della conformità dell'autenticatore agli standard FIDO e aiuta a validare l'AAGUID presentato in un'attestazione WebAuthn.
- **Rilevamento di compromissioni e revoche:** Fornisce informazioni aggiornate sugli autenticator compromessi o revocati, consentendo alle relying party di bloccare i dispositivi non sicuri.
- **Garanzia di sicurezza:** Aiuta a far rispettare le policy di sicurezza, ad esempio consentendo solo gli autenticator che soddisfano un determinato livello di certificazione FIDO o una certa robustezza della verifica dell'utente.
- **Interoperabilità:** Garantisce un comportamento coerente e aspettative di sicurezza uniformi tra diversi browser, piattaforme e dispositivi che utilizzano autenticator differenti.

Configurazione

È possibile configurare l'utilizzo di MDS tramite la proprietà `WebAuthnOptions.MDS`; di seguito sono riportate le proprietà principali:

- **Enabled:** se true (il valore predefinito), le richieste webauthn verranno validate rispetto al file MDS configurato.
- **MDS_FileName:** il percorso in cui è memorizzato il file MDS. Può essere scaricato dal seguente URL: <https://mds3.fidoalliance.org/>
- **RootCert_FileName:** il percorso in cui è memorizzato il certificato radice. Deve essere definito per validare la catena di certificati. Può essere scaricato da <https://valid.r3.roots.globalsign.com/>
- **Leaf_CertificateCRL:** se true (per impostazione predefinita è false), il certificato foglia del blob verrà validato rispetto alla CRL (Certificate Revocation List).
- **CRL_FileName:** il percorso in cui è archiviato il file CRL. Se non è definito e `Leaf_CertificateCRL` è abilitato, verrà tentato il download automatico del CRL.

Autorizzazione WebAuthn

Se si desidera che il server invii un bearer token dopo un'autenticazione riuscita, utilizzabile per aprire una nuova connessione WebSocket o HTTP, passare il parametro `token = true`. Esempio:

```
{  
  "username": "alice@example.com", "token": true  
}
```

Dopo un'autenticazione riuscita, il server invierà una risposta come questa:

```
{  
  "verified": "ok",  
  "authentication": {  
    "token": "C760C1C39E3D4E829693A13F18F5CFDE537B516336FC48F7BAB0276176F9E6DE"  
  }  
}
```

L'evento **OnWebAuthnUnauthorized** viene chiamato quando una richiesta non è autorizzata e la connessione verrà disconnessa. Qui è possibile configurare quali endpoint richiedono l'autenticazione WebAuthn e quali no.

Autorizzazione WebAuthn | HTTP

Una volta ottenuto un nuovo token, inviare semplicemente un'intestazione di autorizzazione con questo bearer token. È possibile utilizzare la proprietà CustomHeaders di TsgcHTTP1Client per configurare il Bearer Token. Esempio:

```
public static async Task<string> GetHTTPRequestAsync(string aURL, string aToken)
{
    using (var client = new HttpClient())
    {
        // Add Authorization header
        client.DefaultRequestHeaders.Add("Authorization", "Bearer " + aToken);
        // Send GET request
        HttpResponseMessage response = await client.GetAsync(aURL);
        // Throw exception if request failed
        response.EnsureSuccessStatusCode();
        // Return the response content as string
        return await response.Content.ReadAsStringAsync();
    }
}
```

WebAuthn Authorization | WebSocket

Una volta ottenuto un nuovo token, inviare semplicemente un'intestazione di autorizzazione con questo bearer token. È possibile utilizzare l'evento OnHandShake per aggiungere il Bearer Token alla richiesta di connessione. Esempio:

```
public static async Task<ClientWebSocket> ConnectWithAuthAsync(Uri serverUri)
{
    var client = new ClientWebSocket();
    // Add custom Authorization header
    client.Options.SetRequestHeader("Authorization", "Bearer C760C1C39E3D4E829693A13F18F5CFDE537B516336FC48F7BAB6
    // Connect to WebSocket server
    await client.ConnectAsync(serverUri, CancellationToken.None);
    return client;
}
```

Webauthn | Javascript Client

WebAuthn (Web Authentication API) è uno standard W3C che consente un'autenticazione sicura senza password utilizzando la **crittografia a chiave pubblica**. Invece delle password, gli utenti si registrano e si autenticano utilizzando **authenticator basati su hardware** (come lettori di impronte digitali, Face ID, YubiKey, ecc.) o authenticator di piattaforma (integrati, come Touch ID).

Di seguito è illustrato come gestire la Registrazione e l'Autenticazione utilizzando un client Javascript.

Registrazione WebAuthn

Come funziona la registrazione WebAuthn

- **L'utente avvia la registrazione:**
 - L'utente fornisce un nome utente e clicca su **Registra**.
- **Il browser richiede le opzioni al server:**
 - Il frontend invia una richiesta POST per ottenere le opzioni di registrazione.
- **Il browser crea le credenziali:**
 - Utilizzando l'API `navigator.credentials.create()` tramite `SimpleWebAuthnBrowser.startRegistration()`, viene creata una credenziale.
- **Il server verifica la registrazione:**
 - Il browser invia le credenziali al server.
 - Il server verifica la registrazione e memorizza la chiave pubblica per quell'utente.

Il componente [TsgcWSAPIServer_WebAuthn](#) include un file HTML per testare il protocollo WebAuthn. Questo file HTML contiene un'interfaccia utente minimale e JavaScript per interagire con WebAuthn.

Analisi dettagliata di `sgcWebAuthn.html`

Panoramica della struttura:

- **Input Nome Utente:** Acquisisce l'identificatore dell'utente.
- **Pulsanti:**
 - Register – avvia la registrazione WebAuthn.
 - Authenticate – avvia il login (gestito in modo simile).
- **Console di debug:** mostra informazioni di debug in tempo reale (JSON da WebAuthn).

1. Interfaccia HTML per l'Input

```
<input type="text" id="username" name="username" autocomplete="username webauthn" />
<button id="btnRegBegin"><strong>Register</strong></button>
```

2. JavaScript: Gestore clic sul pulsante

```
document.querySelector('#btnRegBegin').addEventListener('click', async () => {
  const username = document.getElementById("username").value;
  if (username == "") {
    document.getElementById('Error').innerText = 'Please enter a username to register';
    return;
  }
  const resp = await fetch('<#webauthn_registration_options>', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ username, algorithms: [] })
  });
  const options = await resp.json();
  const attResp = await startRegistration(options); // WebAuthn API
```

3. Risposta del server (endpoint fittizio in HTML)

```
fetch('/sgcWebAuthn/Registration/Options', ...)
fetch('/sgcWebAuthn/Registration/Verify', ...)
```

4. Finalizzazione della registrazione

```
const verificationResp = await fetch('/webauthn/register/verify', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(attResp)
});
const verificationJSON = await verificationResp.json();
if (verificationJSON && verificationJSON.verified) {
  document.getElementById('Success').innerHTML = `Authenticator registered!`;
}
```

Autenticazione WebAuthn

Come funziona l'autenticazione WebAuthn

- **L'utente avvia l'autenticazione:**
 - L'utente fornisce un nome utente e clicca su **Registra**.
- **Il browser richiede le opzioni al server:**
 - Il frontend effettua una richiesta POST per ottenere le opzioni di autenticazione.
- **Il browser crea le credenziali:**
 - Utilizzando l'API `navigator.credentials.get()` tramite `SimpleWebAuthnBrowser.startAuthentication()`.
- **Il server verifica l'autenticazione:**
 - Il browser rinvia la credenziale al server.
 - Il server verifica il risultato firmato.

1. Configurazione UI HTML

```
<div class="container">
<h1>WebAuthn Authentication Sample</h1>
<section id="userdata">
<label for="username">Username:</label>
<input type="text" id="username" name="username" autocomplete="username webauthn" autofocus />
</section>
<button id="btnAuthBegin"><strong>Authenticate</strong></button>
<p id="Success" class="success"></p>
<p id="Error" class="error"></p>
<details open>
<summary>Console</summary>
<textarea id="Debug"></textarea>
</details>
</div>
```

2. Ottieni le opzioni di autenticazione

Prima di chiamare `startAuthentication`, si invia lo username al server

```
const resp = await fetch('<#webauthn_authentication_options>', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    username: document.getElementById("username").value,
    user_verification: 'preferred'
  }),
});
```

Il server risponde con un oggetto JSON che include:

```
{
  "challenge": "base64url-encoded-random-string",
  "allowCredentials": [
    {
      "id": "base64url-credential-id",
      "type": "public-key"
    }
  ],
  "userVerification": "preferred",
  "rpId": "yourdomain.com"
}
```

Questo viene chiamato `PublicKeyCredentialRequestOptions`.

3. Ricezione della risposta dell'autenticatore

Il `asseResp` si presenta così (semplificato):

```
{
  "id": "credentialId",
  "rawId": "base64url-encoded-id",
  "response": {
    "authenticatorData": "...",
    "clientDataJSON": "...",
    "signature": "...",
    "userHandle": "..."
  },
  "type": "public-key",
  "clientExtensionResults": {}
}
```

Questa risposta dimostra che l'utente:

- Possiede la chiave privata memorizzata sull'autenticatore
- Ha firmato la sfida del server
- Era fisicamente presente (se richiesto)

4. Inviare la risposta di autenticazione firmata al server

Dopo che l'utente interagisce con il proprio authenticator (tramite `startAuthentication()`), ottiene un oggetto `response` in JavaScript.

```
const verificationResp = await fetch('/webauthn/authenticate/verify', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(asseResp), // this is the signed response
});
```

5. Ottenere la risposta del server

Il server risponderà con un risultato simile a questo:

```
{ "verified": true }
```

Oppure se qualcosa è andato storto:

```
{ "verified": false, "error": "Invalid signature" }
```

E lo si gestisce nel codice frontend:

```
const result = await verificationResp.json();
if (result.verified) {
  document.getElementById('Success').textContent = 'User authenticated!';
} else {
  document.getElementById('Error').textContent = 'Authentication failed!';
}
```

Amazon AWS | SQS

Che cos'è Amazon SQS?

Amazon Simple Queue Service (SQS) è un servizio di accodamento messaggi completamente gestito che consente di disaccoppiare e scalare microservizi, sistemi distribuiti e applicazioni serverless. SQS elimina la complessità e il sovraccarico associati alla gestione e all'operatività del middleware orientato ai messaggi, e consente agli sviluppatori di concentrarsi sul lavoro differenziante. Utilizzando SQS, è possibile inviare, archiviare e ricevere messaggi tra componenti software a qualsiasi volume, senza perdere messaggi o richiedere la disponibilità di altri servizi.

Vantaggi

Eliminazione degli oneri amministrativi

Con SQS non vi sono costi iniziali, non è necessario acquistare, installare e configurare software di messaggistica e non vi sono attività di configurazione e manutenzione dell'infrastruttura di supporto che richiedono molto tempo.

Consegna affidabile dei messaggi

SQS consente di disaccoppiare i componenti dell'applicazione in modo che vengano eseguiti e falliscano in modo indipendente, aumentando la tolleranza ai guasti complessiva del sistema.

Mantenere i dati sensibili al sicuro

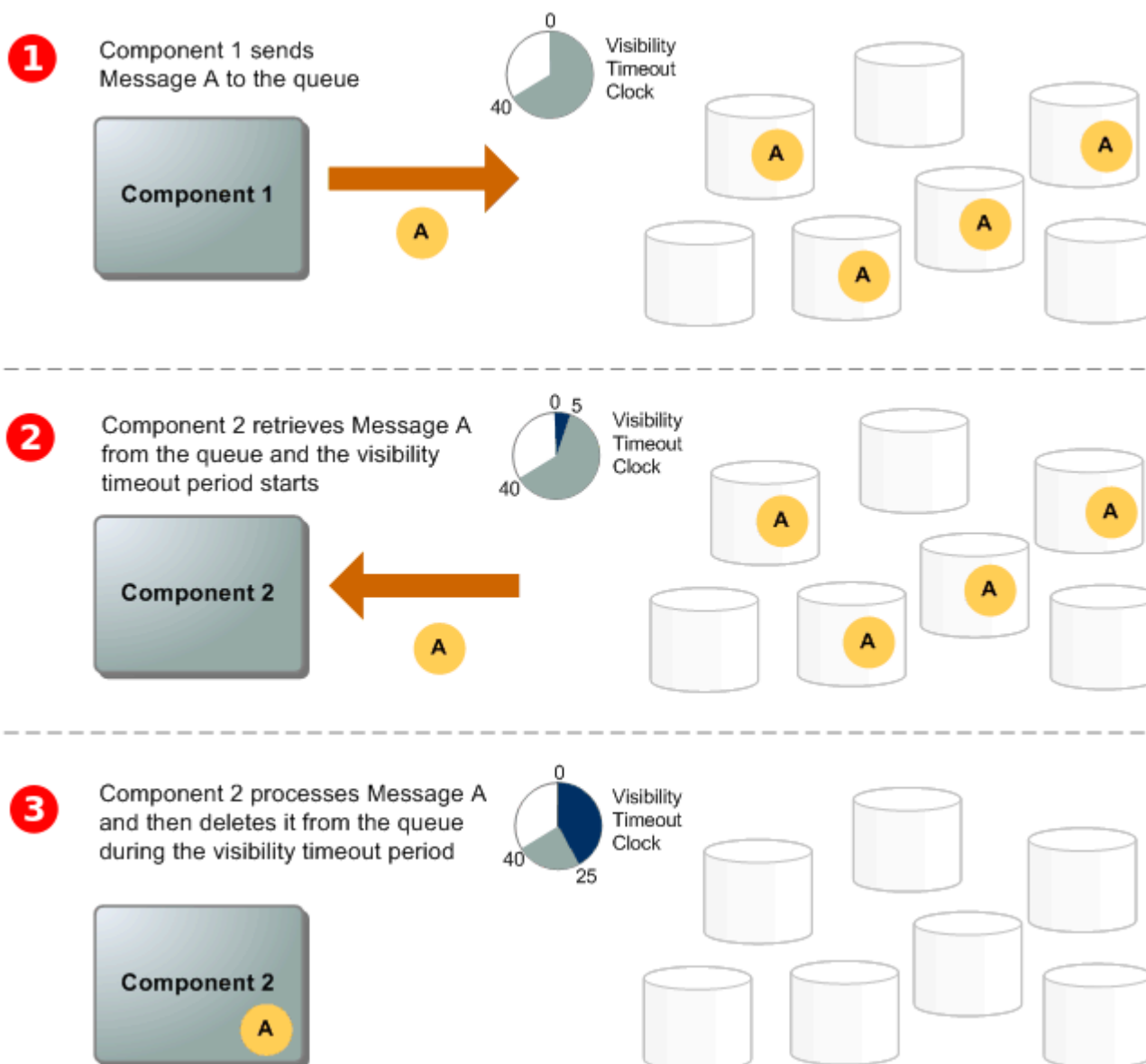
È possibile utilizzare Amazon SQS per scambiare dati sensibili tra applicazioni tramite la crittografia lato server (SSE) per cifrare il corpo di ogni messaggio.

Scalabilità elastica ed economicamente efficiente

SQS si scala elasticamente con la propria applicazione, quindi non è necessario preoccuparsi della pianificazione della capacità e del pre-provisioning.

WorkFlow

Lo scenario seguente descrive il ciclo di vita di un messaggio Amazon SQS in una coda, dalla creazione all'eliminazione.



Introduzione ad Amazon SQS

Prima di iniziare, completare i passaggi descritti in Configurazione di Amazon SQS.

Passaggio 1: Creare una coda

1. Accedere alla console Amazon SQS.
2. Scegli Crea nuova coda.
3. Nella pagina Crea nuova coda, assicurarsi di essere nella regione corretta, quindi digitare il nome della coda.
4. Standard è selezionato per impostazione predefinita. Scegliere FIFO.
5. Per creare la propria coda con i parametri predefiniti, scegliere Crea coda rapida.

La nuova coda è stata creata e selezionata nell'elenco delle code.

Fase 2: Inviare un messaggio

Dopo aver creato la coda, è possibile inviare un messaggio. L'esempio seguente mostra come inviare un messaggio a una coda esistente.

1. Dall'elenco delle code, selezionare la coda creata.
2. Da Azioni coda, selezionare Invia un messaggio.

3. Il messaggio viene inviato e viene visualizzata la finestra di dialogo Send a Message to QueueName, che mostra gli attributi del messaggio inviato.

Passo 3: Ricevere ed eliminare il proprio messaggio

Dopo aver inviato un messaggio in una coda, è possibile consumarlo (recuperarlo dalla coda). Quando si richiede un messaggio da una coda, non è possibile specificare quale messaggio ottenere. Si specifica invece il numero massimo di messaggi (fino a 10) che si desidera ottenere.

Passo 4: Elimina la Tua Coda

Se non usa una coda Amazon SQS (e non prevede di usarla nel prossimo futuro), è buona pratica eliminarla da Amazon SQS.

SQS Client

Eventi

OnSQSBeforeRequest

Questo evento viene chiamato prima che il componente sqs esegua una richiesta HTTP. È possibile accedere al parametro URL e, se il parametro Handled viene impostato su True, il componente non eseguirà la richiesta HTTP.

OnSQSError

Se si verifica un errore durante l'esecuzione di una richiesta da parte del componente, questo evento verrà chiamato con il codice di errore e la descrizione dell'errore.

OnSQSResponse

Questo evento viene chiamato dopo una richiesta HTTP con risposta grezza dal server.

Google Cloud | Chiavi OAuth2 di Google

Per utilizzare i componenti Google Cloud di sgcWebSockets e autenticarsi tramite OAuth2, è innanzitutto necessario ottenere la chiave OAuth2 da Google Cloud.

Di seguito sono riportati i passaggi per ottenere le chiavi OAuth2 di Google e come configurarle nella nostra applicazione di esempio PubSub.

Prima **accedere** al proprio **Account Google Cloud** e usare un progetto esistente o crearne uno nuovo. Successivamente, andare al menu **Credentials** e premere il pulsante **CREATE CREDENTIALS**, selezionare l'opzione **OAuth Client ID**.

Selezionare il tipo di applicazione e impostare un nome descrittivo.

In caso di successo, si riceveranno il proprio Client ID e il Client Secret.

OAuth client created

The client ID and secret can always be accessed from Credentials in APIs & Services



OAuth is limited to 100 [sensitive scope logins](#) until the [OAuth consent screen](#) is verified. This may require a verification process that can take several days.

Your Client ID

843483347040-ehcskpfsp4180r1bdf0e6mc32e3ncmn0.apps.gc



Your Client Secret

pvogD9reE0t9i1L6eR1jE60Z



OK

Non condividere i propri dati OAuth2 con nessuno!

Copiare ora nel campione PubSub di sgcWebSockets, e aggiungere il Project Id (NON il nome del progetto)

Select a project



NEW PROJECT

RECENT

ALL

Name		ID
✓ 	PubSub 	pubsub-270909
	GmailApiTest 	gmailapitest-265010
	My Project 	melodic-voice-265009

CANCEL

OPEN

Ecco come deve essere configurato nel campione sgcWebSocket PubSub.

The screenshot shows the Google Pub/Sub API Client interface, which is divided into three main sections: Google API, Publisher, and Subscriber.

Google API Section:

- Client Id:** 843483347040-ehcskpfsp4180rlbdfoe6mc32e3ncmn0.app
- Client Secret:** pvogD9reEOt9iLL6eR1jE60Z

Publisher Section:

- Project-id:** pubsub-270909
- Topic:** topic-1
- Buttons:** Create Topic, Delete Topic, Get Topic, List Topics, Publish.
- Message:** Message sent from sgcWebSockets Pub/Sub API.

Subscriber Section:

- Project-id:** (empty field)
- Subscription:** subscription-1
- Topic:** topic-1
- Buttons:** Create Subs., Delete Subs., Get Subs., List Subs., Pull, Acknowledge.
- Input Field:** (empty field for message acknowledgment)

Quindi è possibile provare a creare un nuovo argomento, ad esempio. La prima volta, è necessario autorizzare la connessione OAuth2, quindi verrà mostrato un nuovo browser web per richiedere l'autorizzazione all'accesso al proprio account con le credenziali OAuth2 fornite da Google.

 Sign in with Google

Confirm your choices

 sgomez@gmail.com

You are allowing **PubSub** to:

☒ View and manage Pub/Sub topics and subscriptions

Make sure you trust PubSub

You may be sharing sensitive info with this site or app.
Learn about how PubSub will handle your data by reviewing
its terms of service and privacy policies. You can always
see or remove access in your [Google Account](#).

[Learn about the risks](#)

[Cancel](#)[Allow](#)

Consente la connessione e, in caso di esito positivo, può iniziare a lavorare con questa API.

 Google Pub/Sub API Client

Google API

Client Id

Client Secret

Publisher

Project-id Topic

Subscriber

Project-id Subscription

Topic

```

#CreateTopic: { "name": "projects/pubsub-270909/topics/topic-1"}
#Publish: { "messageIds": [ "1780785665748120" ]}
#Publish: { "messageIds": [ "1780799518708503" ]}
    
```


Google Cloud | Pub/Sub

Che cos'è Google Cloud Pub/Sub?

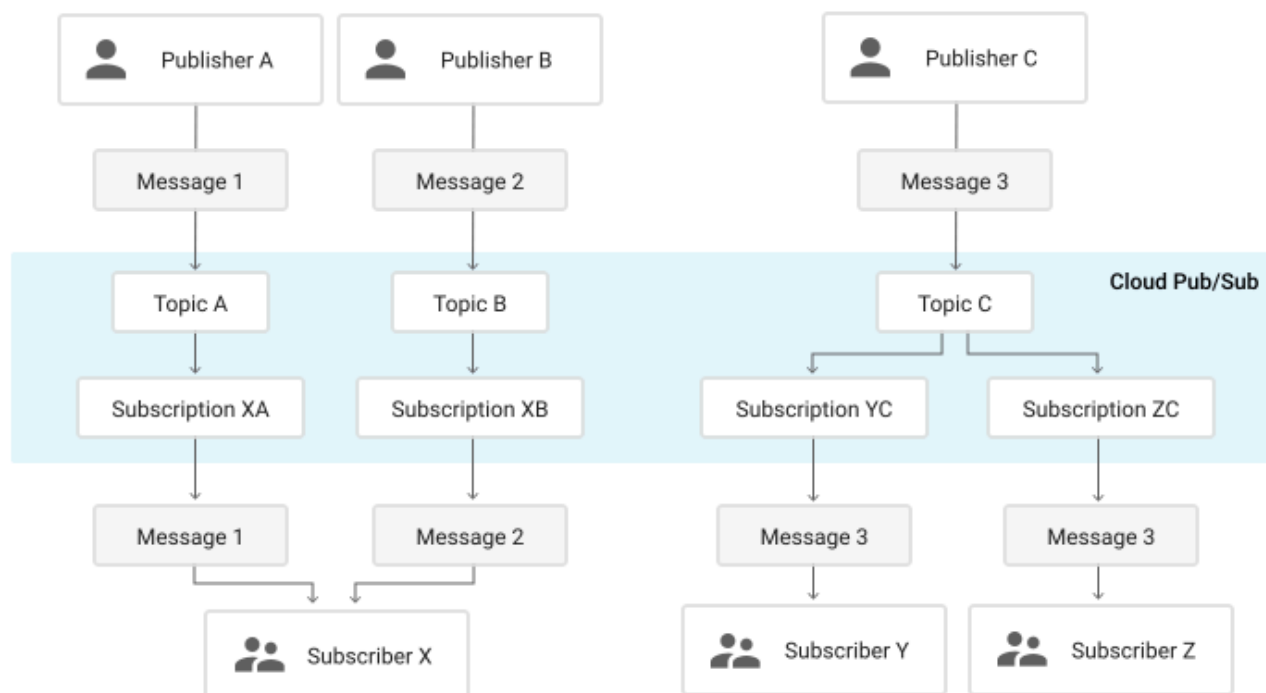
Pub/Sub porta la flessibilità e l'affidabilità del middleware di messaggistica enterprise orientato ai messaggi nel cloud. Allo stesso tempo, Pub/Sub è un sistema scalabile e durevole di acquisizione e distribuzione di eventi che funge da base per le moderne pipeline di analisi dei flussi. Fornendo messaggistica asincrona multi-a-molti che dissocchia mittenti e destinatari, consente una comunicazione sicura e altamente disponibile tra applicazioni scritte in modo indipendente. Pub/Sub offre messaggistica a bassa latenza e durevole che aiuta gli sviluppatori a integrare rapidamente i sistemi ospitati su Google Cloud Platform e quelli esterni.

Funzionalità

Recapito at-least-once La replica sincrona dei messaggi tra zone e il tracciamento della ricezione per ogni messaggio garantiscono il recapito at-least-once a qualsiasi scala.	Open API aperte e librerie client in sette linguaggi supportano distribuzioni cross-cloud e ibride.	Elaborazione exactly-once Cloud Dataflow supporta un'elaborazione affidabile, espressiva ed exactly-once degli stream di Cloud Pub/Sub.
Globale per impostazione predefinita Pubblicare da qualsiasi parte del mondo e consumare da qualsiasi parte, con latenza costante. Nessuna replica necessaria.	Nessuna configurazione, tutto automatico Cloud Pub/Sub non ha shard né partizioni. Basta impostare la propria quota, pubblicare e consumare.	Conformità e sicurezza Cloud Pub/Sub è un servizio conforme a HIPAA, che offre controlli di accesso granulari e crittografia end-to-end.
Integrato Sfruttare le integrazioni con molteplici servizi, come gli eventi di aggiornamento di Cloud Storage e Gmail e Cloud Functions per il computing serverless basato su eventi.	Cerca e riproduci Riavvolga il backlog fino a qualsiasi punto nel tempo o uno snapshot, offrendo la possibilità di rielaborare i messaggi. Avanzi velocemente per eliminare i dati obsoleti.	

Relazioni pubblicatore-sottoscrittore

Un'applicazione publisher crea e invia messaggi a un topic. Le applicazioni subscriber creano una sottoscrizione a un topic per ricevere messaggi da esso. La comunicazione può essere uno-a-molti (fan-out), molti-a-uno (fan-in) e molti-a-molti.



Casi d'uso comuni

- **Bilanciamento dei carichi di lavoro nei cluster di rete.** Ad esempio, una grande coda di attività può essere distribuita in modo efficiente tra più worker, come le istanze di Google Compute Engine.
- **Implementazione di flussi di lavoro asincroni.** Ad esempio, un'applicazione di elaborazione degli ordini può inserire un ordine su un topic, da cui può essere elaborato da uno o più lavoratori.
- **Distribuzione di notifiche di eventi.** Ad esempio, un servizio che accetta registrazioni di utenti può inviare notifiche ogni volta che un nuovo utente si registra, e i servizi downstream possono sottoscrivere per ricevere le notifiche dell'evento.
- **Aggiornamento di cache distribuite.** Ad esempio, un'applicazione può pubblicare eventi di invalidazione per aggiornare gli ID degli oggetti modificati.
- **Registrazione su più sistemi.** Ad esempio, un'istanza di Google Compute Engine può scrivere log nel sistema di monitoraggio, in un database per query successive e così via.
- **Streaming di dati da vari processi o dispositivi.** Ad esempio, un sensore residenziale può trasmettere dati ai server backend ospitati nel cloud.
- **Miglioramento dell'affidabilità.** Ad esempio, un servizio Compute Engine a zona singola può operare in zone aggiuntive sottoscrivendo un argomento comune, per recuperare da guasti in una zona o regione.

Authorization

Il client del componente Google Pub/Sub può effettuare il login ai server Google utilizzando i seguenti metodi:

- **gcaOAuth2:** protocollo OAuth2
- **gcaJWT:** Token JWT.

OAuth2

L'accesso viene effettuato tramite un browser web in cui l'utente accede con il proprio account e autorizza le richieste PubSub.

