



sgcWebSockets 2026.6

June 2026

Documentação para CBuilder

Copyright © 2012-2026 eSeGeCe Software

info@esegece.com

www.esegece.com

Conteúdo

Introdução	23
Visão geral	28
Edições	28
Instalação	30
Instalação	33
Install Setup	33
Instalar pacote	44
Install Errors.....	51
Configure Install	55
Instalar o Pacote sgclndy	57
Configure ZLib	63
Início rápido.....	64
Visão geral.....	64
QuickStart WebSockets	66
QuickStart HTTP	68
Fluxo de threads.....	70
Build	72
Build OSX Application	73
Build Android Application	75
Compilar Aplicação iOS	76
Servidor de Alto Desempenho	78
Gerenciador de Memória	82
OpenSSL	85
OpenSSL Windows	88
OpenSSL OSX	90

OpenSSL Android	92
OpenSSL iOS	93
OpenSSL Own CA Certificates.....	95
OpenSSL P12 Certificates	97
OpenSSL Verify Certificate	98
OpenSSL Load Additional Functions.....	99
Indy	101

Tópicos 103

WebSocket Events	103
WebSocket Parameters Connection	104
Uso dentro de uma DLL	105
Windows Service	106
Teste no navegador web	107
Subprotocolos Personalizados	108
Autenticação	109
Conexões Seguras.....	111
HeartBeat	113
WatchDog.....	114
Logs.....	115
HTTP.....	116
Broadcast e Canais.....	117
Bindings.....	118
Enviar Arquivos Grandes por POST	119
Compressão.....	121
Flash.....	122
Custom Objects	123
Groups.....	124
IOCP	126
EPOLL.....	127
ALPN	128

Encaminhar Requisições HTTP	129
Quality Of Service	130
Queues	132
Transactions	134
Conexões TCP	136
SubProtocol	137
Throttle	138
Server-Sent Events	139
LoadBalancing	141
Files	142
Proxy	143
Mensagens Fragmentadas	144

Componentes 145

TsgcWebSocketClient	145
Connect WebSocket Server	147
Client Open Connection	148
Cliente Fecha Conexão	150
Client Keep Connection Open	151
Dropped Disconnections	152
Connect TCP Server	153
Connections TIME_WAIT	154
WebSocket Redirections	155
Connect Secure Server	156
Certificados OpenSSL	157
Certificates SChannel	158
SChannel Get Connection Info	160
Client Send Text Message	161
Client Send Binary Message	162
Cliente Envia Mensagem de Texto e Binária	163
Receber mensagens de texto	164

Receber Mensagens Binárias	165
Autenticação do cliente	166
Exceções do cliente.....	169
Client WebSocket HandShake	170
Client Register Protocol	171
Client Proxies	172
TsgcTCPClient	173
TsgcTCPClient	174
TsgcWebSocketServer	175
Server Start	177
Server Bindings.....	178
Server Startup Shutdown	179
Servidor Manter Ativo.....	180
Server SSL.....	181
Server SSL SChannel	183
Server Verify Certificate.....	186
Servidor Mantém Conexões Ativas	187
Server Plain TCP	188
Conexão Fechada pelo Servidor.....	189
Client Connections	190
Server Authentication.....	191
Server Send Text Message	193
Server Send Binary Message	194
Server Receive Text Message.....	195
Servidor recebe mensagem binária.....	196
Server Read Headers from Client.....	197
TsgcWebSocketHTTPServer	198
Requisições do Servidor HTTP	200
HTTP Dispatch Files.....	201
HTTP/2 Server	202
HTTP/2 Server Push	203

HTTP/2 Alternate Service.....	205
Threads do Servidor HTTP/2.....	206
Erro HTTP 404 sem corpo de resposta.....	208
HTTP Server Sessions	209
HTTP Server Stream Video	211
Server SSL SChannel	212
TsgcWebSocketServer_HTTPAPI	215
HTTPAPI URL Reservation	217
HTTPAPI FineTune	219
HTTPAPI Server SSL.....	221
Certificados Autoassinados	222
HTTPAPI Disable HTTP/2	223
HTTPAPI Cabeçalhos personalizados	224
HTTPAPI Send Text Response.....	225
HTTPAPI Send File Response	226
OnDisconnect do HTTPAPI não disparado.....	228
TsgcWebSocketClient_WinHTTP	229
TsgcWebSocketFirewall	230
Firewall: Blacklist e Whitelist.....	237
Firewall: Proteção contra Força Bruta	239
Firewall: Detecção de SQL Injection e XSS.....	241
Firewall: Rate Limiting e Proteção contra Flood	244
Firewall: Detecção de Bots	247
TsgcWSRateLimiter	250
TsgcWSCircuitBreaker.....	262
TsgcWSAPIKeyManager	273
TsgcWebSocketLoadBalancerServer.....	286
TsgcWebSocketProxyServer.....	288
TsgcIWWebSocketClient	289
TsgcWSConnection.....	291
Protocolos	293

Protocols Javascript.....	295
Protocol MQTT.....	298
TsgcWSPClient_MQTT	300
Cliente MQTT Connect.....	301
Conectar a Servidores MQTT Mosquitto	302
Sessões MQTT do Cliente	303
Client MQTT Version	304
MQTT Publish Subscribe	305
MQTT Topics	306
MQTT Subscribe	308
MQTT Publish Message	309
Receber Mensagens MQTT	310
MQTT Publish e aguardar resposta	311
MQTT Clear Retained Messages.....	312
Protocol AMQP	313
TsgcWSPClient_AMQP.....	314
Client AMQP Connect	317
Client AMQP Disconnect.....	318
Canais AMQP	319
AMQP Exchanges	321
AMQP Queues	323
AMQP Publish Messages.....	326
AMQP Consume Messages	327
AMQP Get Messages.....	329
AMQP QoS.....	330
AMQP Transactions.....	331
Protocolo Kafka	333
TsgcWSPClient_Kafka.....	334
Client Kafka Connect.....	336
Client Kafka Disconnect.....	337
Kafka Produce Messages	338

Kafka Consume Messages	339
Kafka Consumer Groups	340
Tópicos Kafka.....	341
Kafka Offsets.....	342
Protocolo AMQP1	343
TsgcWSPClient_AMQP1	345
Client AMQP1 Connect	348
Client AMQP1 Disconnect	349
Client AMQP1 Idle Timeout Connection.....	350
Estado da Conexão AMQP1 do Cliente.....	351
Client AMQP1 Authentication	352
Client AMQP1 Azure MessageBus.....	353
Sessões AMQP1	355
Links AMQP1.....	357
AMQP1 Sender Links	358
AMQP1 Receiver Links	361
AMQP1 Send Message.....	363
AMQP1 Read Message.....	365
Protocol STOMP	366
TsgcWSPClient_STOMP.....	367
TsgcWSPClient_STOMP_RabbitMQ.....	368
TsgcWSPClient_STOMP_ActiveMQ	369
Protocolo AppRTC	370
TsgcWSPServer_AppRTC.....	371
Protocolo WebRTC	372
TsgcWSPServer_WebRTC.....	373
Protocolo WebRTC Javascript	374
Protocolo WAMP	375
TsgcWSPServer_WAMP.....	377
TsgcWSPClient_WAMP	378
Protocol WAMP Javascript.....	379