- **GoogleCloudOptions.OAuth2.ClientId:** è il ClientID fornito da Google per l'autenticazione tramite il protocollo OAuth2.
- **GoogleCloudOptions.OAuth2.ClientSecret:** è la stringa Client Secret fornita da Google per l'autenticazione tramite il protocollo OAuth2.
- **GoogleCloudOptions.OAuth2.Scope:** è l'ambito di OAuth2; di norma non è necessario modificare il valore predefinito, a meno che non si necessiti di un accesso superiore a quello predefinito.
- **GoogleCloudOptions.OAuth2.LocalIP:** il protocollo OAuth2 richiede un server locale in ascolto per la risposta dal server di autenticazione. Questo è l'IP o il nome DNS. Per impostazione predefinita è 127.0.0.1.
- **GoogleCloudOptions.OAuth2.LocalPort:** porta di ascolto del server locale.
- **GoogleCloudOptions.OAuth2.RedirectURL:** se è necessario impostare un URL di reindirizzamento diverso da LocalPort + LocalIP, è possibile impostarlo in questa proprietà (esempio: `http://127.0.0.1:8080/oauth2`).

Account di servizio

L'accesso viene effettuato firmando le richieste con una chiave privata fornita da Google. Questo metodo è consigliato per servizi automatizzati o applicazioni senza interazione dell'utente.

- **GoogleCloudOptions.JWT.ClientEmail:** il nome dell'email del client fornito durante la creazione del nuovo service account. Nodo "client_email" nel file di configurazione JSON.
- **GoogleCloudOptions.JWT.PrivateKeyId:** l'ID della chiave privata fornito da Google. Nodo "private_key_id" nel file di configurazione JSON.
- **GoogleCloudOptions.JWT.PrivateKey:** il certificato della chiave privata fornito da Google. Nodo "private_key" nel file di configurazione JSON.

Opzioni TLS

TLSOptions centralizza la configurazione del canale sicuro utilizzato per comunicare con Google Cloud Pub/Sub.

- **TLSOptions.IOHandler:** scegliere l'implementazione TLS adatta al proprio ambiente di distribuzione (OpenSSL, SChannel...).
- **TLSOptions.Version:** limita la negoziazione a una specifica versione TLS quando la propria infrastruttura lo richiede.
- **TLSOptions.VerifyCertificate:** controlla la validazione del certificato, abilitando policy di sicurezza più rigorose.
- **TLSOptions.OpenSSL_Options.LibPath:** impostare la directory contenente i binari OpenSSL distribuiti con l'applicazione.
- **TLSOptions.SChannel_Options.UseLegacyCredentials:** attiva la gestione delle credenziali legacy di Windows quando richiesto da SChannel.

Esempio:

```
oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client();
oPubSub.TLSOptions.IOHandler = TwsTLSIOHandler.iohOpenSSL;
oPubSub.TLSOptions.Version = TwsTLSVersions.tls1_3;
oPubSub.TLSOptions.VerifyCertificate = true;
oPubSub.TLSOptions.OpenSSL_Options.LibPath = oslpDefaultFolder;
```

Quando viene creato un nuovo account di servizio, è possibile scaricare un file JSON con tutte le configurazioni. Questo file può essere elaborato dal componente PubSub: è sufficiente chiamare il metodo **LoadSettingsFromFile** e passare il nome del file JSON come argomento.

Usi più comuni

- Configurazione
 - [Chiavi Google OAuth2](#)
 - Account di servizio

Google Pub/Sub Client

OAuth2

Per lavorare con Google Pub/Sub API, il componente Pub/Sub di `sgcWebSockets` utilizza OAuth2 come autenticazione predefinita, quindi occorre prima impostare il proprio **ClientId** e **ClientSecret** dall'account Google.

```
oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client();
oPubSub.GoogleCloudOptions.Authorization = gcaOAuth2;
oPubSub.GoogleCloudOptions.OAuth2.ClientId = "... your google client id...";
oPubSub.GoogleCloudOptions.OAuth2.ClientSecret = "... your google client secret...";
```

Account di servizio

Gli account di servizio richiedono la costruzione di un JWT e il suo passaggio come token di autorizzazione.

```
TsgcHTTPGoogleCloud_PubSub_Client oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client();
oPubSub.GoogleCloudOptions.Authorization = gcaJWT;
oPubSub.GoogleCloudOptions.JWT.ClientEmail = "...google email...";
oPubSub.GoogleCloudOptions.JWT.PrivateKeyId = "...private key id...";
oPubSub.GoogleCloudOptions.JWT.PrivateKey = "...private key certificate...";
```

Questo è necessario per ottenere un Token Chiave di Autorizzazione da Google che verrà utilizzato per tutte le chiamate REST API.

Tutti i metodi restituiscono una risposta, che può avere esito positivo o restituire un errore.

Projects.Snapshots

Metodo	Parametri	Descrizione	Esempio
Create-Snapshot	project, snapshot, subscription	Crea uno snapshot dalla sottoscrizione richiesta. Gli snapshot vengono utilizzati nelle operazioni <code>subscriptions.seek</code> , che consentono di gestire in blocco i riconoscimenti dei messaggi. È possibile impostare lo stato di riconoscimento dei messaggi in una sottoscrizione esistente allo stato catturato da uno snapshot.	<code>CreateSnapshot('pubsub-270909', 'snapshot-1', 'subscription-1')</code>
Delete-Snapshot	project, snapshot	Rimuove uno snapshot esistente	<code>DeleteSnapshot('pubsub-270909', 'snapshot-1')</code>

ListSnapshots	progetto	Elenca gli snapshot esistenti	ListSnapshots('pubsub-270909')

Projects.Subscriptions

Metodo	Parametri	Descrizione	Esempio
AcknowledgeSubscription			
CreateSubscription	project, subscription, topic	Crea una sottoscrizione a un determinato topic. Se la sottoscrizione esiste già, restituisce ALREADY_EXISTS. Se il topic corrispondente non esiste, restituisce NOT_FOUND.	CreateSubscription('pubsub-270909', 'subscription-1', 'topic-1')
DeleteSubscription	project, subscription	Elimina una sottoscrizione esistente. Tutti i messaggi trattenuti nella sottoscrizione vengono immediatamente eliminati.	DeleteSubscription('pubsub-270909', 'subscription-1')
GetSubscription	project, subscription	Ottiene i dettagli di configurazione di un abbonamento.	GetSubscription('pubsub-270909', 'subscription-1')
ListSubscriptions	progetto	Elenca le sottoscrizioni corrispondenti.	ListSubscriptions('pubsub-270909', 'subscription-1')
ModifyAckDeadlineSubscription	project, subscription, AckIds	Modifica la scadenza dell'ack per un messaggio specifico. Questo metodo è utile per indicare che è necessario più tempo per elaborare un messaggio da parte del sottoscrittore, oppure per rendere il messaggio disponibile per la riconsegna se l'elaborazione è stata interrotta. Si	

		noti che questo non modifica l'ackDeadlineSeconds a livello di sottoscrizione utilizzato per i messaggi successivi.	
Modify-Push-Config-Subscription	project, subscription	Modifica la PushConfig per una sottoscrizione specificata. Questa operazione può essere utilizzata per trasformare una sottoscrizione push in una pull (indicata da una PushConfig vuota) o viceversa, oppure per modificare l'URL dell'endpoint e altri attributi di una sottoscrizione push. I messaggi continueranno ad accumularsi per la consegna in modo continuativo durante la chiamata, indipendentemente dalle modifiche alla PushConfig.	
Pull	project, subscription	Recupera i messaggi dal server. Il server potrebbe restituire UNAVAILABLE se ci sono troppe richieste di pull simultanee in sospeso per la sottoscrizione specificata.	<pre>pull('pubsub-270909', 'subscription-1')</pre>
Seek	project, subscription, timeUTC, snapshot	Ricerca una sottoscrizione esistente fino a un punto nel tempo o a uno snapshot specifico, a seconda di ciò che viene fornito nella richiesta. Gli snapshot vengono utilizzati nelle operazioni subscriptions.seek, che consentono di gestire le conferme dei messaggi in blocco. In altre parole, è possibile impostare lo stato di conferma dei messaggi in una sottoscrizione esi-	

stente allo stato acquisito da uno snapshot. Si noti che sia la sottoscrizione sia lo snapshot devono trovarsi sullo stesso argomento.

Projects.Topics

Metodo	Parametri	Descrizione	Esempio
CreateTopic	project, topic	Crea il topic specificato con il nome specificato	CreateTopic('pubsub-270909', 'topic-1')
DeleteTopic	project, topic	Elimina il topic con il nome specificato. Restituisce NOT_FOUND se il topic non esiste. Dopo l'eliminazione di un topic, è possibile creare un nuovo topic con lo stesso nome; si tratterà di un topic completamente nuovo, privo della configurazione o delle sottoscrizioni precedenti.	DeleteTopic('pubsub-270909', 'topic-1')
GetTopic	project, topic	Ottiene la configurazione di un argomento.	GetTopic('pubsub-270909', 'topic-1')
ListTopics	progetto	Elenca gli argomenti corrispondenti.	ListTopics('pubsub-270909')
Pubblica	project, topic, message	Aggiunge uno o più messaggi al topic. Restituisce NOT_FOUND se il topic non esiste.	Publish('pubsub-270909', 'topic-1', 'My First PubSub Message.')

Projects.Topics.Subscriptions

Metodo	Parametri	Descrizione	Esempio
ListTopicSubscriptions	project, topic	Elenca i nomi delle sottoscrizioni su	ListTopicSubscriptions('pubsub-270909', 'topic-1')

scrip- tions		questo to- pic.	

Metodi più comuni

Di seguito sono riportati i metodi più comuni utilizzati con l'API Google Cloud Pub/Sub.

Come creare un nuovo Topic

Creare un nuovo topic per il progetto con id: pubsub-270909 e nome topic topic-1.

```
oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client();
oPubSub.GoogleCloudOptions.OAuth2.ClientId = "... your google client id...";
oPubSub.GoogleCloudOptions.OAuth2.ClientSecret = "... your google client secret...";
oPubSub.CreateTopic("pubsub-270909", "topic-1");
```

Risposta dal Server

```
{
  "name": "projects/pubsub-270909/topics/topic-1"
}
```

Pubblica un messaggio

Pubblica un nuovo messaggio nel topic appena creato.

```
oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client();
oPubSub.GoogleCloudOptions.OAuth2.ClientId = "... your google client id...";
oPubSub.GoogleCloudOptions.OAuth2.ClientSecret = "... your google client secret...";
oPubSub.Publish("pubsub-270909", "topic-1", "My First Message from sgcWebSockets.");
```

Risposta dal Server

```
{
  "messageIds": [
    "1050732082561505"
  ]
}
```

Pubblica un Messaggio con Attributi

```
oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client();
oPubSub.GoogleCloudOptions.OAuth2.ClientId = "... your google client id...";
oPubSub.GoogleCloudOptions.OAuth2.ClientSecret = "... your google client secret...";
oPubSub.Publish("pubsub-270909", "topic-1", "My First Message from sgcWebSockets.", "origin=gcloud-sample,username=sgcWebSockets");
```

Come creare una nuova sottoscrizione

Crea una nuova sottoscrizione per il progetto con id: pubsub-270909, con nome sottoscrizione subscription-1 e topic-1

```
oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client();
```



```
oPubSub.GoogleCloudOptions.OAuth2.ClientId = "... your google client id...";
oPubSub.GoogleCloudOptions.OAuth2.ClientSecret = "... your google client secret...";
oPubSub.CreateSubscription("pubsub-270909", "subscription-1", "topic-1");
```

Risposta dal Server

```
{
  "name": "projects/pubsub-270909/subscriptions/subscription-1",
  "topic": "projects/pubsub-270909/topics/topic-1",
  "pushConfig": {},
  "ackDeadlineSeconds": 10,
  "messageRetentionDuration": "604800s",
  "expirationPolicy": {
    "ttl": "2678400s"
  }
}
```

Come leggere i messaggi da una Subscription

Leggi i messaggi dalla sottoscrizione precedentemente creata.

```
oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client();
oPubSub.GoogleCloudOptions.OAuth2.ClientId = "... your google client id...";
oPubSub.GoogleCloudOptions.OAuth2.ClientSecret = "... your google client secret...";
oPubSub.pubsub.Pull("pubsub-270909", "subscription-1");
```

Risposta dal Server

```
{
  "receivedMessages": [
    {
      "ackId": "PjA-RVNEUAYWLF1GSFE3GQhoUQ5PXiM_NSAorREFC08CKF15MEorQVh0Dj4N",
      "message": {
        "data": "TXkgRmlyc3QgTWVzc2FnZSBmcm9tIHNNY1dlY1NvY2tldHMu",
        "messageId": "1050732082561505",
        "publishTime": "2020-03-14T15:25:31.505Z"
      }
    }
  ]
}
```

Il messaggio viene ricevuto codificato in Base64, quindi è necessario decodificarlo prima di leggerne il contenuto.

```
sgcBase_Helpers.DecodeBase64("TXkgRmlyc3QgTWVzc2FnZSBmcm9tIHNNY1dlY1NvY2tldHMu");
```

Google Cloud | Calendar

L'API Google Calendar consente di integrare l'applicazione con Google Calendar, creando nuovi modi per coinvolgere gli utenti. L'API Calendar consente di visualizzare, creare e modificare eventi del calendario, nonché di lavorare con molti altri oggetti relativi al calendario, come calendari o controlli di accesso.

Risorse API

Google Calendar utilizza le seguenti risorse:

- **Evento:** Un evento in un calendario contenente informazioni come il titolo, gli orari di inizio e fine e i partecipanti. Gli eventi possono essere singoli o ricorrenti. Un evento è rappresentato da una risorsa Event. La raccolta Events di un dato calendario contiene tutte le risorse evento per quel calendario.
- **Calendar:** Un calendario è una raccolta di eventi. Ogni calendario ha metadati associati, come la descrizione del calendario o il fuso orario predefinito. I metadati per un singolo calendario sono rappresentati da una risorsa Calendar. La raccolta Calendars contiene risorse Calendar per tutti i calendari esistenti.
- **CalendarList:** Un elenco di tutti i calendari nell'elenco calendari di un utente nell'interfaccia utente del Calendar. I metadati per un singolo calendario che appare nell'elenco calendari sono rappresentati da una risorsa CalendarListEntry. Questi metadati includono le proprietà specifiche dell'utente per il calendario, come il colore o le notifiche per nuovi eventi. La raccolta CalendarList contiene tutte le risorse CalendarListEntry per un determinato utente. Per una spiegazione più approfondita della differenza tra le raccolte Calendars e CalendarList, vedere Calendar e Calendar List.
- **Setting:** Una preferenza utente dall'interfaccia del Calendar, come il fuso orario dell'utente. Una singola preferenza utente è rappresentata da una risorsa Setting. La raccolta Settings contiene tutte le risorse Setting per un dato utente.
- **ACL:** Una regola di controllo degli accessi che concede a un utente (o a un gruppo di utenti) un livello di accesso specificato a un calendario. Una singola regola di controllo degli accessi è rappresentata da una risorsa ACL. La raccolta ACL per un determinato calendario contiene tutte le risorse ACL che concedono accesso a quel calendario.
- **Color:** Un colore presentato nell'interfaccia utente del Calendar. La risorsa Colors rappresenta l'insieme di tutti i colori disponibili nell'interfaccia utente del Calendar, in due gruppi: colori disponibili per gli eventi e colori disponibili per i calendari.
- **Free/busy:** Un orario in cui un calendario ha eventi pianificati è considerato "busy", un orario in cui un calendario non ha eventi è considerato "free". La risorsa Freebusy consente di eseguire query sull'insieme degli orari occupati per un determinato calendario o insieme di calendari.

Caratteristiche principali

- API client per Google Calendar completa di tutte le funzionalità V3.
- Tutti i metodi supportati dalle API possono essere chiamati tramite il client API.
- Il client effettua richieste utilizzando il protocollo HTTP/2 (*solo Enterprise Edition).
- Gestione automatica delle risposte parziali tramite PageNextToken.
- Accesso facile alle proprietà dei dati di Calendar ed Event.
- Metodi di autenticazione:
 - OAuth2: richiede l'interazione dell'utente.
 - Account di servizio (richiede la delega a livello di dominio): per servizi Windows, daemon...

Configurazione

Il componente client Google Calendar dispone delle seguenti proprietà:

OAuth2

- **GoogleCloudOptions.OAuth2.ClientId:** è il ClientID fornito da Google per l'autenticazione tramite il protocollo OAuth2.
- **GoogleCloudOptions.OAuth2.ClientSecret:** è la stringa Client Secret fornita da Google per l'autenticazione tramite il protocollo OAuth2.

- **GoogleCloudOptions.OAuth2.Scope:** è l'ambito di OAuth2; di norma non è necessario modificare il valore predefinito, a meno che non si necessiti di un accesso superiore a quello predefinito.
- **GoogleCloudOptions.OAuth2.LocalIP:** il protocollo OAuth2 richiede un server locale in ascolto per la risposta dal server di autenticazione. Questo è l'IP o il nome DNS. Per impostazione predefinita è 127.0.0.1.
- **GoogleCloudOptions.OAuth2.LocalPort:** porta di ascolto del server locale.
- **GoogleCloudOptions.OAuth2.RedirectURL:** se è necessario impostare un URL di reindirizzamento diverso da LocalPort + LocalIP, è possibile impostarlo in questa proprietà (esempio: `http://127.0.0.1:8080/oauth2`).

È possibile modificare gli Scope del client API tramite la proprietà `Scopes`, selezionando semplicemente gli scope supportati dal client.

JWT

L'accesso viene effettuato firmando le richieste tramite una chiave privata fornita da Google; questo metodo è consigliato per i servizi automatizzati o le applicazioni senza interazione dell'utente. Richiede la configurazione dell'Account di servizio con [Delega a livello di dominio](#).

- **GoogleCloudOptions.JWT.ClientEmail:** il nome dell'email del client fornito durante la creazione del nuovo service account. Nodo "client_email" nel file di configurazione JSON.
- **GoogleCloudOptions.JWT.PrivateKeyId:** l'ID della chiave privata fornito da Google. Nodo "private_key_id" nel file di configurazione JSON.
- **GoogleCloudOptions.JWT.PrivateKey:** il certificato della chiave privata fornito da Google. Nodo "private_key" nel file di configurazione JSON.
- **GoogleCloudOptions.JWT.Subject:** è l'account email dell'area di lavoro collegato all'account di servizio tramite la Delega a livello di dominio.

Opzioni TLS

Usare la proprietà **TLSOptions** per personalizzare la connessione sicura stabilita con i server Google.

- **TLSOptions.IOHandler:** seleziona lo stack TLS (OpenSSL, SChannel...).
- **TLSOptions.Version:** forza una versione specifica del protocollo TLS (ad esempio `tls1_2` o `tls1_3`).
- **TLSOptions.VerifyCertificate:** abilita o disabilita la validazione del certificato server.
- **TLSOptions.OpenSSL_Options.LibPath:** punta alla cartella che contiene le librerie OpenSSL distribuite con l'applicazione.
- **TLSOptions.SChannel_Options.UseLegacyCredentials:** abilita il flusso di credenziali legacy richiesto da alcune versioni di Windows quando si utilizza SChannel.

I seguenti snippet mostrano come configurare le opzioni TLS:

```
oCalendar = new TsgcHTTPGoogleCloud_Calendar_Client();
oCalendar.TLSOptions.IOHandler = TwstLSIOHandler.iohOpenSSL;
oCalendar.TLSOptions.Version = TwstLSVersions.tls1_3;
oCalendar.TLSOptions.VerifyCertificate = true;
oCalendar.TLSOptions.OpenSSL_Options.LibPath = oslpDefaultFolder;
```

Usi più comuni

- **Configurazione**
 - [Token di aggiornamento Google Calendar](#)
 - [Account di servizio Google Calendar](#)
- **Sincronizzazione**
 - [Sincronizzazione calendari Google Calendar](#)
 - [Sincronizzazione eventi Google Calendar](#)

Sincronizza calendari

Il componente `TsgcHTTPGoogleCloud_Calendar_Client` consente di sincronizzare i calendari tramite chiamate dirette all'API di Google o utilizzando i nostri metodi `Calendars` semplificati per sincronizzare i calendari.

Method	Parametri	Descrizione
NewCalendar	aSummary: il titolo del calendario.	Crea un nuovo calendario
DeleteCalendar	aid: identificatore del calendario.	Elimina un Calendar esistente.
UpdateCalendar	aResource: oggetto con i dati del calendario.	Aggiorna un Calendar esistente.
LoadCalendars		Carica tutti i calendari e la proprietà Calendars viene riempita con questi dati.
LoadCalendarsChanged	aSyncToken: ultimo token utilizzato per aggiornare il calendario.	Carica tutte le modifiche nei calendari dal set di token.

Il client Calendar ha una proprietà chiamata **Calendars**, dove è possibile accedere ai dati del calendario dopo aver chiamato uno qualsiasi dei metodi precedenti. Questa proprietà viene sincronizzata automaticamente.

Sincronizza eventi

Il componente TsgcHTTPGoogleCloud_Calendar_Client consente di sincronizzare gli eventi tramite chiamate dirette alle API Google o utilizzando i nostri semplici metodi Event per sincronizzare gli eventi.

Method	Parametri	Descrizione
NewEvent	aCalendarId: id del calendario. aResource: oggetto con i dati dell'evento.	Crea un nuovo Evento.
DeleteEvent	aCalendarId: id del calendario. aid: identificatore dell'evento.	Elimina un Evento esistente.
UpdateEvent	aCalendarId: id del calendario. aResource: oggetto con i dati dell'evento.	Aggiorna un evento esistente.
LoadEvents	aCalendarId: id del calendario.	Carica tutti gli eventi del calendario.
LoadEventsChanged	aCalendarId: id del calendario. aSyncToken: ultimo token utilizzato per aggiornare il calendario.	Carica tutti gli eventi del calendario dal token impostato.

È possibile accedere ai dati degli eventi tramite la proprietà **Calendars**. Selezionare uno qualsiasi dei calendari esistenti dall'elenco e accedere alla sua proprietà **Events**.

Chiamate API Google Calendar

Method	Descrizione
ACL_Delete	Elimina una regola di controllo degli accessi.
ACL_Get	Restituisce una regola di controllo degli accessi.
ACL_Insert	Crea una regola di controllo degli accessi.
ACL_List	Restituisce le regole nell'elenco di controllo accessi per il calendario.
ACL_Patch	Aggiorna una regola di controllo degli accessi. Questo metodo supporta la semantica patch.
ACL_Update	Aggiorna una regola di controllo degli accessi.
ACL_Watch	Monitora le modifiche alle risorse ACL.

Method	Descrizione
CalendarList_Delete	Rimuove un calendario dall'elenco dei calendari dell'utente.
CalendarList_Get	Restituisce un calendario dall'elenco dei calendari dell'utente.
CalendarList_Insert	Inserisce un calendario esistente nell'elenco dei calendari dell'utente.
CalendarList_List	Restituisce i calendari nell'elenco dei calendari dell'utente.
CalendarList_Patch	Aggiorna un calendario esistente nell'elenco dei calendari dell'utente. Questo metodo supporta la semantica patch.
CalendarList_Update	Aggiorna un calendario esistente nell'elenco calendari dell'utente.
CalendarList_Watch	Monitora le modifiche alle risorse CalendarList.

Method	Descrizione
Calendar_Clear	Cancella un calendario principale. Questa operazione elimina tutti gli eventi associati al calendario principale di un account.
Calendar_Delete	Elimina un calendario secondario. Utilizzare <code>calendars.clear</code> per cancellare tutti gli eventi nei calendari primari.
Calendar_Get	Restituisce i metadati di un calendario.
Calendar_Insert	Crea un calendario secondario.
Calendar_Patch	Aggiorna i metadati per un calendario. Questo metodo supporta la semantica patch.
Calendar_Update	Aggiorna i metadati di un calendario.

Method	Descrizione
Channel_Stop	Interruzione del monitoraggio delle risorse tramite questo canale.

Method	Descrizione
Color_Get	Restituisce le definizioni di colore per calendari ed eventi.

Method	Descrizione
Event_Delete	Elimina un evento.
Event_Get	Restituisce un evento.
Event_Import	Importa un evento. Questa operazione viene utilizzata per aggiungere una copia privata di un evento esistente a un calendario.
Event_Insert	Crea un evento.
Event_Instances	Restituisce le istanze dell'evento ricorrente specificato.

Event_List	Restituisce gli eventi nel calendario specificato.
Event_Move	Sposta un evento in un altro calendario, ovvero cambia l'organizzatore di un evento.
Event_Patch	Aggiorna un evento. Questo metodo supporta la semantica patch. I valori dei campi specificati sostituiscono i valori esistenti. I campi non specificati nella richiesta rimangono invariati. I campi array, se specificati, sovrascrivono gli array esistenti; questo elimina eventuali elementi dell'array precedenti.
Event_QuickAdd	Crea un evento basato su una semplice stringa di testo.
Event_Update	Aggiorna un evento.
Event_Watch	Monitorare le modifiche alle risorse Events.

Method	Descrizione
Freebusy_Query	Restituisce le informazioni di disponibilità/occupazione per un insieme di calendari.

Method	Descrizione
Settings_Get	Restituisce una singola impostazione utente.
Settings_List	Restituisce tutte le impostazioni utente per l'utente autenticato.
Settings_Watch	Controlla le modifiche alle risorse Settings.

Cambia account

Se desidera passare da un account a un altro, chiami prima il metodo **Clear**, per cancellare la sessione del calendario corrente e configurare la nuova sessione.

Google Calendar | Carica calendari

Il processo per ottenere tutti i calendari dal proprio account è molto semplice. Seguire i passaggi seguenti:

1. Chiamare il metodo **LoadCalendars**.
2. Se il metodo restituisce True, può accedere alla proprietà **Calendars** e iterare l'elenco per accedere a tutti i Calendars.

```
TsgcHTTPGoogleCloud_Calendar_Client oGoogleCalendar = new TsgcHTTPGoogleCloud_Calendar_Client();
// ... configure OAuth2 options
oGoogleCalendar.GoogleCloudOptions.OAuth2.ClientId = "google ClientId";
oGoogleCalendar.GoogleCloudOptions.OAuth2.ClientSecret = "google ClientSecret";
// ... request calendars
if (oGoogleCalendar.LoadCalendars())
{
    // ... get calendars data
    for (int i = 0; i < oGoogleCalendar.Calendars.Count; i++)
    {
        string vCalendarTitle = oGoogleCalendar.Calendars.Calendar[i].Summary;
    }
}
else
{
    throw new Exception("Error Calendar Sync");
}
```


Google Calendar | Sync Events

Il processo per ottenere tutti gli eventi da un calendario è molto semplice. È sufficiente seguire i passaggi successivi:

1. Chiamare il metodo **LoadEvents** e passare il **CalendarId** come parametro.
2. Se il metodo restituisce True, è possibile accedere alla proprietà **Calendars.Events** e iterare sull'elenco per accedere a tutti gli eventi del calendario.

```
TsgcHTTPGoogleCloud_Calendar_Client oGoogleCalendar = new TsgcHTTPGoogleCloud_Calendar_Client();
// ... configure OAuth2 options
oGoogleCalendar.GoogleCloudOptions.OAuth2.ClientId = "google ClientId";
oGoogleCalendar.GoogleCloudOptions.OAuth2.ClientSecret = "google ClientSecret";
// ... request calendars first
oGoogleCalendar.LoadCalendars();
// ... request events from first calendar
TsgcGoogleCalendarItem oCalendar = (TsgcGoogleCalendarItem)oGoogleCalendar.Calendars.Calendar[0];
if (oGoogleCalendar.LoadEvents(oCalendar.ID))
{
    // ... get events data
    for (int i = 0; i < oCalendar.Events.Count; i++)
    {
        string vEventTitle = oCalendar.Events[i].Summary;
    }
}
else
{
    throw new Exception("Error Event Sync");
}
```

Google Calendar | RefreshToken

L'API Google Calendar utilizza OAuth2 per autenticarsi con i server Google. sgcWebSockets dispone di un componente che gestisce l'intero processo di autenticazione, ma se l'applicazione viene chiusa e si tenta di riconnettersi, si hanno due opzioni:

1. Autenticarsi nuovamente utilizzando le proprie API Google
2. Utilizzare il Refresh Token (se ancora valido), in modo da evitare il processo di autenticazione.