Assinantes	382
Publicadores	383
RPC simples	384
Resultados de progresso de RPC	385
Protocolo WAMP2	387
TsgcWSPClient_WAMP2	388
Protocol Default	391
TsgcWSPServer_sgc.....	393
TsgcWSPClient_sgc	394
TsgcIWWSPClient_sgc	395
Protocol Default Javascript.....	396
Protocolo Dataset	400
TsgcWSPServer_Dataset	401
TsgcWSPClient_Dataset	402
TsgcIWWSPClient_Dataset.....	403
Protocolo Dataset Javascript.....	404
Protocol Dataset Replicate Table	407
Protocol Dataset Notify Updates.....	408
Protocol Files	409
TsgcWSPServer_Files.....	410
TsgcWSPClient_Files	411
TsgcWSMessageFile	412
Como enviar arquivos ao servidor	413
Como Enviar Arquivos aos Clientes	414
How Send Big Files	415
Protocol Presence	416
TsgcWSPServer_Presence	417
TsgcWSPPresenceMessage	418
TsgcWSPClient_Presence.....	419
Protocolo Presence Javascript	420
Protocol E2EE.....	423

TsgcWSPServer_E2EE	425
TsgcWSPClient_E2EE	426
APIs	427
API Binance	429
Binance Connect WebSocket API	437
Binance Subscribe WebSocket Channel	438
Binance Get Market Data	439
REST API Privada da Binance	440
Binance Trade Spot.....	441
Binance Private Requests Time	443
Binance Withdraw	444
API Binance Futures	445
API Binance Futures Trade	452
API SocketIO.....	453
API Coinbase	455
Coinbase Connect WebSocket API	460
Coinbase Subscribe WebSocket Channel.....	461
Coinbase Get Market Data	462
Coinbase Private REST API	463
Coinbase Private Requests Time	464
Coinbase Place Orders	465
Coinbase SandBox Account	466
API SignalRCore	467
API SignalR	473
API Kraken.....	476
API Kraken WebSockets Public	478
API Kraken WebSockets Private.....	484
API Kraken REST Public.....	487
API Kraken REST Private	489
API Kraken Futures.....	492
API Kraken Futures WebSockets Public.....	494

API Kraken Futures WebSockets Privada	501
API Kraken Futures REST Public	507
API Kraken Futures REST Private	509
API Pusher	514
API Bitmex	523
Bitmex Connect WebSocket API	527
Bitmex Subscribe WebSocket Channel	528
Como registrar uma ordem na Bitmex	529
API Bitfinex	532
API Kucoin	536
Kucoin Connect WebSocket API	543
Kucoin Subscribe WebSocket Channel	544
Kucoin Get Market Data	545
Kucoin Private REST API	546
Kucoin Trade Spot	547
Horário das Requisições Privadas Kucoin	549
API Kucoin Futures	550
Kucoin Futures Conectar à WebSocket API	556
Kucoin Futures Subscribe WebSocket Channel	557
Kucoin Futures Get Market Data	558
Kucoin Futures Private REST API	559
Kucoin Futures Trade	560
Kucoin Futures Private Requests Time	563
API 3Commas	564
API OKX	569
API XTB	575
API Bybit	579
API Forex	584
Forex Connect REST API	586
Forex Trade Orders	587
Forex Account Queries	589

Dados de Mercado Forex	590
Forex Connect WebSocket API	591
Forex Assinar Streams	592
API Blockchain	594
API Cex.....	596
API Cex Plus	604
API Discord.....	609
API OpenAI	612
API MEXC	614
API MEXC Futures.....	618
API Bitget.....	622
API Gatelo	624
API Deribit	626
API Crypto.com.....	629
API HTX	631
API Whatsapp	633
WhatsApp Create App	637
WhatsApp Phone Number Id.....	639
WhatsApp Token	640
WhatsApp Webhook	641
WhatsApp Security.....	642
WhatsApp Send Messages.....	643
WhatsApp Enviar mensagens interativas.....	646
WhatsApp Send Template Messages.....	651
WhatsApp Receive Messages and Status Notifications.....	653
WhatsApp Send Files	655
WhatsApp Download Media	657
API Telegram.....	658
Send Telegram Message With Inline Buttons.....	667
Send Telegram Message With Buttons.....	668
Send Telegram Message Bold	669

Telegram Chat not found as Bot	670
Mensagens Patrocinadas do Telegram	671
Send Telegram Invoice Message	672
Telegram Get SuperGroup Members	673
Adicionar Proxy do Telegram	674
Register Telegram User	675
RCON	676
CryptoHopper	677
RTCMultiConnection	682
WebPush	684
TsgcWSAPIServer_WebPush	685
TsgcWebPush_Client.....	686
Extensions	687
PerMessage-Deflate	688
Deflate-Frame	689
MCP	690
TsgcWSAPIServer_MCP	691
MCP Server Sessions.....	693
Ferramentas do servidor MCP	694
MCP Server Prompts.....	697
MCP Server Resources.....	700
MCP Server Roots.....	703
MCP Server Sampling	704
MCP Server Elicitation.....	705
TsgcWSAPIClient_MCP	707
MCP Client Tools	708
MCP Client Prompts.....	710
Recursos do Cliente MCP	712
MCP Client Roots.....	714
MCP Client Sampling.....	715
MCP Client Elicitation.....	716

OpenAI.....	717
TsgcHTTP_API_OpenAI.....	724
OpenAI Completion	725
OpenAI Chat.....	726
OpenAI Edit.....	727
OpenAI Audio.....	728
OpenAI Moderation	729
OpenAI RealTime.....	730
OpenAI Responses	731
OpenAI Speech	732
OpenAI Fine-Tuning	733
OpenAI Batch.....	734
OpenAI Uploads	735
OpenAI Applications	736
TsgcAIOpenAIAssistant.....	737
TsgcAIOpenAIAssistant File Search	739
TsgcAIOpenAIAssistant Streaming.....	741
TsgcAIOpenAIAssistant Function Calling.....	743
OpenAI Audio.....	745
TsgcAudioRecorderMCI	746
TsgcAudioPlayerMCI	747
TsgcTextToSpeechSystem.....	748
TsgcTextToSpeechGoogle	749
TsgcTextToSpeechAmazon	750
TsgcAIChat.....	751
TsgcAIOpenAIChatBot	752
TsgcAIOpenAITranslator.....	753
TsgcAIOpenAIEmbeddings.....	754
TsgcAIDatabaseVectorFile.....	755
TsgcAIDatabaseVectorPinecone.....	756
Embeddings Criar Vetores	757

Embeddings ChatBot	758
Pinecone	759
Anthropic	762
TsgcHTTP_API_Anthropic	766
Anthropic Messages	767
Anthropic Vision	769
Anthropic Tool Use	771
Anthropic Models	773
Anthropic Batches	774
Anthropic Extended Thinking	775
Documentos Anthropic	777
Anthropic Prompt Caching	779
Anthropic Citations	781
Anthropic Web Search	782
Anthropic Structured Outputs	784
Anthropic Files	786
Anthropic MCP Connector	788
Gemini	790
TsgcHTTP_API_Gemini	792
DeepSeek	793
TsgcHTTP_API_DeepSeek	795
DeepSeek Messages	796
DeepSeek Vision	797
DeepSeek Models	798
Gemini	799
Mensagens do Gemini	801
Gemini Vision	803
Gemini Models	804
Gemini Structured Outputs	805
Gemini Token Counting	806
Embeddings Gemini	807