Utilizzo di RefreshToken

La prima volta che si autentica, usi l'evento OnAuthToken per salvare il **RefreshToken** se esiste. Può salvarlo in un file INI, per esempio:

```
void OnGoogleCalendarAuthToken(object sender, string TokenType, string Token, string Data)
{
    TsgcJSON oJSON = new TsgcJSON();
    oJSON.Read(Data);
    if (oJSON.Node["refresh_token"] != null)
    {
        // Save refresh token to your preferred storage
        string refreshToken = oJSON.Node["refresh_token"].Value;
        System.IO.File.WriteAllText("refresh_token.txt", refreshToken);
    }
}
```

Poi, quando si avvia nuovamente l'applicazione, se è presente un RefreshToken, chiamare il metodo RefreshToken e passare il token come argomento (è necessario impostare prima le chiavi API di Google Calendar). Se l'operazione ha esito positivo, si accede ai server Google senza dover eseguire nuovamente l'autenticazione.

```
GoogleCalendar.RefreshToken("your refresh token here");
```

Google Cloud FCM

Firebase Cloud Messaging (FCM) è una soluzione di messaggistica multiplatforma che consente di inviare messaggi in modo affidabile senza costi.

Utilizzando FCM, è possibile notificare un'app client che sono disponibili nuove email o altri dati da sincronizzare. È possibile inviare messaggi di notifica per stimolare il re-engagement e la fidelizzazione degli utenti. Per casi d'uso come la messaggistica istantanea, un messaggio può trasferire un payload fino a 4096 byte a un'app client.

Il componente supporta l'API HTTP v1.

Authorization

Il client del componente Google FCM può effettuare il login ai server Google usando i seguenti metodi:

- **gcaOAuth2:** protocollo OAuth2
- **gcaJWT:** Token JWT.

OAuth2

Il login viene eseguito tramite un browser web in cui l'utente accede con il proprio account e autorizza le richieste FCM.

- **GoogleCloudOptions.OAuth2.ClientId:** è il ClientID fornito da Google per l'autenticazione tramite il protocollo OAuth2.
- **GoogleCloudOptions.OAuth2.ClientSecret:** è la stringa Client Secret fornita da Google per l'autenticazione tramite il protocollo OAuth2.
- **GoogleCloudOptions.OAuth2.Scope:** è l'ambito di OAuth2; di norma non è necessario modificare il valore predefinito, a meno che non si necessiti di un accesso superiore a quello predefinito.
- **GoogleCloudOptions.OAuth2.LocalIP:** il protocollo OAuth2 richiede un server locale in ascolto per la risposta dal server di autenticazione. Questo è l'IP o il nome DNS. Per impostazione predefinita è 127.0.0.1.
- **GoogleCloudOptions.OAuth2.LocalPort:** porta di ascolto del server locale.
- **GoogleCloudOptions.OAuth2.RedirectURL:** se è necessario impostare un URL di reindirizzamento diverso da LocalPort + LocalIP, è possibile impostarlo in questa proprietà (esempio: `http://127.0.0.1:8080/oauth2`).

Account di servizio

L'accesso viene effettuato firmando le richieste con una chiave privata fornita da Google. Questo metodo è consigliato per servizi automatizzati o applicazioni senza interazione dell'utente.

- **GoogleCloudOptions.JWT.ClientEmail:** il nome dell'email del client fornito durante la creazione del nuovo service account. Nodo "client_email" nel file di configurazione JSON.
- **GoogleCloudOptions.JWT.Subject:** il nome dell'email del client fornito durante la creazione del nuovo account di servizio. Nodo "client_email" nel file di configurazione JSON.
- **GoogleCloudOptions.JWT.PrivateKeyId:** l'ID della chiave privata fornito da Google. Nodo "private_key_id" nel file di configurazione JSON.
- **GoogleCloudOptions.JWT.PrivateKey:** il certificato della chiave privata fornito da Google. Nodo "private_key" nel file di configurazione JSON.

Opzioni TLS

Le connessioni Firebase Cloud Messaging possono essere ottimizzate tramite la proprietà **TLSOptions**.

- **TLSOptions.IOHandler:** scegliere lo stack TLS da utilizzare (OpenSSL, SChannel...).
- **TLSOptions.Version:** richiede una versione specifica del protocollo TLS richiesta dal Suo progetto.
- **TLSOptions.VerifyCertificate:** forza la validazione del certificato del server durante la connessione a Google.
- **TLSOptions.OpenSSL_Options.LibPath:** definisce dove sono distribuite le librerie runtime OpenSSL.
- **TLSOptions.SChannel_Options.UseLegacyCredentials:** abilitarlo quando Windows SChannel richiede la negoziazione delle credenziali legacy.

Esempio di configurazione:

```
oFCM = new TsgcHTTPGoogleCloud_FCM_Client();
oFCM.TLSOptions.IOHandler = TwstLSIOHandler.iohOpenSSL;
oFCM.TLSOptions.Version = TwstLSVersions.tls1_3;
oFCM.TLSOptions.VerifyCertificate = true;
oFCM.TLSOptions.OpenSSL_Options.LibPath = oslpDefaultFolder;
```

Quando viene creato un nuovo account di servizio, è possibile scaricare un file JSON con tutte le configurazioni. Questo file può essere elaborato dal componente FCM: è sufficiente chiamare il metodo **LoadSettingsFromFile** e passare il nome del file JSON come argomento.

Usi più comuni

- Configurazione
 - [Chiavi Google OAuth2](#)
 - Account di servizio

Google FCM Client

OAuth2

Per lavorare con l'API Google FCM, il componente FCM di `sgcWebSockets` usa OAuth2 come metodo di autenticazione predefinito, quindi per prima cosa deve impostare il suo **ClientId** e il suo **ClientSecret** dal suo account Google.

```
oFCM = new TsgcHTTPGoogleCloud_FCM_Client();
oFCM.GoogleCloudOptions.Authorization = gcaOAuth2;
oFCM.GoogleCloudOptions.OAuth2.ClientId = "... your google client id...";
oFCM.GoogleCloudOptions.OAuth2.ClientSecret = "... your google client secret...";
```

Account di servizio

Gli account di servizio richiedono la costruzione di un JWT e il suo passaggio come token di autorizzazione.

```
TsgcHTTPGoogleCloud_FCM_Client oFCM = new TsgcHTTPGoogleCloud_FCM_Client();
oFCM.GoogleCloudOptions.Authorization = gcaJWT;
oFCM.GoogleCloudOptions.JWT.ClientEmail = "...google email...";
oFCM.GoogleCloudOptions.JWT.Subject = "...google email...";
oFCM.GoogleCloudOptions.JWT.PrivateKeyId = "...private key id...";
oFCM.GoogleCloudOptions.JWT.PrivateKey = "...private key certificate...";
```

Questo è necessario per ottenere un Token Chiave di Autorizzazione da Google che verrà utilizzato per tutte le chiamate REST API.

Tutti i metodi restituiscono una risposta, che può avere esito positivo o restituire un errore.

Invia messaggio

Utilizzare il metodo `SendMessage` per inviare un messaggio di notifica. La funzione ha 2 argomenti:

1. Project: è l'ID del progetto del proprio account Google Cloud.
2. Payload: è il messaggio di testo json. Esempio:

```
{
  "message": {
    "topic": "news",
    "notification": {
      "title": "Breaking News",
      "body": "New news story available."
    },
    "data": {
      "story_id": "story_12345"
    }
  }
}
```

TsgcWebView2

TsgcWebView2 è un componente visivo VCL che integra Microsoft Edge WebView2 (Chromium). È possibile aggiungerlo a un form per incorporare un browser web moderno e completo nella propria applicazione Delphi. Supporta Delphi 7 fino a Delphi 13.

Requisiti

- Microsoft Edge WebView2 Runtime (incluso con Windows 10/11 o scaricabile da Microsoft)
- WebView2Loader.dll (incluso, da posizionare accanto all'eseguibile)
- Solo Windows

Avvio rapido

Rilasci TsgcWebView2 su un form e naviga verso un URL:

```
sgcWebView21 := TsgcWebView2.Create(Self);
sgcWebView21.Parent := Self;
sgcWebView21.Align := alClient;
sgcWebView21.DefaultURL := 'https://www.example.com';
```

Funzionalità

- **Navigazione**
 - [Navigate](#), [NavigateToString](#), [GoBack](#), [GoForward](#), [Reload](#), [Stop](#), [navigazione POST](#)
- **JavaScript**
 - [ExecuteScript \(async\)](#), [ExecuteScriptSync](#), [AddInitScript](#)
- **Gestione Cookie**
 - [GetCookies](#), [AddOrUpdateCookie](#), [DeleteCookie](#), [DeleteAllCookies](#)
- **Controllo dei download**
 - [OnDownloadStarting](#), [OnDownloadProgress](#), [OnDownloadCompleted](#)
- **Impostazioni e Configurazione**
 - [ScriptEnabled](#), [DevToolsEnabled](#), [ContextMenuEnabled](#) e altro
- **Funzionalità avanzate**
 - [Stampa](#), [Audio/Mute](#), [Certificati](#), [Favicon](#), [Host virtuale](#), [Screenshot](#)

Proprietà

- **DefaultURL**: string — URL a cui navigare dopo l'inizializzazione.
- **AutoInitialize**: Boolean — quando True, la WebView2 viene creata automaticamente quando viene allocato l'handle della finestra. Predefinito: True.
- **ZoomFactor**: Double — livello di zoom (1.0 = 100%).
- **UserDataFolder**: string — percorso per la memorizzazione di cookie, cache e permessi.
- **BrowserExecutableFolder**: stringa — percorso di una distribuzione WebView2 Runtime a versione fissa.
- **Settings**: TsgcWebView2Settings — impostazioni del browser in fase di progettazione (ScriptEnabled, DevToolsEnabled, ecc.).
- **URL**: string (sola lettura) — URL della pagina corrente.
- **DocumentTitle**: string (sola lettura) — titolo della pagina corrente.
- **CanGoBack**: booleano (sola lettura) — indica se la navigazione indietro è possibile.
- **CanGoForward**: Boolean (sola lettura) — se la navigazione in avanti è possibile.
- **Initialized**: Boolean (sola lettura) — indica se WebView2 è pronto.
- **IsMuted**: Boolean — attiva/disattiva l'audio.
- **IsDocumentPlayingAudio**: Boolean (sola lettura) — indica se la pagina sta riproducendo audio.
- **FaviconURI**: string (sola lettura) — URI della favicon della pagina corrente.
- **StatusBarText**: string (sola lettura) — testo corrente della barra di stato.
- **CookieManager**: TsgcWebView2CookieManager (sola lettura) — oggetto di gestione dei cookie.
- **WebView**: ICoreWebView2 (sola lettura) — accesso diretto all'interfaccia COM.
- **Controller**: ICoreWebView2Controller (sola lettura) — accesso diretto all'interfaccia COM.

- **Environment:** `ICoreWebView2Environment` (sola lettura) — accesso diretto all'interfaccia COM.

Metodi

- **Navigate**(aURL: string) — naviga verso un URL.
- **NavigateToString**(aHTML: string) — carica il contenuto HTML direttamente.
- **GoBack / GoForward** — naviga nella cronologia.
- **Reload / Stop** — ricarica o interrompe il caricamento.
- **ExecuteScript**(aScript: string) — esegue JavaScript in modo asincrono (risultato in `OnScriptExecuted`).
- **ExecuteScriptSync**(aScript: string): string — esegue JavaScript in modo sincrono, restituisce il risultato JSON.
- **AddInitScript**(aScript: string) — aggiunge JavaScript che viene eseguito ad ogni caricamento di pagina.
- **RemoveInitScript**(aScriptId: string) — rimuovere uno script di inizializzazione precedentemente aggiunto.
- **NavigateWithPostData**(aURI, aMethod, aPostData, aHeaders: string) — naviga con metodo HTTP personalizzato, corpo e intestazioni.
- **PrintToPdf**(aFilePath: string) — salva la pagina come PDF.
- **ShowPrintUI** — mostra la finestra di dialogo di stampa nativa.
- **CapturePreviewToFile**(aFilePath: string; aFormat: Integer) — screenshot (0=PNG, 1=JPEG).
- **OpenDevToolsWindow** — apre gli Edge DevTools.
- **OpenTaskManagerWindow** — apre il task manager di Edge.
- **SetVirtualHostNameToFolderMapping**(aHostName, aFolderPath: string; aAccessKind: Integer) — associa il nome host alla cartella locale.
- **ClearVirtualHostNameToFolderMapping**(aHostName: string) — rimuove la mappatura.
- **PostWebMessageAsString**(aMessage: string) — invia messaggio al contenuto web.
- **PostWebMessageAsJson**(aJson: string) — invia JSON al contenuto web.
- **GetProfileName**: string — ottieni il nome del profilo del browser.
- **ClearBrowsingData**(aKinds: Cardinal) — cancella tipi specifici di dati di navigazione.
- **ClearAllBrowsingData** — cancella tutti i dati di navigazione.
- **PostSharedBufferToScript**(aSharedBuffer: IUnknown; aAccess: Integer; aAdditionalDataAsJson: string) — condivide il buffer di memoria con il contenuto web.
- **InitializeWebView / FinalizeWebView** — controllo manuale dell'inizializzazione.

Eventi

- **OnInitialized** — attivato quando `WebView2` è pronto.
- **OnNavigationStarting**(aURI; alsUserInitiated, alsRedirected: Boolean; var aCancel: Boolean) — prima della navigazione.
- **OnNavigationCompleted**(alsSuccess: Boolean; aWebErrorStatus: Integer) — dopo la navigazione.
- **OnSourceChanged**(alsNewDocument: Boolean) — URL modificato.
- **OnDocumentTitleChanged**(aTitle: string) — il titolo della pagina è cambiato.
- **OnWebMessageReceived**(aSource, aWebMessageAsJson, aWebMessageAsString: string) — messaggio dal contenuto web.
- **OnNewWindowRequested**(aURI: string; alsUserInitiated: Boolean; var aHandled: Boolean) — richiesta di popup/nuova scheda.
- **OnScriptExecuted**(aErrorCode: HRESULT; aResultAsJson: string) — risultato JS asincrono.
- **OnContentLoading**(alsErrorPage: Boolean) — il caricamento del contenuto è iniziato.
- **OnHistoryChanged** — cronologia di navigazione modificata.
- **OnProcessFailed**(aKind: Integer) — processo del browser arrestato in modo anomalo.
- **OnPermissionRequested**(aURI: string; aKind: Integer; alsUserInitiated: Boolean; var aState: Integer) — richiesta di permesso.
- **OnWindowCloseRequested** — `window.close()` chiamato.
- **OnDownloadStarting**(aURI, aResultFilePath: string; var aCancel, aHandled: Boolean; var aFilePath: string) — download avviato.
- **OnDownloadProgress**(aBytesReceived, aTotalBytes: Int64) — avanzamento del download.
- **OnDownloadCompleted**(aFilePath: string; aState: Integer) — download completato.
- **OnContextMenuRequested**(aMenuItems: string; aContextKind: Integer; aLocation: TPoint; var aHandled: Boolean) — menu contestuale con clic destro.
- **OnFaviconChanged**(aFaviconURI: string) — icona della pagina modificata.
- **OnClientCertificateRequested**(aHost: string; aPort: Integer; var aHandled: Boolean) — certificato client richiesto.
- **OnServerCertificateError**(aRequestURI: string; aErrorStatus: Integer; var aAction: Integer) — errore del certificato.
- **OnStatusBarTextChanged**(aText: string) — testo della barra di stato modificato.
- **OnDOMContentLoaded** — DOM pronto.

- **OnBasicAuthRequested**(aURL: string; var aUserName, aPassword: string; var aHandled: Boolean) — autenticazione HTTP basic.
- **OnCapturePreviewCompleted**(aErrorCode: HRESULT) — acquisizione schermata completata.

TsgcWebView2 | Navigazione

TsgcWebView2 fornisce diversi metodi per navigare verso URL, caricare contenuti HTML e controllare la cronologia del browser.

Navigazione di base

Utilizzare il metodo **Navigate** per caricare un URL. Il componente genera **OnNavigationStarting** prima della richiesta e **OnNavigationCompleted** al termine.

```
// Navigate to a URL
sgcWebView21.Navigate('https://www.example.com');
```

È possibile impostare anche la proprietà **DefaultURL** in fase di progettazione o prima dell'inizializzazione. Il componente naviga verso questo URL automaticamente dopo che WebView2 è pronto.

```
// Set the default URL before initialization
sgcWebView21.DefaultURL := 'https://www.example.com';
```

Contenuto HTML

Usare **NavigateToString** per caricare contenuto HTML direttamente senza un server o un file.

```
// Load HTML content directly
sgcWebView21.NavigateToString(
  '<html><body><h1>Hello from Delphi</h1>' +
  '<p>This content was loaded with NavigateToString.</p>' +
  '</body></html>');
```

Navigazione nella cronologia

Usare **GoBack** e **GoForward** per navigare nella cronologia del browser. Verificare **CanGoBack** e **CanGoForward** prima di chiamare questi metodi.

```
// Navigate back
if sgcWebView21.CanGoBack then
  sgcWebView21.GoBack;

// Navigate forward
if sgcWebView21.CanGoForward then
  sgcWebView21.GoForward;
```

Utilizzi **Reload** per aggiornare la pagina corrente e **Stop** per annullare una navigazione in sospeso.

```
// Reload the current page
sgcWebView21.Reload;

// Stop loading
sgcWebView21.Stop;
```

Navigazione POST

Usare **NavigateWithPostData** per inviare una richiesta HTTP con un metodo, corpo e intestazioni personalizzati. Questo è utile per inviare dati di form o chiamare REST API direttamente nel browser.

```
// POST form data with custom headers
sgcWebView21.NavigateWithPostData(
  'https://api.example.com/login',
```

```
'POST',
'username=admin&password=secret',
'Content-Type: application/x-www-form-urlencoded');
```

```
// POST JSON data
sgcWebView21.NavigateWithPostData(
  'https://api.example.com/data',
  'POST',
  '{"name": "John", "age": 30}',
  'Content-Type: application/json');
```

Eventi di navigazione

Utilizzi **OnNavigationStarting** per esaminare o annullare una navigazione prima che inizi. Imposti **aCancel** su **True** per bloccare la richiesta.

```
procedure TFormMain.sgcWebView21NavigationStarting(Sender: TObject;
  const aURI: string; aIsUserInitiated, aIsRedirected: Boolean;
  var aCancel: Boolean);
begin
  // Block navigation to unwanted domains
  if Pos('ads.example.com', aURI) > 0 then
    aCancel := True;
end;
```

Utilizzi **OnNavigationCompleted** per verificare se la navigazione è riuscita o fallita.

```
procedure TFormMain.sgcWebView21NavigationCompleted(Sender: TObject;
  aIsSuccess: Boolean; aWebErrorStatus: Integer);
begin
  if aIsSuccess then
    StatusBar1.SimpleText := 'Page loaded: ' + sgcWebView21.URL
  else
    StatusBar1.SimpleText := 'Navigation failed, error: ' +
      IntToStr(aWebErrorStatus);
end;
```

Gestione nuova finestra

Quando la pagina apre un popup o un link con **target="_blank"**, viene generato l'evento **OnNewWindowRequested**. Impostare **aHandled** su **True** per impedire il popup predefinito e navigare nella stessa finestra.

```
procedure TFormMain.sgcWebView21NewWindowRequested(Sender: TObject;
  const aURI: string; aIsUserInitiated: Boolean;
  var aHandled: Boolean);
begin
  // Navigate in the same window instead of opening a popup
  aHandled := True;
  sgcWebView21.Navigate(aURI);
end;
```

TsgcWebView2 | JavaScript

TsgcWebView2 fornisce metodi per eseguire JavaScript nel contesto del browser, recuperare risultati e stabilire una comunicazione bidirezionale tra la propria applicazione Delphi e il contenuto web.

Async Execution

Utilizzi **ExecuteScript** per eseguire JavaScript in modo asincrono. Il risultato viene restituito nell'evento **OnScriptExecuted** come stringa JSON.

```
// Execute JavaScript asynchronously
sgcWebView21.ExecuteScript('document.title');
```

```
procedure TFormMain.sgcWebView21ScriptExecuted(Sender: TObject;
  aErrorCode: HRESULT; const aResultAsJson: string);
begin
  if aErrorCode = S_OK then
    ShowMessage('Result: ' + aResultAsJson)
  else
    ShowMessage('Script error: ' + IntToStr(aErrorCode));
end;
```

È possibile eseguire qualsiasi espressione JavaScript valida. Il valore restituito viene sempre serializzato come JSON.

```
// Get the number of links on the page
sgcWebView21.ExecuteScript('document.querySelectorAll("a").length');

// Modify page content
sgcWebView21.ExecuteScript(
  'document.body.style.backgroundColor = "lightyellow"');
```

Esecuzione sincrona

Utilizzare **ExecuteScriptSync** per l'esecuzione bloccante che restituisce il risultato direttamente. Questo approccio è più semplice quando si necessita del valore immediatamente, ma blocca il thread chiamante fino al completamento dello script.

```
var
  vResult: string;
begin
  // Get the page title synchronously
  vResult := sgcWebView21.ExecuteScriptSync('document.title');
  ShowMessage('Title: ' + vResult);

  // Get form field value
  vResult := sgcWebView21.ExecuteScriptSync(
    'document.getElementById("email").value');
  ShowMessage('Email: ' + vResult);
end;
```

Script di inizializzazione

Utilizzare **AddInitScript** per registrare JavaScript che viene eseguito automaticamente a ogni caricamento di pagina, prima di qualsiasi altro script sulla pagina. Questo è utile per iniettare polyfill, sovrascrivere le API del browser o configurare gestori di messaggi.

```
// Add a script that runs on every page load
sgcWebView21.AddInitScript(
  'window.addEventListener("DOMContentLoaded", function() {' +
  '  console.log("Page loaded at: " + new Date().toISOString());' +
```

```
'});');
```

```
// Override window.alert to send messages to Delphi
sgcWebView21.AddInitScript(
  'window.alert = function(msg) {' +
  '  window.chrome.webview.postMessage(msg);' +
  '});');
```

Utilizzi **RemoveInitScript** per annullare la registrazione di uno script di init aggiunto in precedenza tramite il suo ID.

Messaggistica Web

La messaggistica web fornisce comunicazione bidirezionale tra l'applicazione Delphi e il JavaScript in esecuzione nel browser. Utilizzare **PostWebMessageAsString** o **PostWebMessageAsJson** per inviare dati da Delphi a JavaScript, e gestire **OnWebMessageReceived** per ricevere dati da JavaScript.

Invio di messaggi da Delphi a JavaScript:

```
// Send a plain string message
sgcWebView21.PostWebMessageAsString('Hello from Delphi!');

// Send a JSON message
sgcWebView21.PostWebMessageAsJson('{"action":"refresh","id":42}');
```

Ricezione di messaggi in JavaScript:

```
// Add an init script to listen for messages from Delphi
sgcWebView21.AddInitScript(
  'window.chrome.webview.addEventListener("message", function(e) {' +
  '  console.log("Received from Delphi: " + e.data);' +
  '});');
```

Ricezione di messaggi da JavaScript in Delphi:

```
procedure TFormMain.sgcWebView21WebMessageReceived(Sender: TObject;
  const aSource, aWebMessageAsJson, aWebMessageAsString: string);
begin
  // aWebMessageAsString contains the plain text message
  // aWebMessageAsJson contains the JSON-serialized message
  Memo1.Lines.Add('Message from web: ' + aWebMessageAsString);
end;
```

Da JavaScript, inviare messaggi a Delphi utilizzando:

```
// JavaScript code inside the web page:
// window.chrome.webview.postMessage('Hello from JavaScript!');
// window.chrome.webview.postMessage({action: 'save', data: [1,2,3]});
```

TsgcWebView2 | Gestione dei Cookie

TsgcWebView2 espone una proprietà **CookieManager** per leggere, creare, aggiornare ed eliminare i cookie nel contesto del browser.

Ottenere i Cookie

Utilizzare **CookieManager.GetCookies** per recuperare i cookie per un determinato URI. Il metodo restituisce un elenco di oggetti cookie che è possibile iterare.

```
procedure TFormMain.ButtonGetCookiesClick(Sender: TObject);
var
  vCookies: TsgcWebView2CookieList;
  i: Integer;
begin
  vCookies := sgcWebView21.CookieManager.GetCookies(
    'https://www.example.com');
  try
    for i := 0 to vCookies.Count - 1 do
      Memo1.Lines.Add(
        vCookies[i].Name + ' = ' + vCookies[i].Value);
    finally
      vCookies.Free;
    end;
  end;
end;
```

Aggiunta di Cookie

Utilizzi **CookieManager.AddOrUpdateCookie** per creare un nuovo cookie o aggiornarne uno esistente. Specifichi come minimo il nome, il valore, il dominio e il percorso.

```
// Add a session cookie
sgcWebView21.CookieManager.AddOrUpdateCookie(
  'session_id',           // name
  'abc123def456',        // value
  '.example.com',        // domain
  '/');                  // path
```

```
// Add a cookie with an expiration date
sgcWebView21.CookieManager.AddOrUpdateCookie(
  'preferences',         // name
  'theme=dark',          // value
  '.example.com',        // domain
  '/',                  // path
  Now + 30);             // expires in 30 days
```

Eliminazione dei cookie

Utilizzare **DeleteCookie** per rimuovere un cookie specifico per nome e URI. Utilizzare **DeleteAllCookies** per cancellare tutti i cookie dal browser.

```
// Delete a specific cookie
sgcWebView21.CookieManager.DeleteCookie(
  'session_id',
  'https://www.example.com');
```

```
// Delete cookies matching a domain and path
sgcWebView21.CookieManager.DeleteCookiesWithDomainAndPath(
  'session_id',
  '.example.com',
  '/');
```

```
// Delete all cookies  
sgcWebView21.CookieManager.DeleteAllCookies;
```

TsgcWebView2 | Controllo Download

TsgcWebView2 fornisce eventi per intercettare, monitorare e controllare i download di file avviati dal browser.

Download Events

L'evento **OnDownloadStarting** viene generato quando inizia un download. È possibile annullare il download, modificare il percorso del file di destinazione o consentirne il proseguimento con il comportamento predefinito.

```
procedure TFormMain.sgcWebView21DownloadStarting(Sender: TObject;
  const aURI, aResultFilePath: string;
  var aCancel, aHandled: Boolean; var aFilePath: string);
begin
  // Log the download
  Memo1.Lines.Add('Download starting: ' + aURI);
  Memo1.Lines.Add('Default path: ' + aResultFilePath);

  // Cancel downloads of .exe files
  if Pos('.exe', LowerCase(aURI)) > 0 then
  begin
    aCancel := True;
    ShowMessage('Executable downloads are blocked.');
```

Progresso del download

L'evento **OnDownloadProgress** si attiva periodicamente durante il download, segnalando i byte ricevuti e il totale dei byte attesi.

```
procedure TFormMain.sgcWebView21DownloadProgress(Sender: TObject;
  aBytesReceived, aTotalBytes: Int64);
begin
  if aTotalBytes > 0 then
    ProgressBar1.Position :=
      Round((aBytesReceived / aTotalBytes) * 100)
  else
    ProgressBar1.Position := 0;

  LabelStatus.Caption := Format('Downloaded %d of %d bytes',
    [aBytesReceived, aTotalBytes]);
end;
```

Download completato

L'evento **OnDownloadCompleted** viene generato al termine del download. Verificare **aState** per determinare se il download è riuscito, è stato annullato o è fallito.

```
procedure TFormMain.sgcWebView21DownloadCompleted(Sender: TObject;
  const aFilePath: string; aState: Integer);
begin
  case aState of
    0: // Completed
      ShowMessage('Download complete: ' + aFilePath);
    1: // Cancelled
      ShowMessage('Download was cancelled.');
```

Percorso di download personalizzato

Impostare il parametro **aFilePath** in **OnDownloadStarting** per reindirizzare il download a una posizione personalizzata. Impostare **aHandled** su **True** per sopprimere la finestra di salvataggio predefinita.

```
procedure TFormMain.sgcWebView21DownloadStarting(Sender: TObject;  
  const aURI, aResultFilePath: string;  
  var aCancel, aHandled: Boolean; var aFilePath: string);  
begin  
  // Redirect all downloads to a custom folder  
  aFilePath := 'C:\Downloads\' + ExtractFileName(aResultFilePath);  
  aHandled := True; // suppress the default save dialog  
end;
```


TsgcWebView2 | Impostazioni

TsgcWebView2 espone una proprietà **Settings** di tipo TsgcWebView2Settings che controlla il comportamento del browser. Queste impostazioni possono essere configurate in fase di progettazione o modificate in fase di esecuzione dopo l'inizializzazione.