Gemini Tool Use	808
Ollama	810
TsgcHTTP_API_Ollama	812
Ollama Messages	813
Ollama Models.....	814
Ollama Embeddings.....	815
Grok	816
TsgcHTTP_API_Grok	818
Grok Messages	819
Grok Vision.....	820
Grok Models.....	821
Mistral AI	822
TsgcHTTP_API_Mistral.....	824
Mistral Messages.....	825
Mistral Vision	826
Mistral Models	827
Embeddings do Mistral.....	828
IoT	829
IoT Amazon MQTT Client.....	830
IoT Azure MQTT Client	838
HTTP.....	843
HTTP2	844
TsgcHTTP2Client.....	845
Request HTTP/2 Method	847
HTTP/2 Server Push	848
HTTP/2 Download File	849
Respostas Parciais HTTP/2.....	850
HTTP/2 Headers	851
Cliente Fecha Conexão	852
Client Keep Connection Active.....	853
HTTP/2 Reason Disconnection	854

Requisições Pendentes do Cliente	855
Autenticação do cliente	856
HTTP/2 e OAuth2.....	857
TsgcHTTP2ConnectionClient.....	858
TsgcHTTP2RequestProperty	859
TsgcHTTP2ResponseProperty.....	860
Apple Push Notifications	861
Gerar uma Notificação Remota APNs.....	862
Enviando Requisições de Notificação ao APNs	863
Token-Based Connection to APNs	864
Conexão Baseada em Certificado com APNs	865
HTTP1	867
OAuth2	871
TsgcHTTP_OAuth2_Client	872
OAuth2 Client for Web Applications	876
OAuth2 Client for Desktop Applications.....	877
TsgcHTTP_OAuth2_Client_Google	878
TsgcHTTP_OAuth2_Client_Microsoft.....	879
Authorization Code Grant (RFC 6749).....	880
Authorization Code com PKCE (RFC 7636)	882
Client Credentials Grant (RFC 6749).....	884
Resource Owner Password Credentials Grant (RFC 6749).....	886
Device Authorization Grant (RFC 8628)	888
DPoP - Demonstrating Proof of Possession (RFC 9449).....	890
TsgcHTTP_OAuth2_Server	894
OAuth2 Server Example	897
OAuth2 Customize Sign-In HTML	901
OAuth2 Server Endpoints.....	902
OAuth2 Register Apps	903
OAuth2 Recover Access Tokens	904
OAuth2 Server Authentication.....	905

OAuth2 None Authenticate URLs	906
Authorization Code Grant	907
Client Credentials Grant	909
Resource Owner Password Credentials Grant	911
Refresh Token Grant.....	913
Device Authorization Grant (RFC 8628)	915
Token Revocation (RFC 7009)	917
Token Introspection (RFC 7662)	919
DPoP Validation (RFC 9449)	921
TsgcHTTP_OAuth2_Server_Provider	923
OAuth2 Provider Azure AD	925
OAuth2 Provider Private Endpoints	926
OAuth2 Provider Authentication	927
OAuth2 Provider Requests.....	929
JWT	930
TsgcHTTP_JWT_Client	932
TsgcHTTP_JWT_Server.....	935
WebAuthn	936
TsgcWSAPIServer_WebAuthn	937
WebAuthn Registration	941
WebAuthn Registration Request	944
WebAuthn Registration Response	945
WebAuthn Registration Result.....	947
WebAuthn Authentication.....	948
Requisição de Autenticação WebAuthn	950
WebAuthn Authentication Response.....	951
Resultado da autenticação WebAuthn	952
WebAuthn MDS	953
WebAuthn Authorization.....	954
WebAuthn Authorization HTTP	955
WebAuthn Authorization WebSocket	956

Webauthn Javascript Client.....	957
Amazon SQS	961
Google OAuth2 Keys	966
Google Service Accounts	972
Google Cloud Pub/Sub	976
Google Calendar	985
Google Calendar Sync Calendars	992
Google Calendar Sync Events	993
Google Calendar RefreshToken.....	994
Google Calendar Service Account	995
Google Cloud FCM.....	996
TsgcWebView2	999
WebView2 Navigation.....	1002
WebView2 JavaScript.....	1004
WebView2 Cookies	1006
WebView2 Downloads	1008
Configurações do WebView2	1010
Recursos Avançados do WebView2	1012
TsgcUDPCClient	1016
TsgcUDPServer	1017
STUN	1018
TsgcSTUNClient	1019
STUN Client UDP Retransmissions.....	1021
STUN Client Long Term Credentials.....	1022
STUN Client Attributes	1023
TsgcSTUNServer	1024
STUN Server Long Term Credentials	1025
STUN Server Alternate Server.....	1026
TURN.....	1027
TsgcTURNClient	1028
TURN Client Allocate IP Address.....	1030

TURN Client Create Permissions	1031
TURN Client Send Indication.....	1032
Canais do TURN Client.....	1033
TsgcTURNServer	1034
TURN Server Long Term Credentials	1036
TURN Server Allocations.....	1037
ICE	1038
TsgcICEClient.....	1039
ICE Gather Candidates.....	1040
ICE Pair Candidates.....	1041
TsgcRTCPeerConnection	1042
RTCPeerConnection WebSocket Server	1044
RTCPeerConnection WebSocket Client.....	1045
RTCPeerConnection STUN TURN	1046
RTCPeerConnection Signaling	1047
RTCPeerConnection ICE.....	1048
RTCPeerConnection DTLS	1049
RTCPeerConnection Data.....	1050
Datasnap	1051
TsgcWSHTTPWebBrokerBridgeServer	1052
TsgcWSHTTP2WebBrokerBridgeServer	1054
TsgcWSServer_HTTPAPI_WebBrokerBridge.....	1055
OpenAPI	1056
OpenAPI	1056
OpenAPI Parser Pascal	1058
OpenAPI Additional Properties.....	1063
OpenAPI Command Line	1065
OpenAPI Library	1067
OpenAPI API.....	1069
OpenAPI Client	1070

OpenAPI Server	1073
OpenAPI Server Spec-First	1075
OpenAPI Server Code-First.....	1076
OpenAPI Amazon AWS	1077
Credenciais Amazon AWS do OpenAPI.....	1083
OpenAPI Amazon AWS S3	1085
OpenAPI Google Cloud	1086
OpenAPI Google Cloud OAuth2.....	1091
OpenAPI Google Cloud Service Accounts.....	1094
OpenAPI Google Cloud PubSub	1099
OpenAPI Google Cloud Calendar	1100
OpenAPI Google Drive	1101
OpenAPI Microsoft.....	1103
OpenAPI Microsoft Tenant.....	1108
OpenAPI Microsoft Register Application	1109
OpenAPI Microsoft OAuth2 Code.....	1112
OpenAPI Microsoft OAuth2 Credentials.....	1114
OpenAPI Microsoft Graph	1116
APIs	1117
AbstractApi Geolocation.....	1118