TsgcWebView2Settings Proprietà

- **ScriptEnabled**: Boolean — abilitare o disabilitare l'esecuzione di JavaScript. Valore predefinito: True.
- **WebMessageEnabled**: Boolean — abilita o disabilita la messaggistica web (postMessage). Predefinito: True.
- **DefaultScriptDialogsEnabled**: Boolean — abilita o disabilita le finestre di dialogo JavaScript predefinite (alert, confirm, prompt). Predefinito: True.
- **StatusBarEnabled**: Boolean — mostra o nasconde la barra di stato del browser. Predefinito: True.
- **DevToolsEnabled**: Boolean — consente o blocca l'accesso a Edge DevTools. Predefinito: True.
- **ContextMenuEnabled**: Boolean — abilita o disabilita il menu contestuale del tasto destro del browser. Predefinito: True.
- **ZoomControlEnabled**: Boolean — consente o blocca lo zoom avviato dall'utente (Ctrl+scroll, Ctrl+più/meno). Predefinito: True.
- **BuiltInErrorPageEnabled**: Boolean — mostra o nasconde la pagina di errore integrata per i fallimenti di navigazione. Predefinito: True.

Configurazione in fase di progettazione

Impostare le proprietà Settings nell'Object Inspector. Vengono applicate automaticamente dopo l'inizializzazione dell'ambiente WebView2.

```
// These values can also be set in the Object Inspector
sgcWebView21.Settings.ScriptEnabled := True;
sgcWebView21.Settings.DevToolsEnabled := False;
sgcWebView21.Settings.ContextMenuEnabled := False;
sgcWebView21.Settings.ZoomControlEnabled := False;
```

Configurazione Runtime

È possibile modificare le impostazioni in fase di esecuzione dopo l'inizializzazione del componente. Le modifiche hanno effetto immediato.

```
procedure TFormMain.sgcWebView21Initialized(Sender: TObject);
begin
    // Disable right-click menu and DevTools at runtime
    sgcWebView21.Settings.ContextMenuEnabled := False;
    sgcWebView21.Settings.DevToolsEnabled := False;

    // Disable JavaScript dialogs
    sgcWebView21.Settings.DefaultScriptDialogsEnabled := False;
end;
```

UserDataFolder

La proprietà **UserDataFolder** specifica la directory in cui il browser memorizza cookie, cache, permessi e altri dati del profilo. Ogni cartella univoca crea un profilo browser isolato.

```
// Use a custom data folder for isolated profiles
sgcWebView21.UserDataFolder := 'C:\AppData\MyApp\Profile1';
```

Imposti questa proprietà prima dell'inizializzazione (prima che il componente crei l'ambiente WebView2). Se non impostata, viene utilizzata una cartella predefinita nella directory temporanea dell'applicazione.

BrowserExecutableFolder

La proprietà **BrowserExecutableFolder** punta a una distribuzione del Runtime WebView2 a versione fissa. Utilizzarla per includere una versione specifica del browser nell'applicazione invece di dipendere dal runtime installato nel sistema.

```
// Use a fixed-version runtime bundled with the application
sgcWebView21.BrowserExecutableFolder :=
  ExtractFilePath(ParamStr(0)) + 'WebView2Runtime';
```

Impostare questa proprietà prima dell'inizializzazione. Se lasciata vuota, il componente utilizza il Runtime WebView2 Evergreen installato nel sistema.

TsgcWebView2 | Funzionalità avanzate

TsgcWebView2 fornisce l'accesso a funzionalità avanzate del browser, tra cui stampa, screenshot, controllo audio, gestione dei certificati e altro ancora.

Supporto alla stampa

Utilizzi **PrintToPdf** per salvare la pagina corrente come file PDF, oppure **ShowPrintUI** per visualizzare la finestra di dialogo di stampa nativa.

```
// Save the current page as PDF
sgcWebView21.PrintToPdf('C:\Output\page.pdf');
```

```
// Show the native print dialog
sgcWebView21.ShowPrintUI;
```

Acquisizione screenshot

Utilizzare **CapturePreviewToFile** per acquisire uno screenshot della pagina corrente. Il parametro **format** specifica il tipo di immagine: 0 per PNG, 1 per JPEG. L'evento **OnCapturePreviewCompleted** viene attivato al completamento dell'acquisizione.

```
// Capture as PNG
sgcWebView21.CapturePreviewToFile('C:\Output\screenshot.png', 0);
```

```
// Capture as JPEG
sgcWebView21.CapturePreviewToFile('C:\Output\screenshot.jpg', 1);
```

```
procedure TFormMain.sgcWebView21CapturePreviewCompleted(
  Sender: TObject; aErrorCode: HRESULT);
begin
  if aErrorCode = S_OK then
    ShowMessage('Screenshot saved.')
  else
    ShowMessage('Screenshot failed: ' + IntToStr(aErrorCode));
end;
```

Audio / Controllo Muto

Utilizzare la proprietà **IsMuted** per disattivare o attivare l'audio. Verificare **IsDocumentPlayingAudio** per rilevare se la pagina sta attualmente riproducendo audio.

```
// Toggle mute
sgcWebView21.IsMuted := not sgcWebView21.IsMuted;
```

```
// Check if audio is playing
if sgcWebView21.IsDocumentPlayingAudio then
  LabelStatus.Caption := 'Audio is playing';
```

Gestione dei Certificati

Utilizzare **OnClientCertificateRequested** per rispondere quando un server richiede un certificato client. Utilizzare **OnServerCertificateError** per gestire gli errori dei certificati TLS (ad esempio, certificati autofirmati in fase di sviluppo).

```
procedure TFormMain.sgcWebView21ClientCertificateRequested(
  Sender: TObject; const aHost: string; aPort: Integer;
  var aHandled: Boolean);
```

```
begin
    // Handle client certificate selection
    aHandled := True;
end;
```

```
procedure TFormMain.sgcWebView21ServerCertificateError(
    Sender: TObject; const aRequestURI: string;
    aErrorStatus: Integer; var aAction: Integer);
begin
    // Accept self-signed certificates in development
    // aAction: 0 = deny, 1 = allow
    aAction := 1;
end;
```

Mappatura degli host virtuali

Utilizzare **SetVirtualHostNameToFolderMapping** per mappare un hostname a una cartella locale. Questo consente al contenuto web di fare riferimento a file locali utilizzando un URL virtuale invece dei percorsi file://.

```
// Map "app.local" to a local folder
sgcWebView21.SetVirtualHostNameToFolderMapping(
    'app.local',
    'C:\MyApp\WebContent',
    0); // 0 = deny remote access

// Now navigate to local content using the virtual host
sgcWebView21.Navigate('https://app.local/index.html');
```

```
// Remove the mapping
sgcWebView21.ClearVirtualHostNameToFolderMapping('app.local');
```

Gestione del profilo

Utilizzare **GetProfileName** per recuperare il nome del profilo del browser corrente. Utilizzare **ClearBrowsingData** o **ClearAllBrowsingData** per rimuovere i dati in cache, i cookie e altri artefatti di navigazione.

```
// Get the profile name
ShowMessage('Profile: ' + sgcWebView21.GetProfileName);

// Clear all browsing data
sgcWebView21.ClearAllBrowsingData;

// Clear specific browsing data types
sgcWebView21.ClearBrowsingData($0001); // cache only
```

Autenticazione di base

L'evento **OnBasicAuthRequested** si attiva quando un server richiede l'autenticazione HTTP Basic. Fornire le credenziali e impostare **aHandled** su True.

```
procedure TFormMain.sgcWebView21BasicAuthRequested(Sender: TObject;
    const aURI: string; var aUserName, aPassword: string;
    var aHandled: Boolean);
begin
    aUserName := 'admin';
    aPassword := 'secret';
    aHandled := True;
end;
```

Menu Contestuale

L'evento **OnContextMenuRequested** viene generato quando l'utente fa clic con il pulsante destro del mouse nel browser. Imposti **aHandled** su True per sopprimere il menu predefinito e mostrare il proprio.

```
procedure TFormMain.sgcWebView21ContextMenuRequested(Sender: TObject;
  const aMenuItems: string; aContextKind: Integer;
  aLocation: TPoint; var aHandled: Boolean);
begin
  // Suppress the default context menu
  aHandled := True;

  // Show a custom popup menu at the click location
  PopupMenu1.Popup(aLocation.X, aLocation.Y);
end;
```

Favicon

La proprietà **FaviconURI** restituisce l'URI del favicon della pagina corrente. L'evento **OnFaviconChanged** si attiva quando il favicon cambia.

```
procedure TFormMain.sgcWebView21FaviconChanged(Sender: TObject;
  const aFaviconURI: string);
begin
  LabelFavicon.Caption := 'Favicon: ' + aFaviconURI;
end;
```

Barra di Stato

La proprietà **StatusBarText** restituisce il testo corrente della barra di stato (in genere l'URL di un collegamento al passaggio del mouse). L'evento **OnStatusBarTextChanged** si attiva quando il testo cambia.

```
procedure TFormMain.sgcWebView21StatusBarTextChanged(Sender: TObject;
  const aText: string);
begin
  StatusBar1.SimpleText := aText;
end;
```

Task Manager

Utilizzare **OpenTaskManagerWindow** per aprire il task manager del browser Edge, che mostra l'utilizzo di memoria e CPU per ogni processo del browser.

```
sgcWebView21.OpenTaskManagerWindow;
```

Buffer condiviso

Utilizzare **PostSharedBufferToScript** per condividere un buffer di memoria tra l'applicazione Delphi e il contenuto web per il trasferimento dati ad alte prestazioni.

```
// Share a buffer with web content
sgcWebView21.PostSharedBufferToScript(
  vSharedBuffer, // IUnknown shared buffer object
  0,             // 0 = read-only, 1 = read-write
  '{"type":"image","width":640,"height":480}');
```

Accesso COM diretto

Per scenari avanzati non coperti dall'API del componente, utilizzare le proprietà **WebView**, **Controller** e **Environment** per accedere direttamente alle interfacce COM WebView2 sottostanti.

```
var
  vWebView: ICoreWebView2;
begin
  vWebView := sgcWebView21.WebView;
  if Assigned(vWebView) then
  begin
    // Call any ICoreWebView2 method directly
```

```
end;  
end;
```

TsgcUDPClient

TsgcUDPClient implementa il Client UDP basato sulla libreria Indy.

Introduzione

TsgcUDPClient implementa il Client UDP basato sulla libreria Indy.

UDP è un protocollo senza connessione in cui non vi è garanzia che i messaggi inviati arrivino a destinazione, ma al contrario del protocollo TCP, è molto più veloce.

1. Trascini un componente **TsgcUDPClient** sul form
2. Impostare **Host** e **Port** (il valore predefinito è 80) per connettersi a un server UDP disponibile.

```
oClient = new TsgcUDPClient();  
oClient.Host = "127.0.0.1";  
oClient.Port = 80;
```

3. È possibile connettersi tramite un server proxy HTTP; è necessario definire le proprietà del proxy:

Host: nome host del server proxy.

Port: numero di porta del server proxy.

Username: nome utente per l'autenticazione, lasciare vuoto per l'accesso anonimo.

Password: password per l'autenticazione, lasciare vuoto per l'accesso anonimo.

4. Se lo desidera, può gestire gli eventi
5. Chiamare il metodo **WriteData** per inviare qualsiasi messaggio al server UDP.

Riferimento

- Proprietà — 9 proprietà disponibili sul componente.
- Metodi — 2 metodi pubblici.
- Eventi — 3 eventi gestibili.

TsgcUDPServer

TsgcUDPServer implementa il server UDP basato sulla libreria Indy.

Introduzione

TsgcUDPServer implementa il server UDP basato sulla libreria Indy.

UDP è un protocollo senza connessione in cui non vi è garanzia che i messaggi inviati raggiungano la destinazione; al contrario del protocollo TCP, è tuttavia molto più veloce.

1. Trascinare un componente **TsgcUDPServer** nel form
2. Impostare la **Porta** di ascolto.

```
oClient = new TsgcUDPServer();  
oClient.Port = 80;
```

3. Per **avviare** il server, impostare la proprietà **Active = true**.

Riferimento

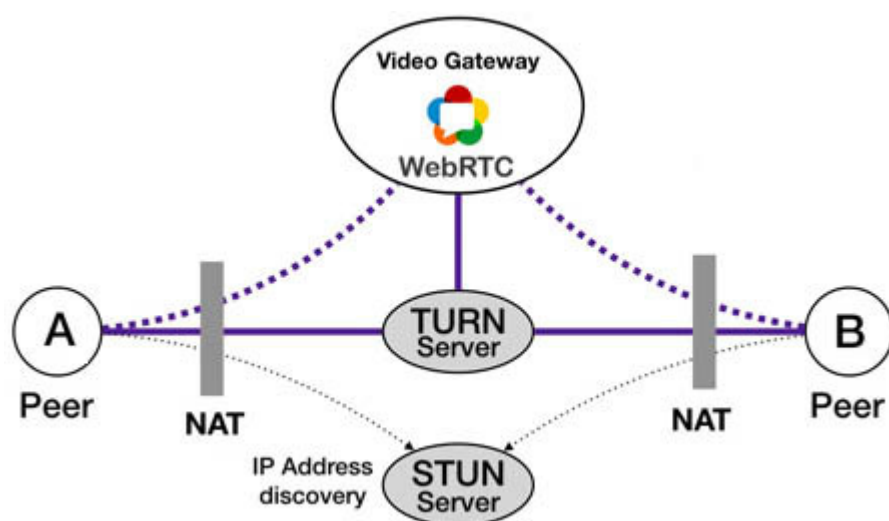
- Proprietà — 11 proprietà disponibili nel componente.
- Metodi — 7 metodi pubblici.
- Eventi — 6 eventi che è possibile gestire.

STUN

STUN (Session Traversal Utilities for NAT) è un protocollo IETF utilizzato per l'audio e il video in tempo reale nelle reti IP. STUN è un protocollo server-client, un server STUN opera solitamente sia su UDP che su TCP e rimane in ascolto sulla porta 3478.

Lo scopo principale del protocollo STUN è consentire a un dispositivo in esecuzione dietro un NAT di scoprire il proprio IP pubblico e il tipo di NAT che lo protegge.

STUN fornisce un meccanismo per comunicare tra peer dietro un NAT. I peer inviano una richiesta a un server STUN per conoscere l'indirizzo IP pubblico e la porta. Le richieste di binding inviate dal client al server vengono utilizzate per determinare le associazioni IP e porte allocate dai NAT. Il client STUN invia una richiesta Binding al server STUN; il server esamina l'IP sorgente e la porta utilizzati dal client e restituisce queste informazioni al client.



Il server STUN invia fondamentalmente due tipi di risposte: di successo o di errore. Ogni risposta ha un elenco di attributi che contengono informazioni sull'indirizzo IP di binding, il codice di errore, il motivo dell'errore, ecc.

Componenti

- **TsgcSTUNClient**: il componente client che implementa il protocollo STUN e consente di inviare richieste di binding ai server STUN.
- **TsgcSTUNServer**: il componente server che implementa il protocollo STUN.

TsgcSTUNClient

Componente *TsgcSTUNClient*.

Introduzione

TsgcSTUNClient è il client che implementa il protocollo STUN e consente di inviare richieste di binding ai server STUN.

Il componente consente di utilizzare **UDP** e **TCP** come trasporto. Quando si utilizza UDP come trasporto, implementa un **meccanismo di ritrasmissione** per reinviare le richieste se la risposta non è arrivata entro un breve intervallo di tempo.

Utilizzo di base

Di solito i server STUN sono in esecuzione sulla porta UDP 3478 e non richiedono autenticazione; pertanto, per inviare una richiesta di binding STUN, compilare le proprietà del server per consentire al client di sapere dove connettersi e gestire gli eventi in cui il componente riceverà la risposta dal server.

Configurare il server

- Host: l'indirizzo IP o il nome DNS del server, ad esempio: `stun.sgcwebsockets.com`
- Porta: la porta del server in ascolto, esempio: `3478`

Chiamare il metodo **SendRequest**, per inviare una richiesta di binding al server STUN.

Gestire gli eventi

- Se il server restituisce una risposta di successo, verrà chiamato l'evento **OnSTUNResponseSuccess** ed è possibile accedere alle informazioni di binding leggendo l'oggetto **aBinding**.
- Se il server restituisce un errore, verrà chiamato l'evento **OnSTUNResponseError** e sarà possibile accedere al codice di errore e al motivo leggendo l'oggetto **aError**.

```
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);
}
```

Riferimento

- Proprietà — 9 proprietà disponibili sul componente.
- Metodi — 3 metodi pubblici.
- Eventi — 4 eventi gestibili.

Guide

- **Bindings**
 - [Ritrasmissioni UDP](#)
 - [Credenziali a lungo termine](#)
- - [Attributes](#)

STUN Client | Ritrasmissioni UDP

Quando si esegue **STUN** su **UDP**, è possibile che il **messaggio STUN** venga **scartato** dalla rete. L'affidabilità delle transazioni richiesta/risposta STUN viene garantita attraverso la ritrasmissione del messaggio di richiesta da parte dell'applicazione client stessa.

Un client dovrebbe ritrasmettere un messaggio di richiesta STUN a partire da un intervallo di RTO ("Retransmission TimeOut"), raddoppiando dopo ogni ritrasmissione. L'RTO è una stima del tempo di andata e ritorno.

Per impostazione predefinita, lo STUN Client di sgcWebSockets è già configurato con un RTO di 500 ms e un valore Max Retries di 7.

Ad esempio, supponendo un RTO di 500 ms, le richieste verrebbero inviate agli istanti 0 ms, 500 ms, 1500 ms, 3500 ms, 7500 ms, 15500 ms e 31500 ms. Se il client non ha ricevuto una risposta dopo 39500 ms, considererà la transazione come scaduta.

```
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();
```

STUN Client | Credenziali a lungo termine

Il meccanismo delle credenziali a lungo termine si basa su una credenziale a lungo termine, sotto forma di nome utente e password condivisi tra client e server. La credenziale è considerata a lungo termine poiché si presume che venga assegnata a un utente e rimanga valida finché l'utente non è più abbonato al sistema o finché non viene modificata.

È possibile configurare le credenziali a lungo termine nel client STUN di sgcWebSockets utilizzando il seguente codice.

```
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();
```

Se il server richiede credenziali a lungo termine e le credenziali inviate dal client sono errate, il client riceverà un errore 401 Unauthorized come risposta nell'evento **OnSTUNResponseError**.

STUN Client | Attributi

Ogni volta che un server invia un messaggio al client, come risposta a un binding di richiesta, il messaggio STUN contiene un elenco di attributi con informazioni sulla risposta.

È possibile accedere a questi attributi utilizzando la classe `TsgcSTUN_Message` e la sua proprietà `Attributes`, che contiene un elenco di oggetti `TsgcSTUN_Attribute` con informazioni utili.

TsgcSTUNServer

Il server STUN può essere configurato con o senza autenticazione, può verificare l'attributo Fingerprint, inviare un server alternativo e altro ancora.

Introduzione

TsgcSTUNServer è il server che implementa il protocollo STUN e consente di elaborare le richieste di binding dai client STUN.

Il server STUN può essere configurato con o senza autenticazione, può verificare l'attributo Fingerprint, inviare un server alternativo e altro ancora.

Utilizzo di base

Di solito i server STUN vengono eseguiti sulla porta UDP 3478 e non richiedono autenticazione, quindi per configurare un server STUN imposti la porta di ascolto (per impostazione predefinita 3478) e avvii il server.

Configurare il server

- Porta: la porta del server in ascolto, esempio: 3478

Impostare la proprietà **Active = True** per avviare il server STUN.

```
TsgcSTUNServer oSTUN = new TsgcSTUNServer();  
oSTUN.Port = 3478;  
oSTUN.Active = true;
```

Riferimento

- Proprietà — 8 proprietà disponibili sul componente.
- Metodi — 2 metodi pubblici.
- Eventi — 4 eventi che è possibile gestire.

Guide

- Configurazioni
 - [Long-Term Credentials](#)
 - [Server alternativo](#)

Server STUN | Credenziali a Lungo Termine

Di solito i server STUN sono configurati senza autenticazione, quindi qualsiasi client STUN può inviare una richiesta di binding e aspettarsi una risposta dal server senza autenticazione.

Il server STUN di `sgcWebSockets` supporta le credenziali a lungo termine, quindi è possibile configurare `TsgcSTUNServer` per consentire solo richieste di binding con informazioni sulle credenziali a lungo termine.

Per configurarlo, acceda alla proprietà `STUNOptions.Authorization` e la abiliti.

Quindi acceda alla proprietà `LongTermCredentials` e la abiliti. Per impostazione predefinita, questo tipo di autorizzazione è già configurato con una stringa `Realm` e con un valore `StaleNonce` predefinito di 10 minuti (= 600 secondi).

```
TsgcSTUNServer oSTUN = new TsgcSTUNServer();
oSTUN.Port = 3478;
oSTUN.STUNOptions.Authentication.Enabled = true;
oSTUN.STUNOptions.Authentication.LongTermCredentials.Enabled = true;
oSTUN.STUNOptions.Authentication.LongTermCredentials.Realm = "sgcWebSockets";
oSTUN.STUNOptions.Authentication.LongTermCredentials.StaleNonce = 600;
oSTUN.Active = true;

private void OnSTUNRequestAuthorization(Component Sender, TsgcSTUN_Message aRequest,
    string aUsername, string aRealm, ref string Password)
{
    if ((aUsername == "my-user") && (aRealm == "sgcWebSockets"))
    {
        Password = "my-password";
    }
}
```


STUN Server | Server Alternativo

Il server alternativo rappresenta un indirizzo di trasporto alternativo che identifica un server STUN diverso che il client STUN dovrebbe provare.

Il server STUN può essere configurato per inviare un server alternativo come risposta a una richiesta di binding; per configurare questo comportamento, accedere alla proprietà `STUNOptions.BindingAttributes.AlternateServer` e configurare qui i valori richiesti.

```
TsgcSTUNServer oSTUN = new TsgcSTUNServer();
oSTUN.Port = 3478;
oSTUN.STUNOptions.BindingAttributes.AlternateServer.Enabled = true;
oSTUN.STUNOptions.BindingAttributes.AlternateServer.IPAddress = "80.54.54.1";
oSTUN.STUNOptions.BindingAttributes.AlternateServer.Port = 3478;
oSTUN.Active = true;
```

Quando il client riceve l'attributo di risposta Alternate Server, proverà a inviare una richiesta di binding al nuovo server.

TURN

Il protocollo TURN (Traversal Using Relays around NAT) consente a un server di inoltrare pacchetti di dati tra dispositivi.

Se l'indirizzo IP pubblico sia del chiamante che del destinatario non viene individuato, TURN fornisce una tecnica di fallback per instradare la chiamata tra gli endpoint.

La creazione di una sessione WebRTC è un'operazione orchestrata effettuata con l'assistenza di più server WebRTC. I server di attraversamento NAT in WebRTC hanno il compito di garantire che il traffico multimediale venga connesso correttamente. Questi server sono STUN e TURN.

Come si connettono le sessioni WebRTC

Direttamente

Se entrambi i dispositivi si trovano sulla rete locale, non è necessario alcuno sforzo particolare per connetterli tra loro. Se un dispositivo conosce l'indirizzo IP locale dell'altro dispositivo, allora possono comunicare direttamente tra loro.

Direttamente con indirizzo IP pubblico

Connessione WebRTC diretta tramite indirizzo IP pubblico ottenuto tramite il protocollo [STUN](#).

Instradamento tramite un TURN Server

Quando i peer si trovano dietro un NAT e ci sono dei Firewall, la connessione diretta non è possibile, quindi è necessario un server TURN per instradare i dati tra i peer.

Componenti

- **TsgcTURNClient**: il componente client che implementa il protocollo TURN e Le consente di allocare, creare permessi, inviare indicazioni, ecc. a un server TURN.
- **TsgcTURNServer**: il componente server che implementa il protocollo TURN.

TsgcTURNClient

Componente *TsgcTURNClient*.

Introduzione

TsgcTURNClient è il client che implementa il protocollo TURN e consente di inviare richieste di allocazione ai server TURN. Il client eredita da STUN Client, quindi tutti i metodi supportati dallo [STUN client](#) sono già supportati dal TURN Client.

Utilizzo di base

Di norma i server TURN operano sulla porta UDP 3478 e non richiedono autenticazione; per inviare una richiesta TURN è quindi sufficiente compilare le proprietà del server affinché il client sappia dove connettersi e gestire gli eventi in cui il componente riceverà la risposta dal server.

Configurare il server

- Host: l'IP o il nome DNS del server, esempio: turn.sgcwebsockets.com
- Porta: la porta del server in ascolto, esempio: 3478

Chiamare il metodo **Allocate**, per inviare una richiesta di allocazione di un indirizzo IP e una porta al server TURN.

Gestire gli eventi

- Se il server restituisce una risposta positiva, verrà richiamato l'evento **OnTURNAllocateSuccess** e sarà possibile accedere alle informazioni sull'allocazione leggendo l'oggetto **aAllocation**.
- Se il server restituisce un errore, verrà chiamato l'evento **OnSTUNResponseError** e sarà possibile accedere al codice di errore e al motivo leggendo l'oggetto **aError**.

```
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.ToString());
}

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

Riferimento

- Proprietà — 10 proprietà disponibili sul componente.
- Metodi — 9 metodi pubblici.
- Eventi — 11 eventi gestibili.

Guide

- **Allocazione**

- [Alloca indirizzo IP](#)
- [Crea permessi](#)
- **Indicazioni**
 - [Invia Indicazione](#)
- **Canali**
 - [Canali Client TURN](#)

TURN Client | Alloca indirizzo IP

Il protocollo TURN consente di utilizzare un indirizzo IP relayato per scambiare dati tra peer che si trovano dietro NAT.

Per creare un nuovo indirizzo IP Relay su un server TURN, il client deve prima chiamare il metodo **Allocate**; questo metodo invia una richiesta al server TURN per creare un nuovo indirizzo IP Relay. Se il server TURN può creare il nuovo indirizzo IP Relay, il client riceverà una risposta positiva. Il client potrà comunicare con altri peer per il tempo definito nella durata dell'allocazione.

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

private void OnTURNAllocate(Component Sender, TsgcSocketConnection aSocket, TsgcSTUN_Message aMessage,
    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);
}
```

La durata può essere aggiornata per evitare la scadenza utilizzando il metodo **Refresh**. Lifetime è il numero di secondi prima della scadenza. Se il valore è zero, l'Allocation verrà eliminata.

```
oTURN.Refresh(600);
```

TURN Client | Crea permessi

Quando una nuova Allocation viene creata in un server TURN, tale allocation non può elaborare alcun pacchetto in ingresso da altri peer se non dispone di permessi. Quindi, per consentire ad altri peer di comunicare utilizzando un Relayed IP Address, il client TURN deve prima creare i permessi per gli indirizzi IP autorizzati a scambiare dati.

Per creare una nuova Permission, chiamare il metodo **CreatePermission** e passare come parametro l'indirizzo IP del peer. Se l'IP del peer esiste già sul server TURN, verrà aggiornato; altrimenti verrà creato. Le Permission scadono dopo 5 minuti se non vengono aggiornate.

Il client TURN consente di chiamare il metodo CreatePermission solo se esiste un'allocazione attiva.

Se il permesso viene creato con successo, viene chiamato l'evento **OnTURNCreatePermission**.