Exemplos..... 1119

Chat do Servidor.....	1119
Client Chat.....	1121
Client.....	1122
Cliente MQTT	1123
Cliente SocketIO	1125
Server Monitor.....	1126
Snapshots do Servidor	1129
Client Snapshots.....	1130
Upload de Arquivo	1131

Server Authentication.....	1133
KendoUI_Grid.....	1134
ServerSentEvents	1136
Server WebRTC	1137
Server AppRTC	1138
Telegram Client	1140
Terceiros	1142
Coturn.....	1142
Licença	1144
License	1144
Índice.....	1146
PDF-Back-Cover	1155

Introdução

sgcWebSockets é uma biblioteca de componentes de nível profissional para comunicação em tempo real, mensageria e integração com IA. Ela fornece implementações prontas para produção de WebSockets, HTTP/2, MQTT, AMQP, WebRTC, Server-Sent Events e mais de 30 conectores de API de terceiros, todos acessíveis por meio de uma arquitetura de componentes consistente e orientada a eventos. Quer você esteja construindo plataformas de trading, gateways IoT, aplicações com IA ou sistemas de mensageria corporativa, o sgcWebSockets fornece a base de rede de que você precisa.

A biblioteca suporta **Delphi 7 até Delphi 13**, **C++Builder**, **Lazarus/FreePascal** e **.NET** (Framework 2.0+, .NET Core 1.0+, .NET 5–9, .NET Standard). As aplicações podem ter como alvo Windows, macOS, Linux, iOS e Android a partir de uma única base de código usando VCL e FireMonkey.

Recursos de Delphi / C++Builder / FreePascal

Core WebSocket

- **servidor WebSocket multithread** totalmente funcional, implementando a **RFC 6455**.
- Suporte a mensagens **Text** e **Binary**.
- **Compressão de Mensagens** utilizando a extensão **PerMessage_Deflate (RFC 7692)**.
- Built-in **WatchDog** e **HeartBeat** integrados para reconexão automática e monitoramento de conexão.
- Servidor de **Balanceamento de Carga** para distribuir conexões entre múltiplos backends.
- Componente **Servidor Proxy** que permite que navegadores web se conectem a qualquer servidor TCP.
- Conexões de cliente através de servidores **HTTP Proxy** e **SOCKS Proxy**.
- Eventos: **OnConnect**, **OnDisconnect**, **OnMessage**, **OnError**, **OnHandshake**.

Servidores de Alto Desempenho

- **Microsoft HTTP Server API** (HTTP.SYS) com **IOCP** para servidores Windows de alto desempenho.
- Servidores Indy com **IOCP** (Windows), **EPOLL** (Linux), ou o modelo padrão de uma thread por conexão.
- Conexões **WebSocket** e **HTTP/2** pela **mesma porta**.
- **WebBroker Server** com suporte a **DataSnap**, **HTTP/2** e **WebSocket** em uma única porta.
- Cliente WebSocket baseado na **WinHTTP API**.

Security and Authentication

- **SSL/TLS** para componentes de servidor e cliente. **OpenSSL 1.1.1** e **3.0** suportados. O cliente suporta **SChannel** no Windows.
- Componentes cliente e servidor **OAuth 2.0** e **JWT** (JSON Web Token).
- **WebAuthn Server Protocol** para autenticação baseada em passkey.
- Subprotocolo **E2EE** (criptografia de ponta a ponta).

AI e Machine Learning

- Integração de **OpenAI** e **Claude** com os componentes **MCP Server** e **MCP Client**.
- Componentes de aplicação **ChatBot** e **Translator** com suporte a comandos de voz.
- Suporte a **Embeddings** e **Vector Database** (baseado em arquivo e Pinecone).
- **Text-To-Speech** via padrão do sistema, Google Cloud e Amazon AWS.

HTTP/2 Protocol

- Implementação completa de **cliente e servidor HTTP/2**.
- **Server-Sent Events** (notificações push) sobre HTTP.

Protocolos de Mensagens

- **MQTT 3.1.1** e **5.0** sobre WebSocket e TCP simples.
- **AMQP 0.9.1** e **1.0** com suporte ao broker RabbitMQ.
- **STOMP** sobre WebSocket e TCP (brokers ActiveMQ, RabbitMQ).
- Protocolos cliente e servidor **WAMP 1.0** e **2.0**.
- Subprotocolo **JSON-RPC 2.0**.

P2P e Comunicação em Tempo Real

- **WebRTC** protocolo de servidor.
- **STUN** e **TURN** componentes de cliente e servidor.
- Cliente **ICE** para travessia de NAT.
- Cliente e servidor **UDP**.

Integrações de API

- **Mais de 30 conectores de API integrados:** Binance, Coinbase, Kraken, Bitfinex, Discord, WhatsApp, Telegram e mais.
- Clientes **SignalR** e **SignalR Core**, **Socket.IO** e **Pusher**.
- Serviços em nuvem: **Amazon SQS**, **Google Cloud Pub/Sub**, **Google Calendar**.

Multiplataforma

- **VCL** e **FireMonkey** (Windows, macOS, iOS, Android, Linux).
- **C++Builder** support.
- Suporte a **Lazarus / FreePascal**.

- **Bibliotecas JavaScript** integradas para clientes de navegador.

Os seguintes componentes estão incluídos na biblioteca sgcWebSockets:

1 sgcWebSockets

- **TsgcWebSocketClient**: Cliente WebSocket baseado na biblioteca Indy.
- **TsgcWebSocketServer**: Servidor WebSocket baseado na biblioteca Indy
- **TsgcWebSocketHTTPServer**: Servidor WebSocket + HTTP baseado na biblioteca Indy.
- **TsgcWebSocketServer_HTTPAPI**: Servidor WebSocket + HTTP de Alto Desempenho baseado na HTTP.SYS Microsoft HTTP API.
- **TsgcWebSocketClient_WinHTTP**: Cliente WebSocket baseado na biblioteca WinHTTP.
- **TsgcWebSocketFirewall**: Firewall de Servidor WebSocket com filtragem de IP, proteção contra força bruta, limitação de taxa e detecção de ameaças.
- **TsgcWebSocketLoadBalancerServer**: Servidor Balanceador de Carga para os protocolos WebSocket e HTTP.
- **TsgcWebSocketProxyServer**: Servidor Proxy WebSocket que permite que navegadores web se conectem a servidores TCP.