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

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

TURN Client | Send Indication

Il protocollo TURN supporta 2 meccanismi per l'invio e la ricezione di dati dai peer, uno dei quali è il meccanismo Send e Data.

Il client TURN può utilizzare il metodo **SendIndication** per inviare dati al server per l'inoltro a un peer. Il client TURN deve assicurarsi che esista un'autorizzazione per l'indirizzo IP del peer a cui verrà inviata la Send Indication.

Le risposte al metodo SendIndication vengono ricevute tramite l'evento **OnTURNDataIndication**.

```
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());
}
```

TURN Client | Canali

I canali forniscono un modo per il client e il server TURN di inviare dati applicativi tramite messaggi `ChannelData`, che hanno un overhead inferiore rispetto alle indicazioni [Send e Data](#).

Prima di utilizzare i messaggi `ChannelData` per scambiare dati tra peer, il client TURN deve creare un nuovo canale; a tale scopo è sufficiente chiamare il metodo **ChannelBind** passando come parametri l'indirizzo IP e la porta del peer.

Se il server TURN riesce ad associare un nuovo canale, il client TURN riceverà una risposta positiva tramite l'evento **OnTURNChannelBind**.

```
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());
}
```

Un'associazione di canale dura 10 minuti salvo rinnovo. Per rinnovare un canale è sufficiente richiamare di nuovo il metodo **ChannelBind**.

Quando il client TURN riceve un nuovo `ChannelMessage`, viene chiamato l'evento **OnTURNChannelData**.

```
void OnTURNChannelData(Component Sender, TsgcSocketConnection aSocket,
    TsgcTURNChannelData aChannelData)
{
    DoLog("#Channel Data: [" + aChannelData.ChannelID.ToString() + "]" + " " +
        aChannelData.Data.ToString());
}
```


TsgcTURNServer

Il TURN Server supporta l'autenticazione a lungo termine, l'allocazione, i permessi, il Channel Data e altro ancora.

Introduzione

TsgcTURNServer è il server che implementa il protocollo TURN e consente di elaborare le richieste dai client TURN. Il componente eredita da [TsgcSTUNServer](#), quindi tutti i metodi e le proprietà sono disponibili su TsgcTURNServer.

Il TURN Server supporta l'autenticazione a lungo termine, l'allocazione, i permessi, il Channel Data e altro ancora.

Utilizzo di base

Di solito i server TURN sono in ascolto sulla porta UDP 3478 e richiedono credenziali Long-Term; per configurare un server TURN è quindi necessario impostare la porta di ascolto (per impostazione predefinita 3478) e avviare il server.

Configurare il server

- Porta: la porta del server in ascolto, esempio: 3478
- Definire le proprietà delle credenziali a lungo termine in `TURNOptions.Authentication.LongTermCredentials`
- Gestire `OnSTUNRequestAuthorization` per impostare la password quando un client TURN invia una richiesta al server TURN.

Impostare la proprietà **Active = True** per avviare il server TURN.

```
TsgcTURNServer oTURN = new TsgcTURNServer();
oTURN.Port = 3478;
oTURN.TURNOptions.Authentication.Enabled = true;
oTURN.TURNOptions.Authentication.LongTermCredentials.Enabled = true;
oTURN.TURNOptions.Authentication.LongTermCredentials.Realm = "esegece.com";
oTURN.Active = true;
void OnSTUNRequestAuthorization(TObject Sender, const TsgcSTUN_Message aRequest,
    const string aUsername, const string aRealm, ref string Password)
{
    if ((aUsername == "user") && (aRealm == "esegece.com"))
    {
        Password = "password";
    }
}
```

Riferimento

- Proprietà — 9 proprietà disponibili sul componente.
- Metodi — 2 metodi pubblici.
- Eventi — 11 eventi gestibili.

Guide

- Configurazioni
 - [Long-Term Credentials](#)
- Allocazioni
 - [Allocations](#)

TURN Server | Credenziali a lungo termine

Di solito i server TURN sono configurati CON autenticazione per le richieste TURN e senza autenticazione per le richieste STUN.

sgcWebSockets TURN Server supporta le credenziali a lungo termine, quindi è possibile configurare TsgcTURN-Server per consentire solo le richieste con informazioni sulle credenziali a lungo termine.

Per configurarlo, accedere alla proprietà `TURNOptions.Authorization` e abilitarla.

Quindi acceda alla proprietà `LongTermCredentials` e la abiliti. Per impostazione predefinita, questo tipo di autorizzazione è già configurato con una stringa `Realm` e con un valore `StaleNonce` predefinito di 10 minuti (= 600 secondi).

```
TsgcTURNServer oTURN = new TsgcTURNServer();
oTURN.STUNOptions.Authentication.Enabled = false;
oTURN.TURNOptions.Authentication.Enabled = true;
oTURN.TURNOptions.Authentication.LongTermCredentials.Enabled = true;
oTURN.TURNOptions.Authentication.LongTermCredentials.Realm = "sgcWebSockets";
oTURN.TURNOptions.Authentication.LongTermCredentials.StaleNonce = 600;
oTURN.Port = 3478;
oTURN.Active = true;

private void OnSTUNRequestAuthorization(TObject Sender, const TsgcSTUN_Message aRequest,
    const string aUsername, const string aRealm, ref string Password)
{
    if ((aUsername == "my-user") && (aRealm == "sgcWebSockets"))
    {
        Password = "my-password";
    }
}
```

TURN Server | Allocazioni

Tutte le operazioni TURN ruotano attorno alle allocazioni e tutti i messaggi TURN sono associati a un'Allocazione. Un'allocazione è composta da:

- L'indirizzo di trasporto inoltrato
- La quintupla: indirizzo IP del client, porta IP del client, indirizzo IP del server, porta del server e protocollo di trasporto.
- Le informazioni di autenticazione.
- Il tempo alla scadenza per ogni indirizzo di trasporto inoltrato.
- Un elenco di permessi per ciascun indirizzo di trasporto inoltrato.
- Un elenco di binding di canali per ogni indirizzo di trasporto inoltrato.

Quando un client TURN invia una richiesta **Allocate**, questo messaggio TURN viene elaborato dal server che tenta di creare un nuovo Relayed Transport Address. Per impostazione predefinita, se è disponibile una porta UDP, verrà creato un nuovo Relayed Address, ma è possibile utilizzare l'evento **OnTURNBeforeAllocate** per rifiutare una nuova richiesta di allocazione.

```
void OnTURNBeforeAllocate(TObject Sender, const TsgcSocketConnection aSocket,
    const string aIP, Word aPort, ref bool Reject)
{
    if (your own rules) == false
    {
        Reject = false;
    }
}
```

Se il processo prosegue, il server crea una nuova allocazione e viene chiamato l'evento **OnTURNCreateAllocation**. Questo evento fornisce informazioni sull'Allocation tramite la classe **TsgcTURNAllocationItem**.

```
void OnTURNCreateAllocation(TObject Sender, const TsgcSocketConnection aSocket,
    const TsgcTURNAllocationItem Allocation)
{
    DoLog("New Allocation: " + Allocation.RelayIP + ":" + IntToStr(Allocation.RelayPort));
}
```

Quando l'Allocation scade o viene eliminata ricevendo una Refresh Request dal client con lifetime pari a zero, viene attivato l'evento **OnTURNDeleteAllocation**.

```
void OnTURNDeleteAllocation(TObject Sender, const TsgcSocketConnection aSocket,
    const TsgcTURNAllocationItem Allocation)
{
    DoLog("Allocation Deleted: " + Allocation.RelayIP + ":" + IntToStr(Allocation.RelayPort));
}
```

TsgcICEClient

ICEOptions.

Introduzione

TsgcICEClient è il client che implementa il protocollo ICE e consente di inviare richieste di allocazione ai server TURN. Il client richiede il [TsgcTURNClient](#) e un [TsgcWebSocketClient](#).

Configurazione

Il client ICE ha le seguenti proprietà

- **ICEOptions.CheckList.MaxCandidates:** è il numero massimo di candidati che il client può gestire; per impostazione predefinita 100.
- **ICEOptions.GatherCandidates.STUN:** per impostazione predefinita true, otterrà i candidati utilizzando il protocollo STUN (indirizzo riflessivo).
- **ICEOptions.GatherCandidates.TURN:** per impostazione predefinita true, otterrà i candidati utilizzando il protocollo TURN (indirizzo con relay).

Il client ICE richiede un **client TURN** per raccogliere i candidati; pertanto, collegare un [TsgcTURNClient](#) alla proprietà TURN di [TsgcICEClient](#) prima di chiamare il metodo **GatherCandidates**. Per ulteriori informazioni su come configurare il [client TURN](#), consultare la documentazione.

```
TsgcICEClient oICE = new TsgcICEClient();
TsgcTURNClient oTURN = new TsgcTURNClient();
oTURN.Host = "www.esegece.com";
oTURN.Port = 3478;
oTURN.TURNOptions.Authentication.Credentials = stauLongTermCredential;
oTURN.TURNOptions.Authentication.Username = "sgc";
oTURN.TURNOptions.Authentication.Password = "secret";
oICE.GatherCandidates();
```

Riferimento

- Proprietà — 3 proprietà disponibili sul componente.
- Metodi — 3 metodi pubblici.
- Eventi — 6 eventi gestibili.

Guide

- **Candidati**
 - Raccolta candidati
 - Accoppiamento candidati

TsgcRTCPeerConnection

TsgcRTCPeerConnection è un componente client che consente di collegare peer tramite P2P attraverso UDP.

Introduzione

[*Attualmente questo componente è in fase di sviluppo]

TsgcRTCPeerConnection è un componente client che consente di connettere peer tramite P2P via UDP. Il flusso può essere suddiviso in 4 fasi:

- Signaling
- Connessione in corso
- Protezione
- Communicating

Per implementare questi passaggi, il client utilizza i seguenti protocolli:

- **WebSocket:** questo protocollo viene utilizzato per la segnalazione; i client si scambiano il Session Description Protocol e gli indirizzi IP locale, pubblico e Relayed.
- **UDP:** questo è il protocollo di trasporto; il client utilizza UDP per inviare/ricevere messaggi tra i peer.
- **DTLS:** simile a TLS, è una specifica di cifratura che protegge i messaggi tra peer, impedendo a terze parti di leggere/scrivere i messaggi.
- **STUN:** protocollo per ottenere l'indirizzo IP pubblico.
- **TURN:** protocollo per il relay degli indirizzi IP quando i peer si trovano dietro NAT.
- **ICE:** protocollo per individuare quali indirizzi IP e porte sono accessibili tra i peer.

Signaling

Quando il client si avvia non sa con chi comunicherà né di cosa parlerà. La segnalazione utilizza SDP (Session Description Protocol), che contiene dettagli come:

- IP e porte tramite cui il peer è raggiungibile.
- Certificato dell'impronta digitale utilizzato per proteggere la comunicazione.
- Utente e Password.
- ...

La segnalazione utilizza il protocollo WebSocket per lo scambio dei dati, opera tramite un sotto-protocollo ed è implementata nel componente TsgcWSPServer_RTCPeerConnection lato server.

TsgcRTCPeerConnection crea internamente già un client WebSocket con TsgcWSPClient_RTCPeerConnection allegato.

Per ottenere gli IP e le porte, il client utilizza i protocolli STUN/TURN per ottenere queste informazioni. È quindi necessario anche un server STUN/TURN.

Link:

- RTCPeerConnection WebSocket Server
- RTCPeerConnection WebSocket Client
- RTCPeerConnection STUN TURN
- Segnalazione RTCPeerConnection

Connessione in corso

Una volta che i 2 peer conoscono i candidati e gli SDP, il client utilizza un altro protocollo standard chiamato ICE.

ICE (Interactive Connection Establishment) consente di stabilire una connessione tra 2 peer. I peer possono trovarsi nella stessa rete o dietro un NAT... ICE è una soluzione per stabilire una connessione diretta senza un server centrale. Se la connessione non può essere P2P, ICE utilizzerà TURN per inviare i dati tramite un server TURN.

Una volta che ICE trova un candidato valido che può connettersi tra i due peer, il passo successivo è crittografare la comunicazione.

Link:

- `RTCPeerConnection` ICE

Protezione

Dopo che i peer si sono connessi, la comunicazione deve essere protetta. Ciò avviene tramite DTLS, un protocollo crittografico utilizzato per proteggere le comunicazioni su UDP.

Una volta elaborato con successo l'handshake DTLS, viene utilizzato un altro protocollo, SRTP (Secure Real-Time Transport Protocol); attualmente SRTP non è implementato.

Link:

- `RTCPeerConnection` DTLS

Communicating

Una volta che i 2 peer utilizzano un protocollo sicuro, la comunicazione avviene tramite 2 protocolli:

- **RTP:** Real Time Transport Protocol: utilizzato per scambiare contenuti multimediali cifrati con SRTP.
- **SCTP:** Stream Control Transmission Protocol, utilizzato per inviare e ricevere messaggi `DataChannel` cifrati con dTLS.

Attualmente questi protocolli non sono implementati, ma è possibile inviare/ricevere dati tramite DTLS su UDP.

Link:

- `RTCPeerConnection` Data

Riferimento

- Proprietà — 2 proprietà disponibili nel componente.
- Metodi — 3 metodi pubblici.
- Eventi — 14 eventi gestibili.

Demo | Chat Server

Questa demo mostra come costruire una Chat Server usando [TsgcWebSocketHTTPServer](#) e WebSockets come protocollo di comunicazione.

Ogni volta che un nuovo peer invia un messaggio, il server legge il messaggio e lo trasmette in broadcast a tutti i client connessi.

Start Server

Prima di avviare il server, deve configurarlo per impostare la porta di ascolto, se utilizzare una connessione sicura o meno...

- Prima creo una nuova istanza di [TsgcWebSocketHTTPServer](#).
- Se il Server utilizzerà connessioni sicure, è necessario un certificato PEM: impostare dove si trova questo certificato e la porta di ascolto per SSL. È possibile configurare la versione TLS e l'API OpenSSL (se necessario)

```
// ... ssl
switch (cboOpenSSLAPI.SelectedIndex)
{
    case 0:
        server.SSLOptions.OpenSSL_Options.APIVersion = TwsOpenSSLAPI.oslAPI_1_0;
        break;
    case 1:
        server.SSLOptions.OpenSSL_Options.APIVersion = TwsOpenSSLAPI.oslAPI_1_1;
        break;
}
switch (cboTLSVersion.SelectedIndex)
{
    case 0:
        server.SSLOptions.Version = TwsTLSVersions.tlsUndefined;
        break;
    case 1:
        server.SSLOptions.Version = TwsTLSVersions.tls1_0;
        break;
    case 2:
        server.SSLOptions.Version = TwsTLSVersions.tls1_1;
        break;
    case 3:
        server.SSLOptions.Version = TwsTLSVersions.tls1_2;
        break;
    case 4:
        server.SSLOptions.Version = TwsTLSVersions.tls1_3;
        break;
    default:
        break;
}
```

- Per impostazione predefinita, se si avvia il server, ascolterà su TUTTI gli IP della porta di ascolto, quindi si consiglia di utilizzare la proprietà binding per ascoltare solo su un IP specifico.

```
server.Bindings = txtHost.Text + ":" + txtDefaultPort.Text;
```

- Una volta configurate tutte le opzioni, chiamare `Server.Active = true` per avviare il server.

Configurazione degli eventi

- Utilizzare gli eventi **OnConnect** e **OnDisconnect** per sapere quando un client si connette al server.
- I **messaggi** inviati dal **client al server** vengono ricevuti nell'evento **OnMessage**, pertanto utilizzare questo gestore di eventi per trasmettere il messaggio ricevuto a tutti i client


```
private void OnMessageEvent(TsgcWSConnection Connection, string Text)
{
    server.Broadcast(Text);
}
```

Esecuzione richieste HTTP

Il WebSocket HTTP Server consente di gestire i protocolli WebSocket e HTTP sulla stessa porta di ascolto, in modo che un browser web possa richiedere una pagina web per accedere al server. **OnCommandGet** è l'evento utilizzato per leggere la richiesta HTTP e inviare le risposte HTTP.

Utilizzare il parametro `ARequestInfo` per leggere la richiesta HTTP e `AResponseInfo` per scrivere la risposta HTTP.

In sostanza, utilizzare `ARequestInfo.Document` per leggere quale documento viene richiesto dal client e inviare una risposta usando le seguenti proprietà: `ResponseNo`, `ContentType` e `ContentText`.

Esempio: un client richiede il documento `'/jquery.js'`

```
private void OnCommandGetEvent(TsgcWSConnection Connection, TsgcWSHTTPRequestInfo RequestInfo, ref TsgcWSHTTPResponseInfo ResponseInfo)
{
    if (RequestInfo.Document == "/jquery.js")
    {
        ResponseInfo.ContentType = "text/javascript";
        ResponseInfo.ContentText = File.ReadAllText(Application.StartupPath + "\\html\\jquery.js");
        ResponseInfo.ResponseNo = 200;
    }
}
```

Chat client

Questa demo mostra come costruire una chat client, utilizzando [TsgcWebSocketClient](#), che si connette a un server WebSocket, invia un messaggio e questo messaggio viene ricevuto da tutti i client connessi.

Connetti al Server

- Per prima cosa creare una nuova istanza di [TsgcWebSocketClient](#).
- Quindi configurare il **Host** e la **Port** del server.
- Se il client utilizza una connessione sicura, configurare la proprietà **TLSOptions** del componente.

```
if (chkTLS.Checked)
{
    client.Port = Int32.Parse(txtSSLPort.Text);
}
else
{
    client.Port = Int32.Parse(txtDefaultPort.Text);
}
client.Host = txtHost.Text;
switch (cboTLSIOHandler.SelectedIndex)
{
    case 0:
        client.TLSOptions.IOHandler = TwsTLSIOHandler.iohOpenSSL;
        break;
    case 1:
        client.TLSOptions.IOHandler = TwsTLSIOHandler.iohSChannel;
        break;
    default:
        break;
}
switch (cboOpenSSLAPI.SelectedIndex)
{
    case 0:
        client.TLSOptions.OpenSSL_Options.APIVersion = TwsOpenSSLAPI.oslAPI_1_0;
        break;
    case 1:
        client.TLSOptions.OpenSSL_Options.APIVersion = TwsOpenSSLAPI.oslAPI_1_1;
        break;
}
switch (cboTLSVersion.SelectedIndex)
{
    case 0:
        client.TLSOptions.Version = TwsTLSVersions.tlsUndefined;
        break;
    case 1:
        client.TLSOptions.Version = TwsTLSVersions.tls1_0;
        break;
    case 2:
        client.TLSOptions.Version = TwsTLSVersions.tls1_1;
        break;
    case 3:
        client.TLSOptions.Version = TwsTLSVersions.tls1_2;
        break;
    case 4:
        client.TLSOptions.Version = TwsTLSVersions.tls1_3;
        break;
    default:
        break;
}
client.TLS = chkTLS.Checked;
```

- Una volta configurate tutte le opzioni, impostare `Client.Active = true` per connettersi al server.

Invia messaggio al server

- Per inviare un messaggio al server, utilizzi **WriteData** ; invii un qualsiasi messaggio di testo e il server invierà come risposta lo stesso messaggio.

```
client.WriteData(txtName.Text + ": " + txtMessage.Text);
```

Ricevi Messaggi dal Server

- Ogni volta che un nuovo messaggio di testo viene ricevuto dal client, **OnMessage** viene attivato l'evento.

```
private void OnMessageEvent(TsgcWSConnection Connection, string Text)
{
    DoLog(Text);
}
```

Demos | Client

Questa demo mostra come costruire un client websocket, utilizzando [TsgcWebSocketClient](#).

Connetti al Server

- Per prima cosa creare una nuova istanza di [TsgcWebSocketClient](#).
- Quindi configurare il **Host** e la **Port** del server.
- Per impostazione predefinita il client si conatterà utilizzando il **protocollo WebSocket**. È tuttavia possibile configurare il client per la connessione tramite **protocollo TCP semplice**. Basta impostare **Specifications.RFC6455 = false**, e il client utilizzerà il protocollo TCP semplice al posto del protocollo WebSocket. Ulteriori informazioni sono disponibili nella sezione [Connessioni TCP](#).

```
client.Host = txtHost.Text;
client.Port = Int32.Parse(txtPort.Text);
client.Options.Parameters = txtParameters.Text;
client.TLS = chkTLS.Checked;
client.TLSOptions.Version = TwstlsVersions.tls1_2;
client.TLSOptions.IOHandler = TwstlsIOHandler.iohSChannel;
client.Specifications.RFC6455 = chkOverWebSocket.Checked;
client.Proxy.Enabled = chkProxy.Checked;
client.Proxy.Username = txtProxyUsername.Text;
client.Proxy.Password = txtProxyPassword.Text;
client.Proxy.Host = txtProxyHost.Text;
if (txtProxyPort.Text != "")
{
    client.Proxy.Port = Int32.Parse(txtProxyPort.Text);
}
client.Extensions.PerMessage_Deflate.Enabled = chkCompressed.Checked;
client.Active = true;
```

Eventi Client

Utilizzare i seguenti eventi per controllare il flusso del client: quando si connette, si disconnette, riceve un messaggio, viene rilevato un errore...

```
private void OnExceptionEvent(TsgcWSConnection Connection, Exception E)
{
    DoLog("#exception: " + E.Message);
}
private void OnConnectEvent(TsgcWSConnection Connection)
{
    DoLog("#connected: " + Connection.IP);
}
private void OnMessageEvent(TsgcWSConnection Connection, string Text)
{
    DoLog(Text);
}
private void OnDisconnectEvent(TsgcWSConnection Connection, int CloseCode)
{
    DoLog("Disconnected (" + CloseCode.ToString() + "): " + Connection.IP);
}
private void OnErrorEvent(TsgcWSConnection Connection, string Error)
{
    DoLog("#error: " + Connection.IP + " - " + Error);
}
```

Demo | Client MQTT

Questa demo mostra come connettersi a un server broker MQTT. Richiede un [TsgcWebSocketClient](#) per gestire i protocolli WebSocket / TCP.

Configurazione

- Prima creare una nuova istanza di [TsgcWebSocketClient](#), consultare il [Demo Client](#).
- Quindi, creare una nuova istanza di [TsgcWSPClient_MQTT](#).
- Successivamente, è necessario assegnare il protocollo MQTT al client WebSocket e configurare le opzioni di connessione nel client WebSocket.

```
if (mqtt == null)
{
    mqtt = new TsgcWSPClient_MQTT();
    mqtt.OnMQTTBeforeConnect += OnMQTTBeforeConnectEvent;
    mqtt.OnMQTTConnect += OnMQTTConnectEvent;
    mqtt.OnMQTTDisconnect += OnMQTTDisconnectEvent;
    mqtt.OnMQTTSubscribe += OnMQTTSubscribeEvent;
    mqtt.OnMQTTUnSubscribe += OnMQTTUnSubscribeEvent;
    mqtt.OnMQTTPing += OnMQTTPingEvent;
    mqtt.OnMQTTPubAck += OnMQTTPubAckEvent;
    mqtt.OnMQTTPubComp += OnMQTTPubCompEvent;
    mqtt.OnMQTTPublish += OnMQTTPublishEvent;
    mqtt.OnMQTTPubRec += OnMQTTPubRecEvent;
    mqtt.Client = client;
}
mqtt.Client = client;
txtParameters.Text = "/";
chkTLS.Checked = false;
mqtt.Authentication.Enabled = false;
mqtt.Authentication.UserName = "";
mqtt.Authentication.Password = "";
mqtt.MQTTVersion = TwsMQTTVersion.mqtt311;
mqtt.HeartBeat.Interval = 5;
mqtt.HeartBeat.Enabled = true;
switch (Index)
{
    case 0: // esegece.com
        txtHost.Text = "www.esegece.com";
        txtPort.Text = "15675";
        txtParameters.Text = "/ws";
        mqtt.Authentication.Enabled = true;
        mqtt.Authentication.UserName = "sgc";
        mqtt.Authentication.Password = "sgc";
        chkOverWebSocket.Checked = true;
        break;
    case 1: // test.mosquitto.org
        txtHost.Text = "test.mosquitto.org";
        txtPort.Text = "1883";
        chkTLS.Checked = false;
        chkOverWebSocket.Checked = false;
        break;
    case 2: // mqtt.fluux.io
        txtHost.Text = "mqtt.fluux.io";
        txtPort.Text = "1883";
        chkTLS.Checked = false;
        chkOverWebSocket.Checked = false;
        mqtt.MQTTVersion = TwsMQTTVersion.mqtt5;
        break;
    case 3: // broker.hivemq.com
        txtHost.Text = "broker.mqttdashboard.com";
        txtPort.Text = "8000";
        txtParameters.Text = "/mqtt";
        chkTLS.Checked = false;
        chkOverWebSocket.Checked = true;
        mqtt.MQTTVersion = TwsMQTTVersion.mqtt5;
        break;
}
```

MQTT Events

Il flusso di connessione è gestito dal componente client MQTT, pertanto è necessario gestire gli eventi MQTT per sapere quando è connesso al broker, quando viene pubblicato un nuovo messaggio, quando viene disconnesso...

```
private void OnMQTTConnectEvent(TsgcWSConnection Connection, bool Session, int ReasonCode, string ReasonName, TsgcWSMQTTConnectProperties Properties)
{
    DoLog("#MQTT Connect");
    chkTLS.Enabled = false;
    chkCompressed.Enabled = false;
    if (FMQTTSubscribeTopic != "")
    {
        mqtt.Subscribe(FMQTTSubscribeTopic);
        FMQTTSubscribeTopic = "";
    }
}

private void OnMQTTPublishEvent(TsgcWSConnection Connection, string Topic, string Text, TsgcWSMQTTPublishProperties Properties)
{
    DoLog(Topic + ": " + Text);
}

private void OnMQTTSubscribeEvent(TsgcWSConnection Connection, int PacketIdentifier, TsgcWSSUBACKS Codes, TsgcWSMQTTSubscribeProperties Properties)
{
    DoLog("#Subscribe: " + PacketIdentifier.ToString());
}

private void OnMQTTDisconnectEvent(TsgcWSConnection Connection, int ReasonCode, string ReasonName, TsgcWSMQTTDisconnectProperties Properties)
{
    DoLog("#disconnected");
    chkTLS.Enabled = true;
    chkCompressed.Enabled = true;
}
```

Demo | Client SocketIO

Questa demo mostra come connettersi a un server Socket.IO. Richiede un [TsgcWebSocketClient](#) per gestire i protocolli WebSocket / TCP.

Configurazione

- Prima creare una nuova istanza di [TsgcWebSocketClient](#), consultare il [Demo Client](#).
- Quindi, creare una nuova istanza di [TsgcWSAPI_SocketIO](#).
- Dopodiché, è necessario assegnare l'API Socket.IO al client WebSocket e configurare le opzioni di connessione nel client WebSocket.

```
if (socketio == null)
{
    socketio = new TsgcWSAPI_SocketIO();
    socketio.Client = client;
}
socketio.Client = client;
txtParameters.Text = "/";
chkTLS.Checked = true;
chkOverWebSocket.Checked = true;
txtHost.Text = "socketio-chat-h9jt.herokuapp.com";
txtPort.Text = "443";
txtParameters.Text = "/";
```

Invia Messaggi

Socket.IO utilizza **TsgcWebSocketClient** per inviare messaggi al server, quindi chiami semplicemente **WriteData** e passi come parametro il messaggio JSON al server socket.io

```
client.WriteData("42[\"new message\", \"\" + txtSocketIOMessage.Text + "\"]");
```

Ricezione Messaggi

I messaggi ricevuti man mano che il flusso della connessione viene gestito da **TsgcWebSocketClient**, quindi utilizzi questo componente per leggere i messaggi inviati dal server e per sapere se la connessione è attiva o meno.

```
private void OnMessageEvent(TsgcWSConnection Connection, string Text)
{
    DoLog(Text);
}
```

Demos | Server Monitor

Questa demo mostra come aggiornare 3 HTML Monitor utilizzando WebSocket come protocollo. Il server ha un timer interno che aggiorna casualmente i valori dei misuratori e aggiorna il valore utilizzando un messaggio websocket. Questo messaggio viene letto dal client javascript e aggiorna il valore del Gauge.