2 sgcWebSocket APIs

- **TsgcWSAPI_Binance**: Cliente Binance Spot, suporta APIs WebSocket + REST.
- **TsgcWSAPI_Binance_Futures**: Cliente Binance Futures, suporta APIs WebSocket + REST.
- **TsgcWSAPI_SocketIO**: Cliente Socket.IO.
- **TsgcWSAPI_Coinbase**: Cliente Coinbase Pro, suporta APIs WebSocket + REST.
- **TsgcWSAPI_Bitmex**: Cliente Bitmex, suporta APIs WebSocket + REST.
- **TsgcWSAPI_SignalR**: Cliente WebSocket SignalR.
- **TsgcWSAPI_SignalRCore**: Cliente WebSocket SignalRCore.
- **TsgcWSAPI_Pusher**: Cliente WebSocket Pusher.
- **TsgcWSAPI_Kraken**: API Cliente Kraken, suporta WebSocket e REST Api.
- **TsgcWSAPI_Kraken_Futures**: API Cliente Kraken Futures, suporta WebSocket e REST Api.
- **TsgcWSAPI_Bitstamp**: Cliente WebSocket Bitstamp.
- **TsgcWSAPI_Cex**: Cliente WebSocket Cex.
- **TsgcWSAPI_Huobi**: Cliente WebSocket Huobi.
- **TsgcWSAPI_ThreeCommas**: API Cliente ThreeCommas.
- **TsgcWSAPI_Bitfinex**: API WebSocket Bitfinex.
- **TsgcWSAPI_Discord**: Cliente WebSocket Discord.
- **TsgcWSAPI_BlockChain**: Cliente WebSocket BlockChain.
- **TsgcWSAPI_Bybit**: Cliente WebSocket Bybit.
- **TsgcWSAPI_OKX**: Cliente WebSocket OKX.
- **TsgcWSAPI_Kucoin**: Cliente WebSocket KuCoin Spot.
- **TsgcWSAPI_Kucoin_Futures**: Cliente WebSocket KuCoin Futures.
- **TsgcWSAPI_Deribit**: Cliente WebSocket Deribit.
- **TsgcWSAPI_CryptoCom**: Cliente WebSocket Crypto.com.
- **TsgcWSAPI_HTX**: Cliente WebSocket HTX.
- **TsgcWSAPI_MEXC**: Cliente WebSocket MEXC.

- **TsgcWSAPI_Bitget**: Cliente WebSocket Bitget.
- **TsgcWSAPI_GateIO**: Cliente WebSocket Gate.io.
- **TsgcWSAPI_XTB**: Cliente WebSocket XTB.
- **TsgcWSAPI_CexPlus**: Cliente WebSocket CexPlus.
- **TsgcWSAPI_OpenAI**: Cliente da OpenAI Realtime WebSocket API.

3 sgcWebSocket Libs

- **TsgcTDLib_Telegram**: Cliente da API do Telegram.
- **TsgcWhatsApp_Client**: Cliente WhatsApp Business Cloud.
- **TsgcHTTP_Cryptohopper**: Cliente da API do Cryptohopper.
- **TsgcLib_RCON**: Cliente RCON.

4 Protocolos sgcWebSocket

- **TsgcWSPClient_MQTT**: Cliente MQTT (3.1.1 e 5.0). Suporta conexões WebSocket e TCP simples.
- **TsgcWSPClient_AMQP1**: Cliente AMQP 1.0.0. Suporta brokers RabbitMQ.
- **TsgcWSPClient_AMQP**: Cliente AMQP 0.9.1. Suporta brokers RabbitMQ.
- **TsgcWSPClient_STOMP**: Cliente STOMP, suporta conexões WebSocket e TCP simples.
 - **TsgcWSPClient_STOMP_ActiveMQ**: Cliente STOMP para o broker ActiveMQ.
 - **TsgcWSPClient_STOMP_RabbitMQ**: Cliente STOMP para o broker RabbitMQ.
- **TsgcWSPClient_WAMP**: Protocolo Cliente WAMP 1.0.
- **TsgcWSPServer_WAMP**: Protocolo Servidor WAMP 1.0.
- **TsgcWSPClient_WAMP2**: Protocolo Cliente WAMP 2.0.
- **TsgcWSPServer_AppRTC**: Servidor WebRTC baseado no projeto AppRTC do Google.
- **TsgcWSPServer_WebRTC**: Protocolo Servidor WebRTC.
- **TsgcWSPClient_sgc**: Protocolo Cliente WebSocket SGC baseado em JSON RPC.
- **TsgcWSPServer_sgc**: Protocolo Servidor WebSocket SGC baseado em JSON RPC.
- **TsgcWSPClient_Files**: Protocolo Cliente WebSocket de Transferência de Arquivos.
- **TsgcWSPServer_Files**: Protocolo Servidor WebSocket de Transferência de Arquivos.
- **TsgcWSPClient_Dataset**: Protocolo Cliente WebSocket de Sincronização de Dataset.
- **TsgcWSPServer_Dataset**: Protocolo Servidor WebSocket de Sincronização de Dataset.
- **TsgcWSPClient_Presence**: Protocolo Cliente WebSocket de Presença.
- **TsgcWSPServer_Presence**: Protocolo Servidor WebSocket de Presença.
- **TsgcWSPClient_E2EE**: Protocolo Cliente de Criptografia de Ponta a Ponta.
- **TsgcWSPServer_E2EE**: Protocolo Servidor de Criptografia de Ponta a Ponta.

5 sgcWebSockets HTTP

- **TsgcHTTP1Client**: Cliente HTTP 1.0 baseado no Indy TIdHTTP.
- **TsgcHTTP2Client**: Cliente HTTP 2.0.
- **TsgcHTTP_JWT_Client**: Cliente JWT (JSON WEB TOKEN).
- **TsgcHTTP_JWT_Server**: Servidor JWT (JSON WEB TOKEN).
- **TsgcHTTP_OAuth2_Client**: Cliente OAuth 2.0.
- **TsgcHTTP_OAuth2_Server**: Servidor OAuth 2.0.
- **TsgcHTTPAWS_SQS_Client**: Cliente Amazon AWS SQS.

- **TsgcHTTPGoogleCloud_PubSub_Client**: Cliente Google Cloud Pub/Sub.
- **TsgcHTTPGoogleCloud_Calendar_Client**: Cliente Google Calendar.
- **TsgcHTTP_OAuth2_Client_Google**: Cliente OAuth 2.0 para Google.
- **TsgcHTTP_OAuth2_Client_Microsoft**: Cliente OAuth 2.0 para Microsoft.
- **TsgcHTTP_OAuth2_Server_Provider**: Provedor de Servidor OAuth 2.0.
- **TsgcWSAPIServer_WebAuthn**: Servidor WebAuthn para autenticação baseada em passkey.
- **TsgcHTTPGoogleCloud_FCM_Client**: Cliente Google Cloud FCM (Firebase Cloud Messaging).

6 sgcWebSockets IoT

- **TsgcIoTAmazon_MQTT_Client**: Cliente Amazon MQTT IoT Core.
- **TsgcIoTAzure_MQTT_Client**: Cliente Azure IoT MQTT.

7 sgcWebSockets P2P

- **TsgcUDPClient**: Cliente UDP.
- **TsgcUDPServer**: Servidor UDP.
- **TsgcSTUNClient**: Cliente STUN.
- **TsgcSTUNServer**: Servidor STUN.
- **TsgcTURNClient**: Cliente STUN / TURN.
- **TsgcTURNServer**: Servidor STUN / TURN.
- **TsgcICEClient**: Cliente ICE.
- **TsgcRTCPeerConnection**: Cliente WebRTC Peer Connection.

8 sgcWebSockets DataSnap

- **TsgcWSHTTPWebBrokerBridgeServer**: Substituição de Servidor DataSnap com suporte a HTTP + WebSockets.
- **TsgcWSHTTP2WebBrokerBridgeServer**: Substituição de Servidor DataSnap com suporte a HTTP + HTTP/2 + WebSockets.
- **TsgcWSServer_HTTPAPI_WebBrokerBridge**: Substituição de Servidor DataSnap baseada no servidor HTTP.SYS da Microsoft.

9 sgcWebSockets AI

- **TsgcAIOpenAIChatBot**: Crie um ChatBot com comandos de voz.
- **TsgcAIOpenAITranslator**: Tradução em tempo real.
- **TsgcAudioRecorderMCI**: Grava áudio utilizando MCI.
- **TsgcAudioPlayerMCI**: Reproduz áudio utilizando MCI.
- **TsgcTextToSpeechSystem**: Text-To-Speech utilizando o padrão do sistema operacional.
- **TsgcTextToSpeechGoogle**: Text-To-Speech utilizando a Google Cloud.
- **TsgcTextToSpeechAmazon**: Text-To-Speech utilizando a Amazon AWS.
- **TsgcAIOpenAIEmbeddings**: permite que você utilize seus dados personalizados para criar aplicações de IA.
- **TsgcAIDatabaseVectorFile**: armazena os vetores em um arquivo de texto simples.
- **TsgcAIDatabaseVectorPinecone**: suporta o banco de dados vetorial Pinecone.
- **TsgcAIOpenAIAssistant**: Cliente da OpenAI Assistants API.
- **TsgcWSAPIServer_MCP**: Servidor MCP (Model Context Protocol).

- **TsgcWSAPIClient_MCP:** Cliente MCP (Model Context Protocol).

Versions Support

IDE suportada pelo Delphi

- Delphi 7 (* suportado somente se atualizado para Indy 10; IntraWeb não é suportado)
- Delphi 2007
- Delphi 2009
- Delphi 2010
- Delphi XE
- Delphi XE2
- Delphi XE3
- Delphi XE4
- Delphi XE5
- Delphi XE6
- Delphi XE7
- Delphi XE8
- Delphi 10 Seattle
- Delphi 10.1 Berlin
- Delphi 10.2 Tokyo
- Delphi 10.3 Rio
- Delphi 10.4 Sydney
- Delphi 11 Alexandria
- Delphi 12 Athens
- Delphi 13 Florence

IDE suportada pelo CBuilder

- CBuilder 2007
- CBuilder 2010
- CBuilder XE
- CBuilder XE2
- CBuilder XE3
- CBuilder XE4
- CBuilder XE5
- CBuilder XE6
- CBuilder XE7
- CBuilder XE8
- CBuilder 10 Seattle
- CBuilder 10.1 Berlin
- CBuilder 10.2 Tokyo
- CBuilder 10.3 Rio
- CBuilder 10.4 Sydney
- CBuilder 11 Alexandria
- CBuilder 12 Athens
- CBuilder 13 Florence

IDE suportada pelo FreePascal

- Lazarus

Versão Trial

Os arquivos *.dcu compilados fornecidos com a versão gratuita utilizam as versões padrão do Indy e do IntraWeb. Se você atualizou algum desses pacotes, eles provavelmente não funcionarão e você precisará comprar a versão de código-fonte completo.

Pacote Indy

Alguns componentes utilizam o Indy como sua biblioteca TCP/IP (como TsgcWebsocketClient ou TsgcWebsocket-Server), o que significa que o Indy é necessário para instalar o pacote sgcWebSockets. Por padrão, o sgcWebSockets utiliza a biblioteca Indy embutida no RAD Studio, mas fornecemos uma versão personalizada do Indy que tem mais recursos: suporte a OpenSSL API 1.1, OpenSSL 3.0, protocolo ALPN...

Instalação

Delphi / CBuilder / Lazarus

1. Descompacte os arquivos incluídos em um diretório {\$DIR}

2. A partir do Delphi\CBuilder:

Adicione o diretório onde os arquivos foram descompactados {\$DIR} ao library path do Delphi\CBuilder em Tools, Environment options, Directories

Todas as versões do Delphi\CBuilder

Adicione o diretório {\$DIR}\source ao caminho da biblioteca

Para uma versão específica do Delphi

Delphi 7 : Adicione o diretório {\$DIR}\libD7 ao library path
 Delphi 2007 : Adicione o diretório {\$DIR}\libD2007 ao library path
 Delphi 2009 : Adicione o diretório {\$DIR}\libD2009 ao caminho de biblioteca
 Delphi 2010 : Adicione o diretório {\$DIR}\libD2010 ao library path
 Delphi XE : Adicione o diretório {\$DIR}\libDXE ao library path
 Delphi XE2 : Adicione o diretório {\$DIR}\libDXE2\\$(Platform) ao caminho de bibliotecas
 Delphi XE3 : Adicione o diretório {\$DIR}\libDXE3\\$(Platform) ao library path
 Delphi XE4 : Adicione o diretório {\$DIR}\libDXE4\\$(Platform) ao library path
 Delphi XE5 : Adicione o diretório {\$DIR}\libDXE5\\$(Platform) ao caminho de bibliotecas
 Delphi XE6 : Adicione o diretório {\$DIR}\libDXE6\\$(Platform) ao caminho de bibliotecas
 Delphi XE7 : Adicione o diretório {\$DIR}\libDXE7\\$(Platform) ao library path
 Delphi XE8 : Adicione o diretório {\$DIR}\libDXE8\\$(Platform) ao library path
 Delphi 10 : Adicione o diretório {\$DIR}\libD10\\$(Platform) ao caminho da biblioteca
 Delphi 10.1 : Adicione o diretório {\$DIR}\libD10_1\\$(Platform) ao caminho de bibliotecas
 Delphi 10.2 : Adicione o diretório {\$DIR}\libD10_2\\$(Platform) ao library path
 Delphi 10.3 : Adicione o diretório {\$DIR}\libD10_3\\$(Platform) ao caminho de bibliotecas
 Delphi 10.4 : Adicione o diretório {\$DIR}\libD10_4\\$(Platform) ao caminho da biblioteca
 Delphi 11 : Adicione o diretório {\$DIR}\libD11\\$(Platform) ao caminho de bibliotecas
 Delphi 12 : Adicione o diretório {\$DIR}\libD12\\$(Platform) ao caminho de bibliotecas
 Delphi 13 : Adicione o diretório {\$DIR}\libD13\\$(Platform) ao caminho de bibliotecas