Configurazione

- Innanzitutto crei una nuova istanza di [TsgcWebSocketServer](#), consulti la [Demo Server Chat](#).
- Quindi creare un nuovo Timer e ogni 500 millisecondi aggiornare i valori di: memoria, rete o CPU. Inviare l'aggiornamento a tutti i client connessi.
- Nel client Javascript, leggere il messaggio inviato dal server e aggiornare il valore dell'indicatore.

```
<pre><code>
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Server Monitor Demo</title>
  <script src="http://127.0.0.1:5413/sgcWebSockets.js"></script>
  <script src="http://127.0.0.1:5413/esegece.com.js"></script>
  <link rel="stylesheet" href="http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.css" />
  <script src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
  <script src="http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.js"></script>
  <script type="text/javascript" src='https://www.google.com/jsapi'></script>
  <style>
    #status {
      padding: 5px;
      color: #fff;
      background: #ccc;
    }
    #status.fail {
      background: #c00;
    }
    #status.offline {
      background: #c00;
    }
    #status.online {
      background: #0c0;
    }
  </style>
  <script type="text/javascript">
    var vMemory;
    var vCpu;
    var vNetwork;
    var chart;
    var data;
    var options;
    var ws;

    vMemory=30;
    vCpu=55;
    vNetwork=68;
    google.load('visualization', '1', {packages:['gauge']});
    google.setOnLoadCallback(drawChart);
    function drawChart() {
      data = google.visualization.arrayToDataTable([
        ['Label', 'Value'],
        ['Memory', vMemory],
        ['CPU', vCpu],
        ['Network', vNetwork]
      ]);
      options = {
        width: 400, height: 120,
        redFrom: 90, redTo: 100,
        yellowFrom:75, yellowTo: 90,
        minorTicks: 5
      };
      chart = new google.visualization.Gauge(document.getElementById('chart_div'));
      chart.draw(data, options);
    }

    function updateChart() {
```



```

        data = google.visualization.arrayToDataTable([
            ['Label', 'Value'],
            ['Memory', vMemory],
            ['CPU', vCpu],
            ['Network', vNetwork]
        ]);
        chart.draw(data, options);
    }

    function subscribe(Channel)
    {
        if (document.getElementById(Channel).checked) {
            ws.subscribe(Channel);
        } else {
            ws.unsubscribe(Channel);
        }
    }

    function wsmonitor()
    {
        if ("WebSocket" in window)
        {
            ws = new SGCWS("ws://127.0.0.1:5413");
            ws.on('open', function(evt){
                document.getElementById('status').innerHTML = "Socket Open";
                document.getElementById('status').className = "online";
                ws.subscribe("memory");
                ws.subscribe("cpu");
                ws.subscribe("network");
            });
            ws.on('close', function(evt){
                document.getElementById('status').innerHTML = "Socket Closed";
                document.getElementById('status').className = "offline";
            });
            ws.on('sgcevent', function(evt){
                if (evt.channel == "memory") {
                    vMemory = parseInt(evt.message);
                } else if (evt.channel == "cpu") {
                    vCpu = parseInt(evt.message);
                } else if (evt.channel == "network") {
                    vNetwork = parseInt(evt.message);
                }
                updateChart();
            });
            ws.on('error', function(evt){
                document.getElementById('status').innerHTML = "Socket Error";
                document.getElementById('status').className = "fail";
            });
        }
    }
}
</script>
</head>
<body>
<div data-role="page" id="wsdemo_monitor">
    <div data-role="header" data-theme="b">
        <h1>Server Monitor</h1>
        <a href="#home" data-icon="home" data-iconpos="notext" data-direction="reverse" class="ui-btn-left">
    </div><!-- /header -->
    <div data-role="content">
        <h2>Press Start to Get Monitor Data</h2>
        <p id="status" class="success"></p>
        <h4>Select which data you want to receive: Memory - CPU - Network</h4>
        <a href="javascript:wsmonitor()" data-role="button" data-inline="true">Start</a>
        <div id="chart_div"></div>
        <div data-role="fieldcontain">
            <fieldset data-role="controlgroup" data-type="horizontal">
                <input type="checkbox" name="memory" id="memory" class="custom" checked="True" onclick=
                <label for="memory">Memory</label>
                <input type="checkbox" name="cpu" id="cpu" class="custom" checked="True" onclick=
                <label for="cpu">CPU</label>
                <input type="checkbox" name="network" id="network" class="custom" checked="True"
                <label for="network">Network</label>
            </fieldset>
        </div>
    </div><!-- /content -->
    <div data-role="footer" class="footer-docs" data-theme="c">
        <p>&copy; 2020 eSeGeCe.com</p>
    </div>
</div><!-- /page -->
</body>
</html>
</code></pre>

```


Demo | Snapshot del server

Questa demo mostra come inviare immagini dal server al client e come tutti i client ricevono la stessa immagine utilizzando il metodo broadcast del componente server.

Configurazione

- Prima creare una nuova istanza di [TsgcWebSocketHTTPServer](#), consultare la [Demo Chat Server](#).
- Abiliti la compressione per inviare meno byte quando il messaggio viene trasmesso ai client

```
Server.Extensions.PerMessage_Deflate.Enabled = true
```

- Poi ogni 5 secondi il server trasmette in broadcast un flusso di immagini a tutti i client connessi

```
Random r = new Random();  
int rInt = r.Next(1, 12);  
Bitmap bmp = new Bitmap(Application.StartupPath + "\\img\\" + rInt.ToString() + ".bmp");  
MemoryStream memoryStream = new MemoryStream();  
bmp.Save(memoryStream, System.Drawing.Imaging.ImageFormat.Bmp);  
server.Broadcast(memoryStream.ToArray());
```

Demo | Snapshot Client

Questa demo mostra come leggere messaggi websocket binari, utilizzando [TsgcWebSocketClient](#), che si connette a un server WebSocket e riceve uno stream che è un'immagine mostrata all'utente.

Connetti al Server

- Per prima cosa creare una nuova istanza di [TsgcWebSocketClient](#).
- Quindi configurare il **Host** e la **Port** del server.
- Abilitare la compressione per ricevere meno byte quando il messaggio viene trasmesso dal server.

```
Client.Extensions.PerMessage_Deflate.Enabled = true
```

- L'immagine inviata dal server arriva come stream, quindi utilizzi l'evento **OnBinary** per leggere le immagini.

```
private void OnBinaryEvent(TsgcWSConnection Connection, byte[] Bytes)
{
    MemoryStream stream = new MemoryStream(Bytes);
    pictureBox1.Image = new Bitmap(stream);
}
```

Demo | Caricamento file

Questa demo mostra come caricare un file dal browser web a un server utilizzando il protocollo WebSocket.

Configurazione

- Innanzitutto crei una nuova istanza di [TsgcWebSocketServer](#), consulti la [Demo Server Chat](#).
- Il file arriverà al server come stream binario, pertanto è necessario gestire l'evento **OnBinary** per leggere il file.

```
private void OnBinaryEvent(TsgcWSConnection Connection, byte[] Bytes)
{
    File.WriteAllBytes(filename, Bytes);
    DoLog("File Received: " + filename);
}
```

- Se desidera conoscere il nome del file, può inviare un messaggio di testo prima dell'invio del file con il nome del file

```
private void OnMessageEvent(TsgcWSConnection Connection, string Text)
{
    if (Text.StartsWith("uploadfile:") == true)
    {
        string[] filePath = Text.Split(':');
        filename = filePath[1];
    }
    else
    {
        DoLog("Message Received (" + Connection.Guid + "): " + Text);
    }
}
```

Il codice **javascript** per inviare un file tramite **websocket** è mostrato di seguito:

```
<script type='text/javascript'>
var ws;
function DoOpen()
{
    if ("WebSocket" in window)
    {
        ws = new sgcWebSocket("ws://127.0.0.1:5418");
        ws.on('open', function(evt){
            ws.binaryType = "arraybuffer";
            document.getElementById('status').innerHTML = "Socket Open";
            document.getElementById('status').className = "online";
        });
        ws.on('close', function(evt){
            document.getElementById('status').innerHTML = "Socket Closed";
            document.getElementById('status').className = "offline";
        });
        ws.on('error', function(evt){
            document.getElementById('status').innerHTML = "Socket Error";
            document.getElementById('status').className = "fail";
        });
    }
}
function DoClose()
{
    ws.close();
}
function DoUploadFile() {
    var file = document.getElementById('filename').files[0];
    var reader = new FileReader();
    var rawData = new ArrayBuffer();
}
```

```
reader.onloadend = function() {  
  }  
  reader.onload = function(e) {  
    ws.send("uploadfile:" + file.name);  
    rawData = e.target.result;  
    ws.send(rawData);  
    document.getElementById('status').innerHTML = "File Uploaded";  
    document.getElementById('status').className = "online";  
  }  
  reader.readAsArrayBuffer(file);  
}  
</script>
```

Demo | Autenticazione Server

Questa demo mostra come utilizzare l'autenticazione server; se si desidera saperne di più sui diversi tipi di autenticazione, leggere il seguente articolo sull'[Autenticazione](#).

Autenticazione

- Prima creare una nuova istanza di [TsgcWebSocketServer](#). Abilitare la proprietà Authentication, `server.Authentication.Enabled = true`;
- Quindi, verificare nel gestore dell'evento **OnAuthentication** se il nome utente e la password sono corretti. Se sono corretti, impostare la proprietà `Authenticated` su `true`, altrimenti su `false`.

```
private void OnAuthenticationEvent(TsgcWSCConnection Connection, string User, string Password, ref bool Authenticated)
{
    if ((User == "user") && (Password == "1234"))
    {
        Authenticated = true;
    }
}
```

Demos | KendoUI_Grid

Questa demo mostra come funziona la griglia KendoUI utilizzando WebSocket come protocollo e un browser Web come client. Fondamentalmente è una griglia JavaScript che viene aggiornata ogni volta che uno dei client apporta una modifica; queste modifiche vengono propagate tramite il protocollo WebSocket a tutti i client connessi, in modo che tutti possano vedere in tempo reale gli stessi dati, incluse tutte le modifiche apportate dai client.

Configurazione

- Prima creare una nuova istanza di [TsgcWebSocketHTTPServer](#), consultare la [Demo Chat Server](#).
- Dovrà quindi gestire OnCommandGet per inviare i file richiesti dai client browser web.

```
if (RequestInfo.Document == "/")
{
    string readText = File.ReadAllText(Application.StartupPath + "\\files\\index.html");
    readText = readText.Replace("<#port>", txtDefaultPort.Text);
    readText = readText.Replace("<#host>", txtHost.Text);
    ResponseInfo.ContentText = readText;
    ResponseInfo.ResponseNo = 200;
}
else
{
    ResponseInfo.ResponseNo = 404;
}
```

Aggiornamenti WebSockets

Quando un client aggiorna un record della griglia, questa modifica viene trasmessa a tutti i client connessi tramite il protocollo WebSocket. Utilizzare l'evento OnMessage per ricevere notifiche sulle modifiche alla griglia. I messaggi sono in formato JSON, quindi è sufficiente leggere il testo JSON, decodificarlo e inviare una risposta all'altro peer.

```
private void OnMessageEvent(TsgcWSConnection Connection, string Text)
{
    DoLog(Text);
    var oJSON = JObject.Parse(Text);
    if (oJSON["type"].ToString() == "read")
    {
        JArray oArray = new JArray();
        for (int i = 0; i < 20; i++)
        {
            JObject oObject = new JObject();
            oObject.Add("ContactID", i.ToString());
            oObject.Add("ContactName", ContactName[i]);
            oObject.Add("ContactTitle", ContactTitle[i]);
            oObject.Add("CompanyName", CompanyName[i]);
            oObject.Add("Country", Country[i]);
            oArray.Add(oObject);
        }
        oJSON.Add("data", oArray);
        Connection.WriteData(oJSON.ToString());
    }
    else if (oJSON["type"].ToString() == "update")
    {
        Text = Text.Replace("\"type\": \"update\"", "\"type\": \"push-update\"");
        server.Broadcast(Text, "", "", Connection.Guid);
    }
    else if (oJSON["type"].ToString() == "destroy")
    {
        Text = Text.Replace("\"type\": \"destroy\"", "\"type\": \"push-destroy\"");
        server.Broadcast(Text, "", "", Connection.Guid);
    }
    else if (oJSON["type"].ToString() == "create")
    {
        Text = Text.Replace("null", DateTime.Now.ToString("yyyyMMddHHmmss"));
        string vText = Text.Replace("\"type\": \"create\"", "\"type\": \"push-create\"");
        server.Broadcast(vText, "", "", Connection.Guid);
        Connection.WriteData(Text);
    }
}
```



```
} }
```

Demo | ServerSentEvents

Questa demo mostra come funzionano gli eventi Server-Sent nel WebSocket Server. `sgcWebSockets` consente al server di gestire più di un protocollo sulla stessa porta di ascolto.

È possibile leggere ulteriori informazioni sugli [Eventi Inviati dal Server](#).

Questa demo mostra come il server invierà ogni secondo l'ora a tutti i client connessi utilizzando Server Sent Events.

Una volta avviato il server, **invia in broadcast** a tutti i client connessi un messaggio con l'**Ora del Server**, in modo che ogni volta che il client riceve questo messaggio, lo mostri all'utente.

```
private void timer1_Tick(object sender, EventArgs e)
{
    server.Broadcast("data: " + "Server Time: " + DateTime.Now.ToString("HH:mm:ss"));
}
```

Il **codice javascript** per gestire la connessione websocket è mostrato di seguito:

```
socket = new sgcWebSocket('sse', '', 'sse');
socket.on('open', function(evt){
    document.getElementById('status').innerHTML = "Socket Open";
    document.getElementById('status').className = "online";
});
socket.on('close', function(evt){
    document.getElementById('status').innerHTML = "Socket Closed";
    document.getElementById('status').className = "offline";
});
socket.on('message', function(evt){
    document.getElementById('log').innerHTML = evt.message;
});
socket.on('error', function(evt){
    document.getElementById('status').innerHTML = "Socket Error";
    document.getElementById('status').className = "fail";
});
```

Demo | Server WebRTC

Questa demo mostra come costruire un Video Conference Server utilizzando [TsgcWebSocketHTTPServer](#) e WebRTC come libreria javascript.

La demo utilizza il protocollo WebSocket per segnalare WebRTC e usa server STUN/TURN pubblici; per i siti in produzione è necessario utilizzare i propri server STUN/TURN. Gli utenti registrati possono scaricare Coturn per Windows, già compilato e con tutte le librerie necessarie per essere eseguito sui propri server.

Il client deve essere un browser web con supporto per le connessioni WebRTC.

Configurazione

- Prima creare una nuova istanza di [TsgcWebSocketHTTPServer](#), consultare la [Demo Chat Server](#).
- Quindi, creare una nuova istanza di [TsgcWSPServer_WebRTC](#).
- Dopo di ciò, è necessario assegnare il Protocollo WebRTC al server WebSocket e configurare l'host e la porta del server.

```
server.DocumentRoot = Application.StartupPath + "\\html";  
server.Port = Int32.Parse(txtDefaultPort.Text);  
server.Active = true;
```

La demo richiede una pagina HTML di indice utilizzata per inviare la pagina principale WebRTC; questa pagina è fornita con la demo.

Esecuzione nel WebBrowser

Una volta configurato il server, avviarlo e selezionare uno dei browser web disponibili. Si aprirà una nuova sessione del browser web chiedendo di avviare una nuova sessione. Se l'operazione ha esito positivo, verrà visualizzato il proprio video e, aprendo lo stesso URL in un altro browser web, si vedranno entrambi i peer connessi.

La demo viene eseguita per impostazione predefinita senza SSL, il che è valido solo per le connessioni localhost. Per i siti in produzione, utilizzare connessioni SSL. Consultare la [Demo Chat Server](#) per configurare SSL sul lato server.

Demo | Server AppRTC

Questa demo mostra come costruire un Server per Videoconferenze usando [TsgcWebSocketHTTPServer](#) e AppRTC come libreria javascript.

La demo utilizza il protocollo WebSocket per segnalare WebRTC e usa server STUN/TURN pubblici; per i siti in produzione è necessario utilizzare i propri server STUN/TURN. Gli utenti registrati possono scaricare Coturn per Windows, già compilato e con tutte le librerie necessarie per essere eseguito sui propri server.

Il client deve essere un browser web con supporto per le connessioni WebRTC.

Configurazione

- Prima creare una nuova istanza di [TsgcWebSocketHTTPServer](#), consultare la [Demo Chat Server](#).
- Quindi, creare una nuova istanza di [TsgcWSPServer_AppRTC](#).
- Successivamente, è necessario assegnare il protocollo AppRTC al server WebSocket e configurare l'host e la porta del server. WebRTC richiede connessioni sicure, quindi sarà necessario utilizzare un certificato PEM e configurare la proprietà SSLOptions del componente.

```
server.DocumentRoot = Application.StartupPath + "\\html";
server.Port = Int32.Parse(txtDefaultPort.Text);
AppRTC.AppRTC.RoomLink = "https://" + txtHost.Text + ":" + txtDefaultPort.Text + "/r/";
AppRTC.AppRTC.WebSocketURL = "wss://" + txtHost.Text + ":" + txtDefaultPort.Text;
server.Active = true;
```

- AppRTC.RoomLink è l'url verso il quale il browser web verrà reindirizzato per accedere a una room
- AppRTC.WebSocketURL è l'url della connessione websocket
- Gli IceServer possono essere configurati nel protocollo AppRTC Server.

La demo richiede una pagina HTML indice utilizzata per distribuire la pagina principale AppRTC; questa pagina è fornita con la demo.

Esecuzione nel WebBrowser

Una volta configurato il server, avviarlo e selezionare uno dei browser web disponibili. Si aprirà una nuova sessione del browser Web richiedendo di partecipare a una nuova stanza. Entrare in questa stanza e, in caso di successo, verrà visualizzato un link che deve essere utilizzato da un altro browser web per avviare una nuova videoconferenza.

AppRTC

Please enter a room name.

959454873

JOIN

RANDOM

Recently used rooms:

Demo | Client Telegram

Questa demo mostra come connettersi a Telegram, ricevere tutti i contatti, inviare messaggi di testo, inviare immagini... e molto altro

Configurazione

- Prima creare una nuova istanza di [TsgcTDLib_Telegram](#).
- Quindi, prima di tentare la connessione a Telegram, è necessario passare alcuni parametri al componente client come API Hash, API Id... Una volta impostati tutti i parametri obbligatori, impostare la proprietà `Active = true` per avviare una connessione.

```
telegram.Telegram.API.ApiHash = txtApiHash.Text;
telegram.Telegram.API.ApiId = txtApiId.Text;
telegram.Telegram.PhoneNumber = "";
telegram.Telegram.BotToken = "";
if (chkLoginBot.Checked)
{
    telegram.Telegram.BotToken = txtBotToken.Text;
}
else
{
    telegram.Telegram.PhoneNumber = txtPhoneNumber.Text;
}

telegram.Active = true;
```

- Quando il client tenta di connettersi a Telegram, di solito è richiesto un codice, pertanto è necessario gestire **OnTelegramAuthenticationCode** e restituire il parametro `Code` con il valore fornito dal proprio account Telegram.

```
private void OnAuthenticationCodeEvent(TsgcTDLib_Telegram Sender, ref string Code)
{
    Code = InputBox("Telegram", "Introduce Telegram Code");
}
```

Invia Messaggi Telegram

Per inviare un messaggio Telegram (testo, file, immagini...) è necessario impostare prima il `Chald` in cui si desidera inviare il messaggio e quindi il parametro che può essere un messaggio di testo, un nome file...

```
// send text message
sgcTelegram.SendTextMessage("456413", "Hello From sgcWebSockets!!!");

// send file message
sgcTelegram.SendDocumentMessage("383784", "c:\\yourfile.txt");
```

Ricezione di messaggi Telegram

I messaggi ricevuti dal client Telegram vengono gestiti da specifici gestori di eventi. Esiste un evento per quando viene ricevuto un nuovo messaggio di testo, quando viene ricevuto un nuovo documento, una foto...

```
private void OnMessageTextEvent(TsgcTDLib_Telegram Sender, TsgcTDLib_Telegram_Client.TsgcTelegramMessageText Mess
{
```

```
DoLogMessage(MessageText.ChatId, MessageText.SenderUserId.ToString(), MessageText.Text);
}

private void OnMessageDocumentEvent(TsgcTDLib_Telegram Sender, TsgcTDLib_Telegram_Client.TsgcTelegramMessageDocun
{
    DoLogMessage(MessageDocument.ChatId, MessageDocument.SenderUserId.ToString(), MessageDocument.FileName);
}
```

Coturn

Coturn

A partire da sgcWebSockets 4.5.2 edizione ENTERPRISE, è possibile creare il proprio server STUN/TURN utilizzando Delphi/CBuilder.

È un'implementazione open source gratuita di server TURN e STUN.

Il TURN Server è un server e gateway di NAT traversal per il traffico multimediale VoIP. Può essere utilizzato anche come server e gateway TURN per traffico di rete generico.

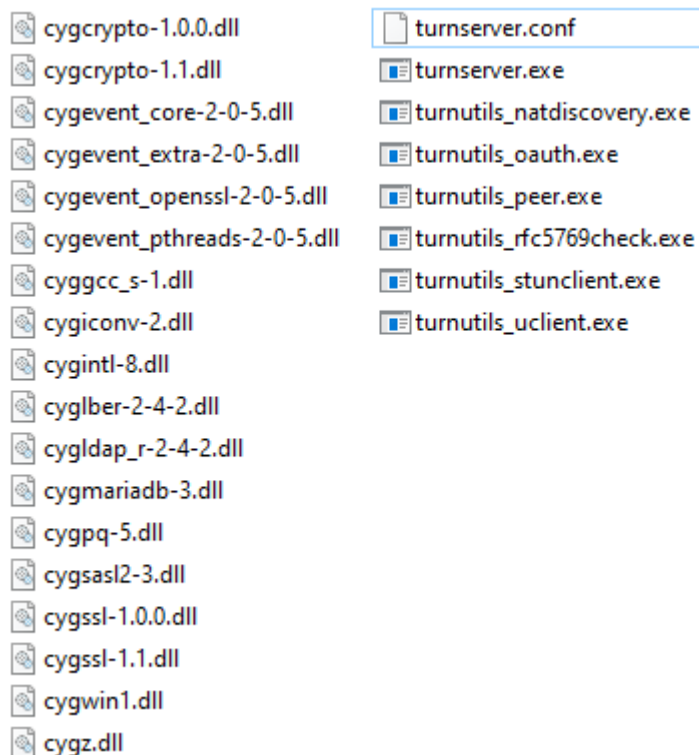
Le piattaforme di destinazione del progetto supportate sono:

- **Linux**
- **Mac OS X**
- **Windows** (Cygwin): i binari compilati sono disponibili per gli utenti registrati.

Configurazione Windows

Prima è necessario scaricare i binari compilati dal proprio account; sono disponibili 2 versioni: win32 e win64. Selezionare la piattaforma desiderata e decomprimere i binari in una cartella. Verranno creati i seguenti file:

1. Alcune librerie cygwin sono necessarie per eseguire l'applicazione; è necessario distribuire queste librerie insieme al server coturn.
2. Alcune applicazioni console:
 - 2.1 turnserver.exe: è la principale applicazione console per eseguire un server TURN/STUN
 - 2.2 Altre applicazioni: vengono utilizzate per scopi di configurazione o test.
3. Turnserver.conf: è il file di configurazione per il server coturn.



turnserver.conf

Questo è il file di configurazione per il server coturn; aprendolo si vedrà una configurazione predefinita.

Configurazione semplice

Il Suo server ha il seguente IP pubblico 80.15.44.123 ed è in ascolto sulla porta 80. Le credenziali per la connessione sono: username = demo, password = secret
Impostare la seguente configurazione:

```
listening-ip=80.15.44.123
listening-port=80
realm=yourrealm.com
user=demo:secret
```

Configurazione con TLS abilitato

Il server ha il seguente IP pubblico 80.15.44.123 e ascolta sulla porta 80 e 443 (connessioni TLS). Le credenziali per la connessione sono: username = demo, password = secret. Il nome del certificato (deve essere in formato PEM) è certificate.crt e la chiave privata è private.key.
Impostare la seguente configurazione:

```
listening-ip=80.15.44.123
listening-port=80
realm=yourrealm.com
tls-listening-port=443
cert=certificate.crt
pkey=private.key
user=demo:secret
```

Sono disponibili ulteriori configurazioni, basta aprire turnserver.conf e leggere le sezioni documentate.

Eeguire coturn

Una volta configurato, è possibile avviare il server eseguendo turnserver.exe; verrà aperta una nuova applicazione console e verrà creato un file di log. È possibile aumentare la verbosità dell'applicazione console (ottenere messaggi più dettagliati) abilitando "verbose" nel file turnserver.conf.

```
FC 5766.
0: IPv6: On this platform, I am using alternative behavior of TTL <HOPLIMIT> according to RFC 6156.
0: IPv4: On this platform, I am using alternative behavior of TOS according to RFC 5766.
0: WARNING: I cannot support STUN CHANGE_REQUEST functionality because only one IP address is provided
0: Wait for relay ports initialization...
0: relay 46.183.117.249 initialization...
0: relay 46.183.117.249 initialization done
0: Relay ports initialization done
0: IO method <general relay thread>: poll
0: turn server id=0 created
0: IO method <udp listener/relay thread>: poll
0: IO method <general relay thread>: poll
0: turn server id=1 created
0: turn server id=128 created
0: IO method <udp listener/relay thread>: poll
0: turn server id=129 created
0: Total UDP servers: 2
0: Total General servers: 2
0: IO method <auth thread>: poll
0: IO method <auth thread>: poll
0: IO method <admin thread>: poll
```

Licenza

Contratto di licenza per l'utente finale dei componenti eSeGeCe

Il Contratto di licenza con l'utente finale ("EULA") dei componenti eSeGeCe ("eSeGeCe") è un accordo legale tra Lei (sia come persona fisica che come entità giuridica) e l'Autore di eSeGeCe per tutti i componenti eSeGeCe che possono includere componenti software associati, supporti, materiali stampati e documentazione "online" o elettronica ("componenti eSeGeCe"). Installando, copiando o utilizzando in altro modo i componenti eSeGeCe, Lei accetta di essere vincolato dai termini di questo EULA. Il presente contratto di licenza rappresenta l'intero accordo relativo al programma tra Lei e l'Autore di eSeGeCe (di seguito denominato "LICENSER") e sostituisce qualsiasi proposta, rappresentazione o accordo precedente tra le parti. Se non accetta i termini di questo EULA, non installi né utilizzi i componenti eSeGeCe.

I componenti eSeGeCe sono protetti dalle leggi sul copyright e dai trattati internazionali sul copyright, nonché da altre leggi e trattati sulla proprietà intellettuale. I componenti eSeGeCe sono concessi in licenza, non venduti.

Se si desidera il CODICE SORGENTE è necessario pagare la quota di registrazione. È assolutamente vietato cedere le chiavi di licenza e/o le edizioni complete di eSeGeCe (incluse le edizioni DCU e le edizioni con sorgente) a terzi individui e/o entità. È altresì vietato utilizzare le chiavi di licenza e/o le edizioni complete di eSeGeCe provenienti da terzi individui e/o entità.

1. CONCESSIONE DELLA LICENZA

I componenti eSeGeCe sono concessi in licenza come segue:

(a) Installazione e Utilizzo.

IL LICENZIANTE le concede il diritto di installare e utilizzare copie dei componenti eSeGeCe sul computer che esegue una copia regolarmente autorizzata del sistema operativo per cui i componenti eSeGeCe sono stati progettati [ad es., Windows 2000, Windows 2003, Windows XP, Windows ME, Windows Vista, Windows 7, Windows 8, Windows 10].

(b) Esenzione da royalty.

È possibile creare applicazioni commerciali basate sui componenti eSeGeCe e distribuirle con i propri eseguibili, senza royalty.

(c) Modifiche (solo edizioni Source).

È possibile apportare modifiche, miglioramenti, opere derivate e/o estensioni al CODICE SORGENTE concesso in licenza fornito secondo i termini stabiliti nel presente contratto di licenza.

(d) Copie di backup.

È inoltre possibile effettuare copie dei componenti eSeGeCe nel caso in cui siano necessarie per scopi di backup e archiviazione.

2. DESCRIZIONE DI ALTRI DIRITTI E LIMITAZIONI

(a) Conservazione degli avvisi di copyright.

Non è consentito rimuovere o alterare alcun avviso di copyright su qualsiasi copia dei componenti eSeGeCe.

(b) Distribuzione.

Non è consentito distribuire copie registrate dei componenti eSeGeCe a terze parti. Le edizioni di valutazione disponibili per il download dai siti web ufficiali di eSeGeCe possono essere distribuite liberamente.

È possibile creare componenti/controlli ActiveX/librerie che includono i componenti eSeGeCe per le proprie applicazioni, ma NON è consentito distribuirli o pubblicarli a terze parti.

(c) Divieto di distribuzione del CODICE SORGENTE (solo edizioni Source).

NON deve distribuire o pubblicare il CODICE SORGENTE, né alcuna modifica, miglioramento, opera derivata e/o estensione, in forma di CODICE SORGENTE a terze parti.

NON deve distribuire, pubblicare, divulgare o rendere in altro modo disponibile a terzi alcuna parte del CODICE SORGENTE.

(d) Divieto di reverse engineering, decompilazione e disassemblaggio.

Non è consentito decodificare, decompilare o disassemblare i componenti eSeGeCe, salvo e solo nella misura in cui tale attività sia espressamente consentita dalla legge applicabile nonostante questa limitazione.

(e) Noleggio.

Non è consentito affittare, cedere in leasing o prestare i componenti eSeGeCe.

(f) Servizi di supporto.

LICENSER può fornire servizi di supporto relativi ai componenti eSeGeCe ("Servizi di supporto"). Qualsiasi codice software supplementare fornito come parte dei Servizi di supporto sarà considerato parte dei componenti eSeGeCe e soggetto ai termini e alle condizioni di questo EULA.

eSeGeCe è concessa in licenza per essere utilizzata da un solo sviluppatore alla volta. E il supporto tecnico verrà fornito a un solo specifico sviluppatore.

(g) Conformità alle leggi applicabili.

È necessario rispettare tutte le leggi applicabili riguardanti l'utilizzo dei componenti eSeGeCe.

3. RISOLUZIONE

Fatti salvi eventuali altri diritti, LICENSER può risolvere il presente EULA se l'utente non rispetta i termini e le condizioni del presente EULA. In tal caso, l'utente deve distruggere tutte le copie dei componenti eSeGeCe in suo possesso.

4. COPYRIGHT

Tutti i titoli, inclusi ma non limitati ai diritti d'autore, sui componenti eSeGeCe e qualsiasi loro copia sono di proprietà del LICENZIANTE o dei suoi fornitori. Tutti i titoli e i diritti di proprietà intellettuale sui contenuti a cui si può accedere tramite l'uso dei componenti eSeGeCe sono di proprietà del rispettivo proprietario del contenuto e possono essere protetti da applicabili diritti d'autore o da altre leggi e trattati sulla proprietà intellettuale. Il presente EULA non Le concede alcun diritto di utilizzo di tali contenuti. Tutti i diritti non espressamente concessi sono riservati al LICENZIANTE.

5. ASSENZA DI GARANZIE

IL LICENZIANTE esclude espressamente qualsiasi garanzia per i componenti eSeGeCe. I componenti eSeGeCe sono forniti "così come sono" senza alcuna garanzia espressa o implicita di alcun tipo, incluse, a titolo esemplificativo, garanzie di commerciabilità, non violazione o idoneità a uno scopo particolare. IL LICENZIANTE non garantisce né assume responsabilità per l'accuratezza o la completezza di qualsiasi informazione, testo, grafica, link o altro elemento contenuto nei componenti eSeGeCe. IL LICENZIANTE non fornisce garanzie riguardo a eventuali danni causati dalla trasmissione di un virus informatico, worm, time bomb, logic bomb o altro programma informatico simile. IL LICENZIANTE esclude inoltre espressamente qualsiasi garanzia o dichiarazione agli Utenti Autorizzati o a terzi.

6. LIMITAZIONE DI RESPONSABILITÀ

In nessun caso il LICENZIANTE sarà responsabile per qualsiasi danno (inclusi, senza limitazione, perdita di profitti, interruzione dell'attività commerciale o perdita di informazioni) derivante dall'utilizzo o dall'impossibilità di utilizzo dei componenti eSeGeCe da parte degli "Utenti Autorizzati", anche se il LICENZIANTE è stato informato della possibilità di tali danni. In nessun caso il LICENZIANTE sarà responsabile per la perdita di dati o per danni indiretti, speciali, incidentali, consequenziali (inclusa la perdita di profitti) o altri danni basati su contratto, illecito o altro. IL LICENZIANTE non avrà alcuna responsabilità riguardo al contenuto dei componenti eSeGeCe o di qualsiasi loro parte, inclusi ma non limitati agli errori od omissioni in essi contenuti, diffamazione, violazioni dei diritti di pubblicità, privacy, diritti di marchio, interruzione dell'attività commerciale, lesioni personali e perdita di privacy, diritti morali o divulgazione di informazioni riservate.

Indice

- Aggiungi proxy Telegram [583](#)
- Allocazioni Server TURN [888](#)
- ALPN [68](#)
- Amazon SQS [820](#)
- AMQP Consuma Messaggi [260](#)
- AMQP Exchange [254](#)
- AMQP QoS [263](#)
- AMQP Transactions [264](#)
- AMQP1 Invio Messaggio [284](#)
- AMQP1 Lettura messaggio [286](#)
- AMQP1 Links [278](#)
- AMQP1 Receiver Links [282](#)
- AMQP1 Sessions [276](#)
- Anthropic [633](#)
- Anthropic Batches [642](#)
- Anthropic Citations [649](#)
- Anthropic Documents [645](#)
- Anthropic Extended Thinking [643](#)
- Anthropic Files [653](#)
- Anthropic Messages [637](#)
- Anthropic Models [641](#)
- Anthropic Prompt Caching [647](#)
- Anthropic Structured Outputs [651](#)
- Anthropic Tool Use [640](#)
- Anthropic Vision [639](#)
- API [342](#), [367](#), [368](#), [375](#), [378](#), [403](#), [421](#), [477](#), [519](#), [524](#), [568](#)
- API 3Commas [477](#)
- API Binance [344](#), [360](#), [367](#)
- API Binance Futures [360](#), [367](#)
- API Binance Futures Trade [367](#)
- API Bitfinex [445](#)
- API Bitget [532](#)
- API Bitmex [436](#)
- API Bybit [492](#)
- API Cex [506](#)
- API Cex Plus [514](#)
- API Coinbase [370](#)
- API Coinbase Pro [370](#)
- API Crypto.com [539](#)
- API Deribit [536](#)
- API Discord [519](#)
- API Forex [497](#)
- API GatelIO [534](#)
- API HTX [541](#)
- API Kraken [391](#), [393](#), [398](#), [401](#), [403](#), [406](#), [408](#), [415](#), [421](#), [423](#)
- API Kraken Futures [406](#), [408](#), [415](#), [421](#), [423](#)
- API Kraken Futures REST Private [423](#)
- API Kraken Futures REST Public [421](#)
- API Kraken REST Private [403](#)
- API Kraken REST Public [401](#)
- API Kucoin [449](#)
- API Kucoin Futures [463](#)
- API MEXC [524](#)
- API MEXC Futures [528](#)
- API OKX [482](#)
- API OpenAI [522](#)
- API Pusher [428](#)
- API SignalR [388](#)
- API SignalRCore [382](#)
- API SocketIO [368](#)
- API Telegram [568](#)
- API Whatsapp [543](#)
- API XTB [488](#)
- APIs [344](#), [352](#), [355](#), [360](#), [370](#), [382](#), [388](#), [391](#), [393](#), [398](#), [401](#), [406](#), [408](#), [415](#), [423](#), [428](#), [436](#), [445](#), [497](#), [503](#), [506](#), [528](#)
- APNs [721](#), [722](#), [723](#)
 - Certificate-Based Connection [723](#)
 - Connessione basata su token [722](#)
- Apple Push Notifications [719](#)
- Attributi client STUN [874](#)
- Autenticazione [51](#), [103](#)
- Autenticazione Client [103](#)
- Autenticazione Client AMQP1 [272](#)
- Autenticazione provider OAuth2 [785](#)
- Autenticazione WebAuthn [807](#)
- Authentication [128](#)
- Authorization Code con PKCE (RFC 7636) [740](#)
- Authorization Code Grant [765](#)
- Authorization Code Grant (RFC 6749) [738](#)

- Auto Ban [176](#)
- Autorizzazione WebAuthn [813](#)
- Avvio Server [114](#)
- Barre di prezzo [502](#)
- Binance Connect [352](#)
- Binance Ottieni dati di mercato [354](#)
- Binance Private REST API [355](#)
- Binance Trade Spot [356](#)
- Binance Withdraw [359](#)
- Binary Message [100](#)
- Bindings [60](#), [115](#)
- Bitmex Connect WebSocket API [440](#)
- Bitmex Sottoscrivi Canale WebSocket [441](#)
- Bot [579](#)
- Broadcast [59](#)
- Brute Force [176](#)
- Build [30](#)
- Canali [59](#), [252](#), [353](#)
- Canali AMQP [252](#)
- Caricamento file [905](#)
- Certificate-Based Connection [723](#)
 - APNs [723](#)
- Certificates SChannel [97](#)
- Certificati autofirmati [159](#)
- Certificati CA propri OpenSSL [42](#)
- Certificati OpenSSL [96](#)
- Certificati OpenSSL P12 [44](#)
- Channels [376](#), [504](#), [884](#)
- Chat client [894](#)
- CIDR [174](#)
- Client [87](#), [89](#), [90](#), [98](#), [99](#), [100](#), [103](#), [106](#), [107](#), [108](#), [109](#), [134](#), [234](#), [236](#), [237](#), [250](#), [251](#), [329](#), [329](#), [710](#), [711](#), [713](#), [714](#), [872](#), [873](#), [874](#), [881](#), [882](#), [883](#), [884](#), [894](#), [894](#), [896](#), [897](#), [899](#), [904](#), [914](#)
 - Invia file [329](#)
- Client AMQP Connect [250](#)
- Client AMQP Disconnect [251](#)
- Client AMQP1 Azure MessageBus [273](#)
- Client AMQP1 Connect [268](#)
- Client AMQP1 Connection State [271](#)
- Client Apertura della connessione [87](#)
- Client Authentication [714](#)
- Client Chiusura connessione [89](#), [710](#)
- Client Connections [127](#)
- Client Credentials Grant [767](#)
- Client Credentials Grant (RFC 6749) [742](#)
- Client Exceptions [106](#)
- Client Invia Messaggio Binario [99](#)
- Client Javascript Webauthn [816](#)
- Client Keep Connection Active [711](#)
- Client Mantieni Connessione Aperta [90](#)
- Client MQTT Connect [234](#)
- Client MQTT Sessions [236](#)
- Client MQTT Version [237](#)
- Client OAuth2 per applicazioni desktop [735](#)
- Client Send Text [98](#), [100](#)
- Client Send Text Message [98](#)
- Client Snapshots [904](#)
- Client SocketIO [899](#)
- Client Telegram [914](#)
- Clients [713](#)
- Clients [236](#), [250](#), [711](#), [897](#)
- Code [71](#)
- Code AMQP [256](#)
- Coinbase Pro — Recupera dati di mercato [377](#)
- Coinbase Pro Connect [375](#)
- Coinbase Pro Place Orders [380](#)
- Coinbase Pro Private REST API [378](#)
- Coinbase Pro SandBox Account [381](#)
- Coinbase Pro Subscribe [376](#)
- Come effettuare un ordine su Bitmex [442](#)
- Compressione [63](#)
- Connect Mosquitto [235](#)
- Connect WebSocket Server [86](#)
- Connessione basata su token [722](#)
 - APNs [722](#)
- Connessione Client AMQP1 con Timeout di Inattività [270](#)
- Connessioni sicure [53](#)
- Connessioni TCP [75](#)
- Connessioni TIME_WAIT [93](#)
- Connettere il server TCP [92](#)
- Connetti al Server Sicuro [95](#)
- Connettore MCP Anthropic [654](#)
- Conto Forex [501](#)
- Cookie WebView2 [857](#)
- Corpo risposta [145](#)
- Coturn [916](#)
- Creazione App WhatsApp [547](#)

- Credenziali a lungo termine del server TURN [887](#)
- Cross-Site Scripting [178](#)
- CryptoHopper [586](#)
- Dati di mercato Forex [502](#)
- DeepSeek [658](#)
- DeepSeek Messaggi [660](#)
- DeepSeek Vision [661](#)
- Deflate-Frame [598](#)
- Device Authorization Grant (RFC 8628) [746](#), [773](#)
- Disconnessione client AMQP1 [269](#)
- Disconnessioni Abbandonate [91](#)
- DPoP - Demonstrating Proof of Possession (RFC 9449) [748](#)
- Edizioni [21](#)
- Elicitation del server MCP [612](#)
- Endpoint del server [760](#)
- Errore [145](#)
- Esempio Server [755](#)
- Estensioni [596](#)
- File [61](#), [81](#), [328](#), [329](#), [330](#), [707](#), [905](#)
- File di protocollo [324](#)
- Files [138](#)
- Filtraggio IP [174](#)
- Filtraggio Messaggi [178](#)
- Fired [165](#)
- Firewall Blacklist [174](#)
- Firewall Whitelist [174](#)
- Flash [64](#)
- Flood Protection [181](#)
- Flusso [29](#)
 - Threading [29](#)
- Forex Connect [499](#), [503](#)
- Forex Subscribe [504](#)
- Forex Trade [500](#)
- Found [579](#)
- Funzionalità avanzate WebView2 [863](#)
- Gemini [656](#)
- Gemini CountTokens [670](#)
- Gemini EmbedContent [671](#)
- Gemini Embeddings [671](#)
- Gemini Function Calling [672](#)
- Gemini Models [668](#)
- Gemini Structured Outputs [669](#)
- Gemini Token Counting [670](#)
- Gemini Tool Use [672](#)
- Gemini Vision [667](#)
- Generate [720](#)
 - Remote Notification APNs [720](#)
- Google Calendar [838](#), [844](#), [845](#), [846](#)
- Google Calendar RefreshToken [846](#)
- Google Calendar Sync Calendars [844](#)
- Google Calendar Sync Events [845](#)
- Google Cloud FCM [847](#)
- Google Cloud Pub [829](#)
- Google OAuth2 Keys [823](#)
- Grok [679](#)
- Grok Vision [681](#)
- Gruppi [65](#)
- Handshake WebSocket client [107](#)
- HeartBeat [55](#)
- HighPerfAcceptsPerWorker [156](#)
- HTTP [27](#), [58](#), [69](#), [137](#), [138](#), [145](#), [146](#), [160](#), [701](#), [705](#)
- HTTP Dispatch Files [138](#)
- HTTP Server Stream Video [148](#)
- HTTP/2 [139](#), [140](#), [142](#), [706](#), [707](#), [708](#), [709](#), [712](#), [715](#)
- HTTP/2 Alternate Service [142](#)
- HTTP/2 Download file [707](#)
- HTTP/2 Headers [709](#)
- HTTP/2 Reason Disconnection [712](#)
- HTTP/2 Risposte parziali [708](#)
- HTTP/2 Server Push [140](#), [706](#)
- HTTP1 [725](#)
- HTTP2 [702](#)
- HTTPAPI [158](#), [160](#), [161](#), [165](#)
- HTTPAPI Custom Headers [161](#)
- HTTPAPI Disable HTTP [160](#)
- HTTPAPI FineTune [156](#)
- HTTPAPI Invia risposta di testo [162](#)
- HTTPAPI Invia Risposta File [163](#)
- HTTPAPI OnDisconnect [165](#)
- HTTPAPI URL Reservation [154](#)
- Impostazioni WebView2 [861](#)
- In HTML [759](#)
- Indy SChannel [120](#), [149](#)
- Inoltro richieste HTTP [69](#)
- Installazione [22](#)
- Introduzione [19](#)
- Invia file [328](#), [329](#)

- Client [329](#)
- Server [328](#)
- Invia file ai client [329](#)
- Invia file al server [328](#)
- Invia il messaggio fattura Telegram [581](#)
- Invia messaggio Telegram con pulsanti [576](#)
- Invia messaggio Telegram con pulsanti in-line [576](#)
- Invia messaggio Telegram in grassetto [578](#)
- Inviare file di grandi dimensioni tramite POST [61](#)
- Inviare un messaggio Telegram con pulsanti [577](#)
- Invio [721](#)
 - Richieste di notifica [721](#)
- Invio di file tramite WhatsApp [565](#)
- Invio File di Grandi Dimensioni [330](#)
- IOCP [67](#)
- IoT [689](#), [690](#), [697](#)
- IoT Amazon MQTT Client [690](#)
- IoT Azure MQTT Client [697](#)
- IP Ban [176](#)
- JavaScript WebView2 [855](#)
- JWT [788](#)
- KendoUI_Grid [908](#)
- Kucoin Connect WebSocket API [456](#)
- Kucoin Futures Connessione WebSocket API [469](#)
- Kucoin Futures Private Requests Time [476](#)
- Kucoin Futures Recupero dei dati di mercato [471](#)
- Kucoin Futures REST API Privata [472](#)
- Kucoin Futures Subscribe WebSocket Channel [470](#)
- Kucoin Futures Trade [473](#)
- Kucoin Get Market Data [458](#)
- Kucoin Private REST API [459](#)
- Kucoin Subscribe WebSocket Channel [457](#)
- Kucoin Trade Spot [460](#)
- Licenza [918](#)
- Lightstreamer [503](#), [504](#)
- Link Sender AMQP1 [279](#)
- LoadBalancing [80](#)
- Log [57](#)
- MaxConnectionsPerIP [181](#)
- MaxMessagesPerSec [181](#)
- MCP [599](#)
- MCP Server Resources [607](#)
- MCP Server Sampling [611](#)
- MCP Server Tools [601](#)
- Messaggi Frammentati [83](#)
- Messaggi Gemini [665](#)
- Messaggi Grok [680](#)
- Messaggi Ollama [676](#)
- Messaggi sponsorizzati di Telegram [580](#)
- Metodo [705](#)
- Mistral AI [683](#)
- Mistral Embeddings [688](#)
- Mistral Messages [685](#)
- Modelli DeepSeek [662](#)
- Modelli Grok [682](#)
- Modelli Mistral [687](#)
- Modelli Ollama [677](#)
- Moderazione OpenAI [625](#)
- Monitor server [900](#)
- MQTT [234](#), [235](#), [236](#), [237](#), [238](#), [239](#), [241](#), [242](#), [243](#), [244](#), [245](#), [690](#), [697](#), [897](#)
- MQTT Cancella Messaggi Conservati [245](#)
- MQTT Pubblicare un messaggio [242](#)
- MQTT Publish [238](#), [242](#), [244](#)
- MQTT Publish Subscribe [238](#)
- MQTT Receive Messages [243](#)
- MQTT Subscribe [241](#)
- MQTT Topics [239](#)
- Navigazione WebView2 [853](#)
- OAuth2 [715](#), [729](#), [755](#), [759](#), [760](#), [761](#), [762](#), [763](#), [764](#), [823](#)
- OAuth2 Client for Web Applications [734](#)
- OAuth2 Customize Sign [759](#)
- OAuth2 Nessun URL di Autenticazione [764](#)
- OAuth2 Provider Private Endpoints [784](#)
- OAuth2 Recupero Access Token [762](#)
- OAuth2 Registra App [761](#)
- Ollama [674](#)
- Ollama Embeddings [678](#)
- ompClassic [156](#)
- ompHighPerf [156](#)
- OpenAI [614](#)
- OpenAI Audio [624](#), [632](#)
- OpenAI Batch [630](#)
- OpenAI Chat [622](#)
- OpenAI Completion [621](#)

- OpenAI Edit [623](#)
- OpenAI Fine-Tuning [629](#)
- OpenAI RealTime [626](#)
- OpenAI Responses [627](#)
- OpenAI Speech [628](#)
- OpenAI Uploads [631](#)
- OpenSSL [35](#), [38](#), [40](#), [96](#)
- OpenSSL Load Additional Functions [46](#)
- OpenSSL OSX [40](#)
- OpenSSL Verify Certificate [45](#)
- OpenSSL Windows [38](#)
- OperatingMode [156](#)
- Orario delle richieste private Binance [358](#)
- Orders [500](#)
- Ordini [501](#)
- Panoramica [23](#)
- Parametri di connessione WebSocket [48](#)
- PDF-Back-Cover [927](#)
- PDF-Front-Cover [1](#)
- PerMessage-Deflate [597](#)
- Posizioni [501](#)
- Prompt del server MCP [604](#)
- Protocol AMQP [246](#)
- Protocol AMQP1 [266](#)
- Protocol MQTT [231](#)
- Protocol Presence [331](#), [335](#)
- Protocol Presence Javascript [335](#)
- Protocol STOMP [287](#)
- Protocol WAMP2 [308](#)
- Protocol WebRTC [293](#)
- Protocolli [108](#), [226](#), [228](#), [291](#), [295](#), [296](#), [300](#), [312](#), [317](#), [324](#), [335](#)
- Protocolli Javascript [228](#)
- Protocollo AppRTC [291](#)
- Protocollo Default [317](#)
- Protocollo Default Javascript [317](#)
- Protocollo E2EE [338](#)
- Protocollo predefinito [312](#)
- Protocollo WAMP [296](#), [300](#)
- Protocollo WAMP Javascript [300](#)
- Protocollo WebRTC [295](#)
- Protocollo WebRTC Javascript [295](#)
- Protocols [231](#), [246](#), [287](#), [293](#), [308](#), [331](#)
- Provider OAuth2 Azure AD [783](#)
- Proxy [82](#), [109](#), [583](#)
- Proxy Client [109](#)
- Pubblicazione Messaggi AMQP [259](#)
- Publisher WAMP [304](#)
- QueueLength [156](#)
- Queues [256](#)
- QuickStart HTTP [27](#)
- QuickStart WebSockets [25](#)
- Radici del server MCP [610](#)
- Rate Limiting [181](#)
- RCON [585](#)
- Recupero Messaggi AMQP [262](#)
- Refresh Token Grant [771](#)
- Registra [584](#)
- Registra utente Telegram [584](#)
- Registrazione WebAuthn [799](#)
- Registro protocollo client [108](#)
- Reindirizzamenti WebSocket [94](#)
- Remote Notification APNs [720](#)
- Generate [720](#)
- Resource Owner Password Credentials Grant [769](#)
- Resource Owner Password Credentials Grant (RFC 6749) [744](#)
- REST API [499](#)
- Revoca Token (RFC 7009) [775](#)
- Ricerca Web Anthropic [650](#)
- Ricevi messaggi binari [102](#), [133](#)
- Ricezione di messaggi di testo [132](#)
- Ricezione messaggi di testo [101](#)
- Richiesta di Registrazione WebAuthn [802](#)
- Richiesta HTTP [705](#)
- Richieste Client in Attesa [713](#)
- Richieste del provider OAuth2 [787](#)
- Richieste del server [137](#)
- Richieste di notifica [721](#)
- Invio [721](#)
- Risposta di autenticazione WebAuthn [810](#)
- Risposta di registrazione WebAuthn [803](#)
- Risultato autenticazione WebAuthn [811](#)
- Risultato della registrazione WebAuthn [805](#)
- Ritrasmissioni UDP client STUN [872](#)
- RTCMultiConnection [591](#)
- SChannel [120](#), [149](#)
- SChannel Server [120](#), [149](#)
- Server [114](#), [115](#), [116](#), [117](#), [118](#), [124](#), [125](#), [126](#), [128](#), [130](#), [131](#), [132](#), [133](#), [134](#), [137](#), [139](#), [146](#),

[158](#), [235](#), [328](#), [706](#), [755](#), [760](#), [763](#), [876](#), [877](#),
[887](#), [888](#), [892](#), [900](#), [903](#), [907](#), [911](#), [912](#)
 Invia file [328](#)
 Server ad alte prestazioni [31](#)
 Server AppRTC [912](#)
 Server Authentication [128](#), [763](#), [907](#)
 Server Bindings [115](#)
 Server Chat [892](#)
 Server Chiudi Connessione [126](#)
 Server Invia Messaggio Binario [131](#)
 Server Invia Messaggio di Testo [130](#)
 Server Keep Active [117](#)
 Server Leggi intestazioni [134](#)
 Server Mantieni le connessioni attive [124](#)
 Server Plain TCP [125](#)
 Server Sessions [146](#)
 Server SSL [118](#), [158](#)
 Server SSL SChannel [120](#), [149](#)
 Server Startup Shutdown [116](#)
 Server-Sent Events [78](#)
 ServerSentEvents [910](#)
 Service [142](#)
 Sessioni server MCP [600](#)
 Sicurezza WhatsApp [552](#)
 SkipIOCPOnSuccess [156](#)
 Snapshot del server [903](#)
 Sottoscrittori WAMP [303](#)
 Sottoscrizione Binance [353](#)
 SQL Injection [178](#)
 Stop Limit [500](#)
 Storico [501](#)
 STUN [869](#), [872](#), [873](#), [874](#), [876](#), [877](#)
 STUN Client Long Term Credentials [873](#)
 STUN Server Alternate Server [877](#)
 STUN Server Long Term Credentials [876](#)
 Sub [829](#)
 SubProtocol [76](#)
 Telegram - Ottieni membri del supergruppo [582](#)
 Telegram Chat [579](#)
 Tempo per richieste private Coinbase Pro [379](#)
 Tempo richieste private Kucoin [462](#)
 Test del browser Web [50](#)
 Thread del Server HTTP/2 [143](#)
 Threading [29](#)
 Flusso [29](#)
 Throttle [77](#)
 Tick di prezzo [502](#)
 Token Introspection (RFC 7662) [777](#)
 Token WhatsApp [550](#)
 Transazioni [73](#)
 TsgcHTTP_JWT_Client [790](#)
 TsgcHTTP_JWT_Server [793](#)
 TsgcHTTP_OAuth2_Client [730](#)
 TsgcHTTP_OAuth2_Client_Google [736](#)
 TsgcHTTP_OAuth2_Client_Microsoft [737](#)
 TsgcHTTP_OAuth2_Server [752](#)
 TsgcHTTP_OAuth2_Server_Provider [781](#)
 TsgcHTTP2Client [703](#)
 TsgcHTTP2ConnectionClient [716](#)
 TsgcHTTP2RequestProperty [717](#)
 TsgcHTTP2ResponseProperty [718](#)
 TsgcICEClient [889](#)
 TsgcIWebSocketClient [222](#)
 TsgcIWWSPClient_Dataset [323](#)
 TsgcIWWSPClient_sgc [316](#)
 TsgcRTCPeerConnection [890](#)
 TsgcServerHTTPAPI_FineTune [156](#)
 TsgcSTUNClient [870](#)
 TsgcSTUNServer [875](#)
 TsgcTCPClient [110](#)
 TsgcTURNClient [879](#)
 TsgcTURNServer [885](#)
 TsgcUDPClient [867](#)
 TsgcUDPServer [868](#)
 TsgcWebPush_Client [595](#)
 TsgcWebSocketClient [84](#)
 TsgcWebSocketClient_WinHTTP [166](#)
 TsgcWebSocketFirewall [167](#), [174](#), [176](#), [178](#),
[181](#)
 TsgcWebSocketHTTPServer [135](#)
 TsgcWebSocketHTTPServer_Sessions [146](#)
 TsgcWebSocketLoadBalancerServer [220](#)
 TsgcWebSocketProxyServer [221](#)
 TsgcWebSocketServer [112](#)
 TsgcWebSocketServer_HTTPAPI [152](#)
 TsgcWebView2 [850](#)
 TsgcWSAPIKeyManager [207](#)
 TsgcWSAPIServer_WebAuthn [795](#)
 TsgcWSAPIServer_WebPush [594](#)
 TsgcWSCircuitBreaker [196](#)

TsgcWSConnection [224](#)
 TsgcWSMessageFile [327](#)
 TsgcWSPClient_AMQP [247](#)
 TsgcWSPClient_Dataset [322](#)
 TsgcWSPClient_E2EE [341](#)
 TsgcWSPClient_Files [326](#)
 TsgcWSPClient_MQTT [233](#)
 TsgcWSPClient_Presence [334](#)
 TsgcWSPClient_sgc [315](#)
 TsgcWSPClient_STOMP [288](#)
 TsgcWSPClient_STOMP_ActiveMQ [290](#)
 TsgcWSPClient_STOMP_RabbitMQ [289](#)
 TsgcWSPClient_WAMP [299](#)
 TsgcWSPClient_WAMP2 [309](#)
 TsgcWSPPresenceMessage [333](#)
 TsgcWSPServer_AppRTC [292](#)
 TsgcWSPServer_Dataset [321](#)
 TsgcWSPServer_E2EE [340](#)
 TsgcWSPServer_Files [325](#)
 TsgcWSPServer_Presence [332](#)
 TsgcWSPServer_sgc [314](#)
 TsgcWSPServer_WAMP [298](#)
 TsgcWSPServer_WebRTC [294](#)
 TsgcWSRateLimiter [184](#)
 TURN [878](#), [881](#), [882](#), [883](#), [884](#), [887](#), [888](#)
 TURN Client Alloca Indirizzo IP [881](#)
 TURN Client Create Permissions [882](#)
 TURN Client Send Indication [883](#)
 Utilizzo della DLL [49](#)
 Validazione DPoP (RFC 9449) [779](#)
 Verifica del certificato del server [123](#)
 Visione con Mistral [686](#)
 Wait Response [244](#)
 WAMP [303](#), [304](#), [305](#), [306](#)
 WAMP RPC Progress Results [306](#)
 WAMP Simple RPC [305](#)
 WatchDog [56](#)
 WebAuthn [794](#)
 WebAuthn Authentication Request [809](#)
 WebAuthn Authorization HTTP [814](#)
 WebAuthn Authorization WebSocket [815](#)
 WebAuthn MDS [812](#)
 WebPush [593](#)
 WebRTC [911](#)
 WebSocket [48](#), [375](#)
 WebSocket Events [47](#)
 WebSockets [25](#), [47](#), [86](#), [94](#), [107](#), [352](#), [353](#),
[376](#), [393](#), [398](#), [408](#), [415](#), [503](#), [504](#)
 WebSockets Private [398](#), [415](#)
 WebSockets Public [393](#), [408](#)
 WebView2 Downloads [859](#)
 WhatsApp - Invio di messaggi interattivi [556](#)
 WhatsApp Download Media [567](#)
 WhatsApp Invia Messaggi [553](#)
 WhatsApp Invio Messaggi Template [561](#)
 WhatsApp Phone Number Id [549](#)
 WhatsApp Ricezione Messaggi e Notifiche di Stato [563](#)
 WhatsApp Webhook [551](#)
 XSS [178](#)

API Huobi

Huobi (HTX)

Huobi (ora rinominata HTX) è un exchange di criptovalute internazionale multilingua.