Para uma versão específica do CBuilder

C++ Builder 2010 : Adicione o diretório {\$DIR}\libD2010 ao caminho de bibliotecas
 C++ Builder XE : Adicione o diretório {\$DIR}\libDXE ao library path
 C++ Builder XE2 : Adicione o diretório {\$DIR}\libDXE2\\$(Platform) ao library path
 C++ Builder XE3 : Adicione o diretório {\$DIR}\libDXE3\\$(Platform) ao caminho da biblioteca
 C++ Builder XE4 : Adicione o diretório {\$DIR}\libDXE4\\$(Platform) ao caminho de bibliotecas
 C++ Builder XE5 : Adicione o diretório {\$DIR}\libDXE5\\$(Platform) ao library path
 C++ Builder XE6 : Adicione o diretório {\$DIR}\libDXE6\\$(Platform) ao caminho de bibliotecas
 C++ Builder XE7 : Adicione o diretório {\$DIR}\libDXE7\\$(Platform) ao caminho de bibliotecas
 C++ Builder XE8 : Adicione o diretório {\$DIR}\libDXE8\\$(Platform) ao caminho de biblioteca
 C++ Builder 10 : Adicione o diretório {\$DIR}\libD10\\$(Platform) ao caminho da biblioteca
 C++ Builder 10.1 : Adicione o diretório {\$DIR}\libD10_1\\$(Platform) ao caminho de bibliotecas
 C++ Builder 10.2 : Adicione o diretório {\$DIR}\libD10_2\\$(Platform) ao library path
 C++ Builder 10.3 : Adicione o diretório {\$DIR}\libD10_3\\$(Platform) ao caminho de bibliotecas
 C++ Builder 10.4 : Adicione o diretório {\$DIR}\libD10_4\\$(Platform) ao caminho da biblioteca
 C++ Builder 11 : Adicione o diretório {\$DIR}\libD11\\$(Platform) ao caminho de bibliotecas
 C++ Builder 12 : Adicione o diretório {\$DIR}\libD12\\$(Platform) ao caminho de biblioteca

C++ Builder 13 : Adicione o diretório {\$DIR}\libD13\\$(Platform) ao library path

Para todas as versões do CBuilder, adicione dcp\\$(Platform) ao library path (contém arquivos .bpi)

3. A partir do Delphi

Escolha

File, Open e navegue até o Packages\sgcWebSockets.groupproj correto (Primeiro compile sgcWebSocketsX.dpk e então instale dclsgcWebSocketsX.dpk)

arquivos de pacotes para Delphi

sgcWebSocketsD7.groupproj : Delphi 7
 sgcWebSocketsD2007.groupproj : Delphi 2007
 sgcWebSocketsD2009.groupproj : Delphi 2009
 sgcWebSocketsD2010.groupproj : Delphi 2010
 sgcWebSocketsDXE.groupproj : Delphi XE
 sgcWebSocketsDXE2.groupproj : Delphi XE2
 sgcWebSocketsDXE3.groupproj : Delphi XE3
 sgcWebSocketsDXE4.groupproj : Delphi XE4
 sgcWebSocketsDXE5.groupproj : Delphi XE5
 sgcWebSocketsDXE6.groupproj : Delphi XE6
 sgcWebSocketsDXE7.groupproj : Delphi XE7
 sgcWebSocketsDXE8.groupproj : Delphi XE8
 sgcWebSocketsD10.groupproj : Delphi 10
 sgcWebSocketsD10_1.groupproj : Delphi 10.1
 sgcWebSocketsD10_2.groupproj : Delphi 10.2
 sgcWebSocketsD10_3.groupproj : Delphi 10.3
 sgcWebSocketsD10_4.groupproj : Delphi 10.4
 sgcWebSocketsD11.groupproj : Delphi 11
 sgcWebSocketsD12.groupproj : Delphi 12
 sgcWebSocketsD13.groupproj : Delphi 13

4. A partir do CBuilder

Escolha

File, Open e navegue até o Packages\sgcWebSockets.groupproj correto (Primeiro compile sgcWebSocketsX.dpk e então instale dclsgcWebSocketsX.dpk)

arquivos de pacotes para o CBuilder

sgcWebSocketsC2010.groupproj : C++ Builder 2010
 sgcWebSocketsCXE.groupproj : C++ Builder XE
 sgcWebSocketsCXE2.groupproj : C++ Builder XE2
 sgcWebSocketsCXE3.groupproj : C++ Builder XE3
 sgcWebSocketsCXE4.groupproj : C++ Builder XE4
 sgcWebSocketsCXE5.groupproj : C++ Builder XE5
 sgcWebSocketsCXE6.groupproj : C++ Builder XE6
 sgcWebSocketsCXE7.groupproj : C++ Builder XE7
 sgcWebSocketsCXE8.groupproj : C++ Builder XE8
 sgcWebSocketsC10.groupproj : C++ Builder 10
 sgcWebSocketsC10_1.groupproj : C++ Builder 10.1
 sgcWebSocketsC10_2.groupproj : C++ Builder 10.2
 sgcWebSocketsC10_3.groupproj : C++ Builder 10.3
 sgcWebSocketsC10_4.groupproj : C++ Builder 10.4
 sgcWebSocketsC11.groupproj : C++ Builder 11
 sgcWebSocketsC12.groupproj : C++ Builder 12
 sgcWebSocketsC13.groupproj : C++ Builder 13

5. A partir do Lazarus

Escolha: File, Open e navegue até `Packages\sgcWebSocketsLazarus.lpk` (primeiro compile e depois instale)

Os arquivos compilados estão localizados no diretório Lazarus. Dentro dele, há um diretório Indy com a última versão do código-fonte do Indy.

Testado com Lazarus 2.0.6 e Indy 10.5.9.4930

6. Demos

Todos os demos estão disponíveis no subdiretório Demos. Basta abrir o projeto e executá-lo. Os demos do IntraWeb podem exigir a modificação de algumas units devido às diferentes versões do IntraWeb.

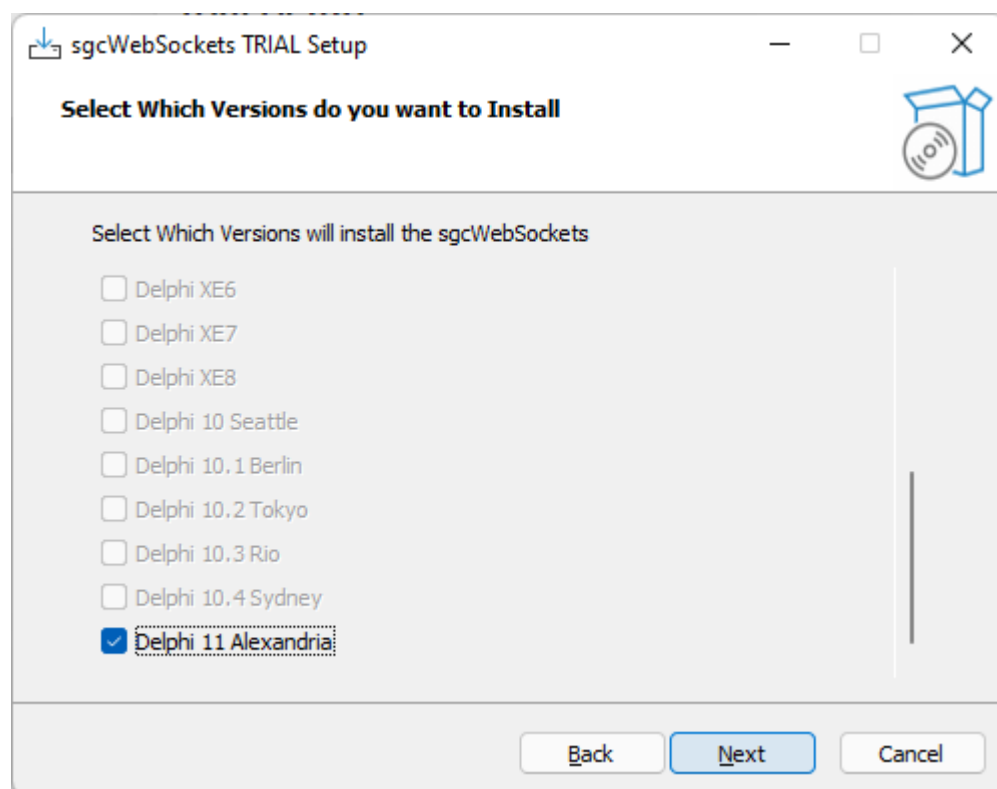
Install Setup

*Requer Windows Vista no mínimo (Windows 2000, XP e Server 2003 não são suportados).