Configurazione

Se si desidera sottoscrivere agli aggiornamenti privati dell'account, è necessario creare una API Key nel proprio account Huobi. Una volta creata, impostare la API Key e il Secret nel client API Huobi

- ApiKey: la api key creata nel proprio account huobi.
- ApiSecret: il segreto dell'API.

Se ApiKey non è vuota, il client tenterà di connettersi al server websocket privato e saranno disponibili solo i metodi privati. Se ApiKey è vuota, il client si connetterà al server websocket pubblico e saranno disponibili solo i metodi pubblici. Se è necessario sottoscrivere a metodi pubblici e privati, sono necessarie 2 connessioni.

Metodi pubblici

È possibile iscriversi ai seguenti canali pubblici (la chiave API non è necessaria)

Method	Description
SubscribeKLine	Questo topic invia una nuova candela ogni volta che è disponibile. Periodi supportati: 1min, 5min, 15min, 30min, 60min, 4hour, 1day, 1mon, 1week, 1year.
SubscribeMarketDepth	Questo argomento invia il più recente libro degli ordini di mercato per prezzo in modalità snapshot a intervalli di 1 secondo. Livelli di aggregazione della profondità supportati: da step0 a step15.
SubscribeTradeDetail	Questo argomento invia gli ultimi scambi completati. Si aggiorna tick per tick.
SubscribeMarketDetail	Questo topic invia le ultime statistiche di mercato con riepilogo 24h. Si aggiorna in modalità snapshot, con una frequenza non superiore a 10 volte al secondo.
SubscribeBBO	L'utente può ricevere l'aggiornamento BBO (Best Bid/Offer) in modalità tick by tick.
SubscribeMarketTicker	Recuperare il ticker di mercato. I dati vengono inviati ogni 100 ms.
SubscribeMarketByPrice	L'utente può iscriversi a questo canale per ricevere aggiornamenti del book degli ordini Market By Price. L'intervallo di aggiornamento è di circa 100 ms. Livelli supportati: 5, 10, 20, 150, 400.

Metodi pubblici Futures

I seguenti metodi sono disponibili per il client API Futures (TsgcWS_API_Huobi_Fut) in aggiunta a tutti i metodi pubblici sopra indicati.

Method	Description
SubscribeIncrementalMarketDepth	Sottoscrivi aggiornamenti incrementali della profondità di mercato con dimensione e tipo di dati configurabili (snapshot o incrementale).
SubscribePremiumIndexKLine	Sottoscrivere i dati kline/candlestick premium dell'indice per i contratti futures.
SubscribeEstimateRateKLine	Iscriviti ai dati kline/candlestick del tasso di finanziamento stimato per i contratti futures.
SubscribeBasisData	Sottoscrivere ai dati di base (spread di prezzo spot-futures). Supporta diversi tipi di prezzi di base: open, close, high, low.
SubscribeMarkPriceKLine	Iscriviti ai dati kline/candlestick del prezzo mark per i contratti futures.

SubscribeLiquidationOrders	Abbonati al feed degli ordini di liquidazione pubblica per un determinato contratto. Non è richiesta l'autenticazione.
SubscribeFundingRate	Sottoscrivere gli aggiornamenti pubblici del tasso di finanziamento per un determinato contratto. Non è richiesta autenticazione.
SubscribeContractInfo	Sottoscrive le modifiche dei parametri del contratto (ad esempio quotazioni, ritiri dalla quotazione, aggiustamenti dei parametri dei contratti).

Metodi privati

È possibile sottoscrivere i seguenti canali privati (è richiesta una chiave API). Se le credenziali non sono corrette, la connessione verrà chiusa automaticamente.

Method	Description
SubscribeOrderUpdates	Un aggiornamento dell'ordine può essere attivato da uno dei seguenti eventi: - Fallimento attivazione ordine condizionale (eventType=trigger) - Cancellazione ordine condizionale prima dell'attivazione (eventType=deletion) - Creazione ordine (eventType=creation) - Corrispondenza ordine (eventType=trade) - Cancellazione ordine (eventType=cancellation)
SubscribeTradeClearing	Aggiornamento solo quando l'ordine è in transazione o in cancellazione. L'aggiornamento della transazione dell'ordine avviene in modalità tick by tick, il che significa che se l'ordine di un taker corrisponde a più ordini di maker, le più transazioni simultanee verranno diffuse una alla volta. Tuttavia, la sequenza di aggiornamento delle transazioni multiple potrebbe non corrispondere esattamente alla sequenza delle transazioni effettuate. Inoltre, se un ordine viene annullato automaticamente immediatamente dopo le sue esecuzioni parziali, come un tipico ordine IOC, questo canale potrebbe diffondere prima l'aggiornamento di cancellazione rispetto alla transazione.
SubscribeAccountChange	Specificato il campo di sottoscrizione, l'aggiornamento può essere attivato da uno dei seguenti eventi. Il parametro aMode controlla il comportamento dell'aggiornamento: - Modalità 0: aggiorna solo quando il saldo del conto viene modificato. - Modalità 1: aggiornamento quando il saldo del conto o il saldo disponibile cambia (aggiornamenti separati). - Modalità 2: aggiornamento quando il saldo del conto o il saldo disponibile cambia (aggiornamento combinato).

Eventi

OnHuobiSubscribed: evento chiamato dopo una sottoscrizione avvenuta con successo.

OnHuobiUnSubscribed: evento chiamato dopo una cancellazione della sottoscrizione avvenuta con successo.

OnHuobiUpdate: ogni volta che c'è un aggiornamento nei dati (kline, market depth...) viene chiamato questo evento.

OnHuobiError: se si verifica un errore nell'API Huobi, questo evento fornirà informazioni sull'errore.

TsgcWSPClient_AMQP1

Il client **TsgcWSPClient_AMQP1** implementa il protocollo **AMQP 1.0.0** seguendo la specifica OASIS. Il client supporta connessioni TCP semplice e WebSocket; sono supportate anche le connessioni TLS (sicure).

Configurazione

Il client AMQP 1.0.0 dispone della proprietà **AMQPOptions** dove è possibile configurare la connessione.

- **ChannelMax:** Il valore channel-max è il numero di canale più alto che può essere utilizzato sulla connessione. Questo valore più uno è il numero massimo di sessioni che possono essere contemporaneamente attive sulla connessione
- **ContainerId:** (facoltativo) è il nome del container di origine, identifica univocamente la connessione nel server.
- **CreditSize:** dimensione predefinita del flusso di credito.
- **IdleTimeout:** Il timeout viene attivato da un peer locale quando non vengono ricevuti frame dopo il superamento di un valore di soglia. Il timeout di inattività è misurato in millisecondi e parte dal momento in cui viene ricevuto l'ultimo frame.
- **MaxFrameSize:** la dimensione massima accettata del frame.
- **MaxLinksPerSession:** il numero massimo di link per sessione.
- **WindowSize:** la dimensione predefinita della finestra.

L'autenticazione AMQP deve essere configurata nella proprietà Authentication.

- **AuthType:** tipo di autenticazione
 - **amqp1authNone:** non configurato.
 - **amqp1authSASLAnonymous:** autenticazione anonima
 - **amqp1authSASLPlain:** autenticazione utente/password. Questo tipo di autenticazione richiede la compilazione delle seguenti proprietà:
 - Username
 - Password
 - **amqp1authSASLExternal:** autenticazione esterna

Connessione

La connessione inizia con il client (solitamente un'applicazione o un servizio di messaggistica) che avvia una connessione TCP verso il server (il message broker). Il client si connette alla porta del server, tipicamente 5672 per connessioni non TLS e 5671 per connessioni protette da TLS. Una volta stabilita la connessione TCP, client e server negoziano la versione del protocollo AMQP da utilizzare. AMQP 1.0.0 supporta varie versioni e, durante la negoziazione, entrambe le parti concordano di utilizzare la versione 1.0.0.

Dopo la negoziazione del protocollo, il client potrebbe dover autenticarsi presso il server, a seconda della configurazione del server. I meccanismi di autenticazione possono includere meccanismi SASL (Simple Authentication and Security Layer) come PLAIN, EXTERNAL o altri supportati dal server.

Esempio: connessione al server AMQP in ascolto sulla porta sicura 5671 con credenziali SASL

```
oAMQP = new TsgcWSPClient_AMQP1(this);
// Setting AMQP authentication options
oAMQP.AMQPOptions.Authentication.AuthType = amqp1authSASLPlain;
oAMQP.AMQPOptions.Authentication.Username = "sgc";
oAMQP.AMQPOptions.Authentication.Password = "sgc";
// Creating WebSocket client
oClient = new TsgcWebSocketClient(this);
// Setting WebSocket specifications
oClient.Specifications.RFC6455 = false;
// Setting WebSocket client properties
oClient.Host = "www.esegece.com";
oClient.Port = 5671;
oClient.TLS = true;
// Assigning WebSocket client to AMQP client
oAMQP.Client = oClient;
// Activating WebSocket client
oClient.Active = true;
```

Sessioni

Una volta autenticato, il client apre una sessione AMQP. Una sessione è un contesto logico per la comunicazione tra client e server. Le sessioni vengono utilizzate per raggruppare operazioni di messaggistica correlate. Utilizzare il

metodo **CreateSession** per creare una nuova sessione; il metodo consente di impostare il nome della sessione oppure di lasciarlo vuoto affinché il componente ne assegni uno automaticamente.

Se la sessione è stata creata correttamente, verrà generato l'evento **OnAMQPSessionOpen** con i dettagli della sessione.

```
oAMQP.OnAMQPSessionOpen += AMQP1AMQPSessionOpen;
oAMQP.CreateSession("MySession");
private void AMQP1AMQPSessionOpen(object sender, TsgcAMQP1Session aSession, TsgcAMQP1FrameBegin aBegin)
{
    Console.WriteLine("#session-open: " + aSession.Id);
}
```

Link

All'interno di una sessione, il client crea link per comunicare con entità specifiche come code, topic o altre risorse fornite dal server. I link sono canali di comunicazione bidirezionali utilizzati per l'invio e la ricezione di messaggi.

Il componente può funzionare come nodo mittente e ricevente. Consente di creare qualsiasi numero di link per ogni sessione, fino al limite impostato nella proprietà **MaxLinksPerSession**.

Sender Link

Per creare un nuovo link mittente, utilizzare il metodo **CreateSenderLink** e passare il nome della sessione e facoltativamente il nome del link mittente. Se il link viene creato correttamente, viene generato l'evento **OnAMQPLinkOpen**.

```
oAMQP.OnAMQPLinkOpen += AMQP1AMQPLinkOpen;
oAMQP.CreateSenderLink("MySession", "MySenderLink");
private void AMQP1AMQPLinkOpen(object sender, TsgcAMQP1Session aSession, TsgcAMQP1Link aLink, TsgcAMQP1FrameAttac
{
    Console.WriteLine("#link-open: " + aLink.Name);
}
```

Link ricevitore

Per creare un nuovo link al ricevitore, utilizzare il metodo **CreateReceiverLink** e passare il nome della sessione e facoltativamente il nome del link al ricevitore. Se il link viene creato correttamente, viene attivato l'evento **OnAMQPLinkOpen**.

```
oAMQP.AMQPLinkOpen += AMQP1AMQPLinkOpen;
oAMQP.OnCreateReceiverLink("MySession", "MySReceiverLink");
private void AMQP1AMQPLinkOpen(object sender, TsgcAMQP1Session aSession, TsgcAMQP1Link aLink, TsgcAMQP1FrameAttac
{
    Console.WriteLine("#link-open: " + aLink.Name);
}
```

Invio messaggi

Con la sessione stabilita e i link creati, il client può iniziare a eseguire operazioni sui messaggi, come l'invio di messaggi a una destinazione. Utilizzare il metodo **SendMessage** per inviare un messaggio tramite un link mittente.

```
oAMQP.SendMessage("MySession", "MySenderLink", "My first AMQP Message");
```

Ricezione di messaggi

Per impostazione predefinita, i Receiver Link vengono creati in **modalità automatica**, il che significa che ogni volta che arriva un nuovo messaggio, verrà consegnato al client.

Se i Receiver Link sono stati creati in **modalità manuale**, utilizzi il metodo Sync **GetMessage** per recuperare e attendere finché non arriva un nuovo messaggio.

In modalità Automatica e Manuale, ogni volta che arriva un nuovo messaggio viene attivato l'evento **OnAMQPMes-**
sage.

```
public void OnAMQPMessageEvent(object Sender, TsgcAMQP1Session aSession,
    TsgcAMQP1ReceiverLink aLink, TsgcAMQP1Message aMessage,
    ref TsgcAMQP1MessageDeliveryState DeliveryState)
{
    System.Windows.Forms.MessageBox.Show(aMessage.ApplicationData.AMQPValue.Value);
}
```

API Bitstamp

Bitstamp

Bitstamp è un exchange bitcoin con sede in Lussemburgo. Consente il trading tra la valuta USD e la criptovaluta bitcoin. Permette depositi e prelievi in USD, EUR, bitcoin, litecoin, ethereum, ripple o bitcoin cash.

- Supporta la più recente **WebSocket API V2**.
- Supporta i seguenti endpoint **REST API: Account Balance, User Transactions, Fees, Orders, Withdrawal, Deposit Addresses, Sub-Account Transfers, Earn/Staking, Travel Rule e Markets**.

Configurazione

L'API privata richiede la creazione di una API key dal Suo account Bitstamp. Una volta ottenute le Sue API key, configuri queste chiavi nelle seguenti proprietà:

- Bitstamp.ApiKey
- Bitstamp.ApiSecret

WebSocket Public Methods

SubscribeLiveTicker: ottieni le negoziazioni in tempo reale dalla valuta selezionata. Dati JSON:

Proprietà	Descrizione
id	ID univoco del trade.
amount	Importo della transazione.
amount_str	Importo dell'operazione rappresentato in formato stringa.
price	Prezzo di scambio.
price_str	Prezzo di negoziazione rappresentato in formato stringa.
type	Tipo di operazione (0 - acquisto; 1 - vendita).
timestamp	Timestamp dell'operazione.
microtimestamp	Microtimestamp della negoziazione.
buy_order_id	ID ordine di acquisto del trade.
sell_order_id	ID dell'ordine di vendita.

SubscribeLiveOrders: ottieni gli ordini in tempo reale dalla valuta selezionata. Dati JSON:

Proprietà	Descrizione
id	ID ordine.
amount	Importo dell'ordine.
amount_str	Importo dell'ordine rappresentato in formato stringa.
price	Prezzo dell'ordine.
price_str	Prezzo dell'ordine rappresentato in formato stringa.
order_type	Tipo di ordine (0 - acquisto; 1 - vendita).
datetime	Data e ora dell'ordine.
microtimestamp	Timestamp dell'azione dell'ordine espresso in microsecondi.

SubscribeLiveOrderBook: ottiene l'order book in tempo reale dalla valuta selezionata. Dati JSON:

Proprietà	Descrizione
bids	Elenco delle prime 100 offerte di acquisto.

Proprietà	Descrizione
asks	Elenco dei primi 100 ordini ask.
timestamp	Timestamp del book degli ordini.
microtimestamp	Microtimestamp del libro degli ordini.

SubscribeLiveDetailOrderBook: ottieni il libro ordini dettagliato in tempo reale dalla valuta selezionata. Dati JSON:

Proprietà	Descrizione
bids	Elenco dei primi 100 ordini di acquisto [prezzo, quantità, ID ordine].
asks	Elenco delle prime 100 offerte di vendita [prezzo, quantità, id ordine].
timestamp	Timestamp del book degli ordini.
microtimestamp	Microtimestamp del libro degli ordini.

SubscribeLiveFullOrderBook: ottieni il book degli ordini completo in tempo reale dalla valuta selezionata. Dati JSON:

Proprietà	Descrizione
bids	Elenco delle offerte modificate dall'ultima trasmissione.
asks	Elenco delle richieste di vendita modificate dall'ultima trasmissione.
timestamp	Timestamp del book degli ordini.
microtimestamp	Microtimestamp del libro degli ordini.

Metodi privati WebSocket

SubscribeMyOrders: ricevi aggiornamenti sugli ordini inviati. Dati JSON:

Proprietà	Descrizione
id	ID ordine.

Proprietà	Descrizione
id_str	ID ordine rappresentato in formato stringa.
client_order_id	ID ordine client (se utilizzato al momento del piazzamento dell'ordine).
amount	Importo dell'ordine.
amount_str	Importo dell'ordine rappresentato in formato stringa.
price	Prezzo dell'ordine.
price_str	Prezzo dell'ordine rappresentato in formato stringa.
order_type	Tipo di ordine (0 - acquisto, 1 - vendita).
datetime	Data e ora dell'ordine.
microtimestamp	Timestamp dell'azione dell'ordine espresso in microsecondi.

SubscribeMyTrades: ottiene aggiornamenti sugli scambi. Dati JSON:

Proprietà	Descrizione
id	ID del trade.
order_id	ID ordine associato alla negoziazione.
client_order_id	ID dell'ordine client associato al trade.
amount	Importo della transazione.
price	Prezzo di scambio.
fee	Commissione di negoziazione.
side	Lato operazione (acquisto o vendita).
microtimestamp	Timestamp della transazione rappresentato in microsecondi.

Metodi pubblici REST API

Tickers

- **GetCurrencies:** Vista che restituisce l'elenco di tutte le valute con i dati di base.
- **GetAllCurrencyPairsTickers:** Restituisce i dati ticker per tutte le coppie di valute. Il passaggio di eventuali parametri GET comporterà il rifiuto della richiesta.

- **GetCurrencyPairTicker:** Restituisce i dati del ticker per la coppia di valute richiesta. Il passaggio di parametri GET comporterà il rifiuto della richiesta.
- **GetHourlyTicker:** Restituisce i dati del ticker orario per la coppia di valute richiesta. Il passaggio di qualsiasi parametro GET comporterà il rifiuto della richiesta.

Libro degli ordini

- **GetOrderBook:** Restituisce i dati del book degli ordini.

Transazioni

- **GetTransactions:** Restituisce i dati delle transazioni per un determinato intervallo di tempo.

Informazioni di mercato

- **GetEURUSDConversionRate:** Restituisce il tasso di conversione EUR/USD.
- **GetOLHCDData:** Vista che restituisce i dati OHLC (Open High Low Close) su richiesta API.
- **GetTradingPairsInfo:** Restituisce informazioni sulle coppie di trading.
- **GetMarkets:** Restituisce l'elenco di tutti i mercati disponibili con i dati di base.

Travel Rule

- **GetTravelRuleVASPs:** Restituisce l'elenco dei Virtual Asset Service Provider (VASP) per la conformità al Travel Rule.

Metodi Privati REST API

Saldo del conto

- **GetAccountBalances:** Restituisce i saldi del conto.
- **GetAccountBalanceForCurrency:** Restituisce i saldi dell'account per valuta.

Ordini

- **BuyInstantOrder:** Apre un ordine istantaneo di acquisto. Effettuando un ordine istantaneo si riconosce che l'esecuzione dell'ordine dipende dalle condizioni di mercato e che tali condizioni possono essere soggette a cambiamenti improvvisi non prevedibili. Questa chiamata verrà eseguita sull'account (Sub o Main) a cui è associata la chiave API utilizzata.
- **BuyMarketOrder:** Apre un ordine di acquisto al mercato. Inserendo un ordine al mercato l'utente riconosce che l'esecuzione del proprio ordine dipende dalle condizioni di mercato e che tali condizioni possono essere soggette a cambiamenti improvvisi e imprevedibili. Questa chiamata verrà eseguita sull'account (Sub o Main) a cui è associata la chiave API utilizzata.
- **BuyLimitOrder:** Apre un ordine limite di acquisto. Questa chiamata verrà eseguita sull'account (Sub o Main) a cui è associata la chiave API utilizzata.
- **CancelAllOrders:** Annulla tutti gli ordini aperti. Questa chiamata verrà eseguita sull'account (Sub o Main) al quale è associata la chiave API utilizzata.
- **CancelAllOrdersForCurrencyPair:** Annulla tutti gli ordini aperti per una coppia di valute. Questa chiamata verrà eseguita sull'account (Sub o Main) a cui è associata la chiave API utilizzata.
- **CancelOrder:** Annulla un ordine. Questa chiamata verrà eseguita sull'account (Sub o Main) a cui è associata la chiave API utilizzata.
- **GetTradingPairs:** Restituisce tutte le coppie di trading che possono essere negoziate sull'account selezionato.
- **GetAllOpenOrders:** Restituisce gli ordini aperti dell'utente. Questa chiamata API viene memorizzata nella cache per 10 secondi. Questa chiamata verrà eseguita sull'account (Sub o Main) a cui è associata la chiave API utilizzata.

- **GetOpenOrders:** Restituisce gli ordini aperti dell'utente per la coppia di valute. Questa chiamata API viene memorizzata nella cache per 10 secondi. La chiamata verrà eseguita sull'account (Sub o Main) a cui è associata la chiave API utilizzata.
- **GetOrderStatus:** Restituisce lo stato dell'ordine. Questa chiamata verrà eseguita sull'account (Sub o Main) a cui è associata la chiave API utilizzata. L'ordine può essere recuperato utilizzando il parametro id o client_order_id. Per gli ordini chiusi, questa chiamata restituisce informazioni solo per gli ultimi 30 giorni. Verrà restituito l'errore 'Order not found' per gli ordini al di fuori di questo intervallo di tempo.
- **SellInstantOrder:** Apre un ordine di vendita istantaneo. Piazzando un ordine istantaneo si riconosce che l'esecuzione dell'ordine dipende dalle condizioni di mercato e che tali condizioni possono essere soggette a cambiamenti improvvisi che non possono essere previsti. Questa chiamata verrà eseguita sull'account (Sub o Main) a cui è associata la chiave API utilizzata.
- **SellMarketOrder:** Apre un ordine di vendita a mercato. Inserendo un ordine a mercato si riconosce che l'esecuzione dell'ordine dipende dalle condizioni di mercato e che tali condizioni possono essere soggette a cambiamenti improvvisi non prevedibili. Questa chiamata verrà eseguita sull'account (Sub o Main) a cui è associata la chiave API utilizzata.
- **SellLimitOrder:** Apre un ordine limit di vendita. Questa chiamata verrà eseguita sull'account (Sub o Main) a cui è associata la chiave API utilizzata.

Prelievi

- **RippleIOUWithdrawal:** Questa chiamata verrà eseguita sull'account (Sub o Main) a cui è associata la chiave API utilizzata. Questo endpoint supporta i prelievi di IOU in USD, BTC o EUR sull'XRP Ledger.
- **WithdrawalRequests:** Restituisce le richieste di prelievo dell'utente. Questa chiamata verrà eseguita sull'account (Sub o Main) a cui è associata la chiave API utilizzata.
- **CancelBankOrCryptoWithdrawal:** Annulla una richiesta di prelievo bancario o in criptovaluta. Questa chiamata può essere effettuata solo dal suo Main Account.
- **OpenBankWithdrawal:** Apre una richiesta di prelievo bancario (SEPA o internazionale). Le richieste di prelievo aperte tramite API vengono confermate automaticamente (non verrà inviata alcuna e-mail di conferma), ma vengono elaborate esattamente come i prelievi aperti tramite l'interfaccia della piattaforma. Questa chiamata può essere effettuata solo dal conto principale.
- **FiatWithdrawalStatus:** Verifica lo stato di una richiesta di prelievo in valuta fiat. Questa chiamata può essere eseguita solo dal conto principale.
- **CryptoWithdrawal:** Richiede un prelievo di criptovaluta.

Transazioni utente

- **GetUserTransactions:** Restituisce tutte le transazioni dell'utente. Supporta i parametri limit, offset e sort.
- **GetUserTransactionsForCurrencyPair:** Restituisce le transazioni dell'utente per una specifica coppia di valute. Supporta i parametri limit, offset e sort.

Commissioni

- **GetTradingFees:** Restituisce tutte le commissioni di trading.
- **GetTradingFeesForCurrencyPair:** Restituisce le commissioni di trading per una specifica coppia di valute.
- **GetWithdrawalFees:** Restituisce le commissioni di prelievo per tutte le valute.

Indirizzi di Deposito

- **GetCryptoDepositAddress:** Restituisce l'indirizzo di deposito per la criptovaluta specificata.

Trasferimenti tra sub-account

- **TransferToMain:** Trasferisce fondi da un sotto-account all'account principale.
- **TransferFromMain:** Trasferisce fondi dall'account principale a un sotto-account.

Guadagni / Staking

- **EarnSubscribe:** Effettua la sottoscrizione al programma Earn per la valuta e l'importo specificati.
- **EarnUnsubscribe:** Annulla l'iscrizione al programma Earn per la valuta e l'importo specificati.
- **GetEarnSubscriptions:** Restituisce le sottoscrizioni Earn correnti.
- **GetEarnTransactions:** Restituisce la cronologia delle transazioni Earn (premi, sottoscrizioni, disdette).