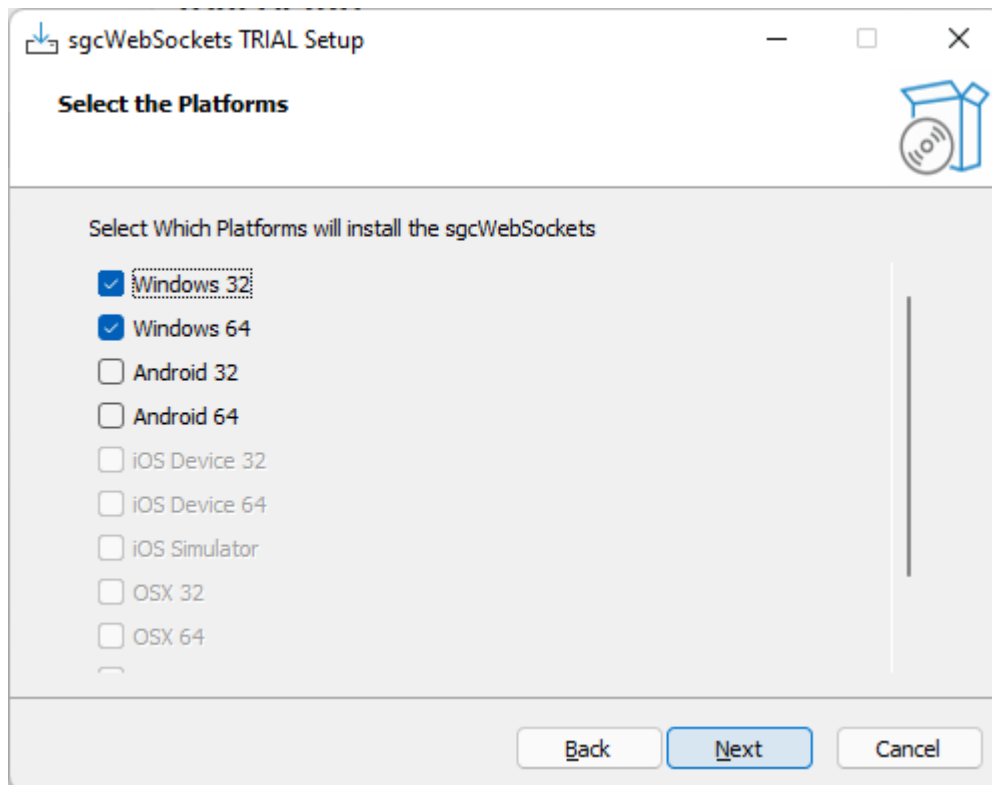
Se você usar o Windows Setup para instalar a biblioteca sgcWebSockets, a instalação é guiada e muito simples. Se houver algum erro durante a instalação, consulte a página [Install Errors](#) e você pode tentar [instalar o pacote manualmente](#).

Trial Setup

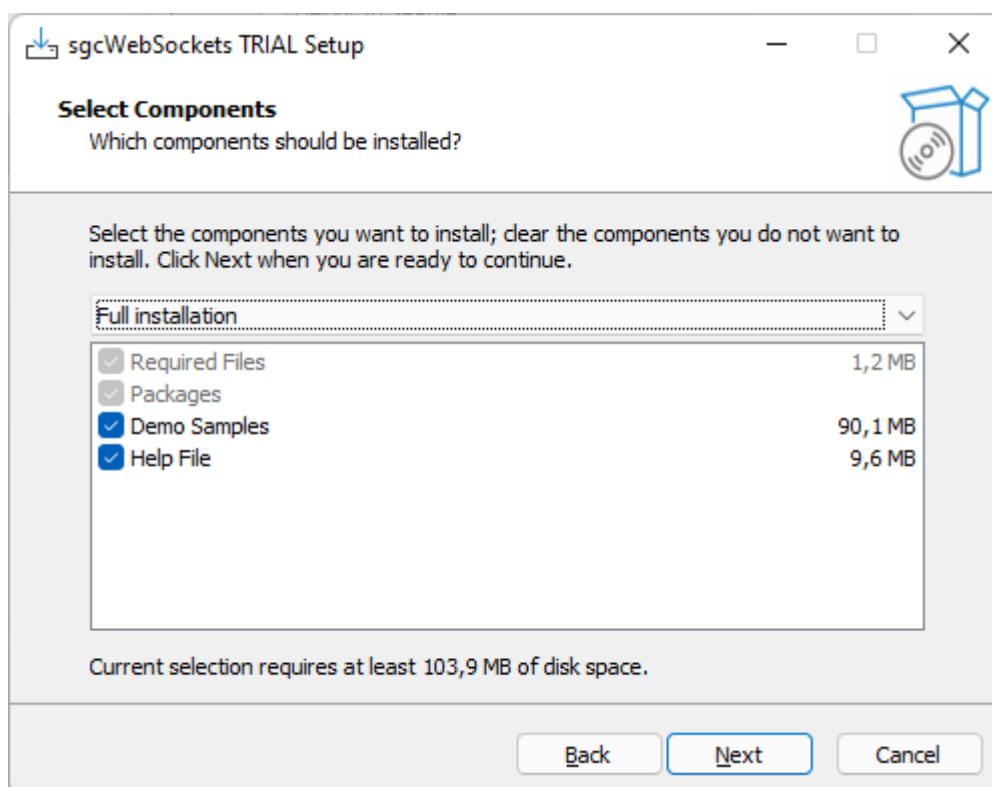
- Execute o Trial Installer.
- O instalador do Trial requer privilégios de Administrador.
- O instalador mostrará uma lista de versões do Delphi / CBuilder / Rad Studio e, por padrão, a versão baixa-da será habilitada. Se esta versão NÃO for detectada pelo instalador, o instalador extrairá os arquivos mas não tentará compilar. Consulte [instalar o pacote manualmente](#).



- A próxima página mostra as Plataformas que podem ser instaladas, apenas aquelas plataformas detectadas pelo instalador são habilitadas.



- A próxima página mostra o contrato de licença que deve ser aceito para instalar a trial.
- Depois de aceitar o contrato de licença, são exibidos os componentes que serão instalados. Por padrão, o pacote, os DCUs compilados, os demos e os arquivos de ajuda serão todos instalados. Você pode personalizar se os arquivos de ajuda e os demos serão instalados ou não.

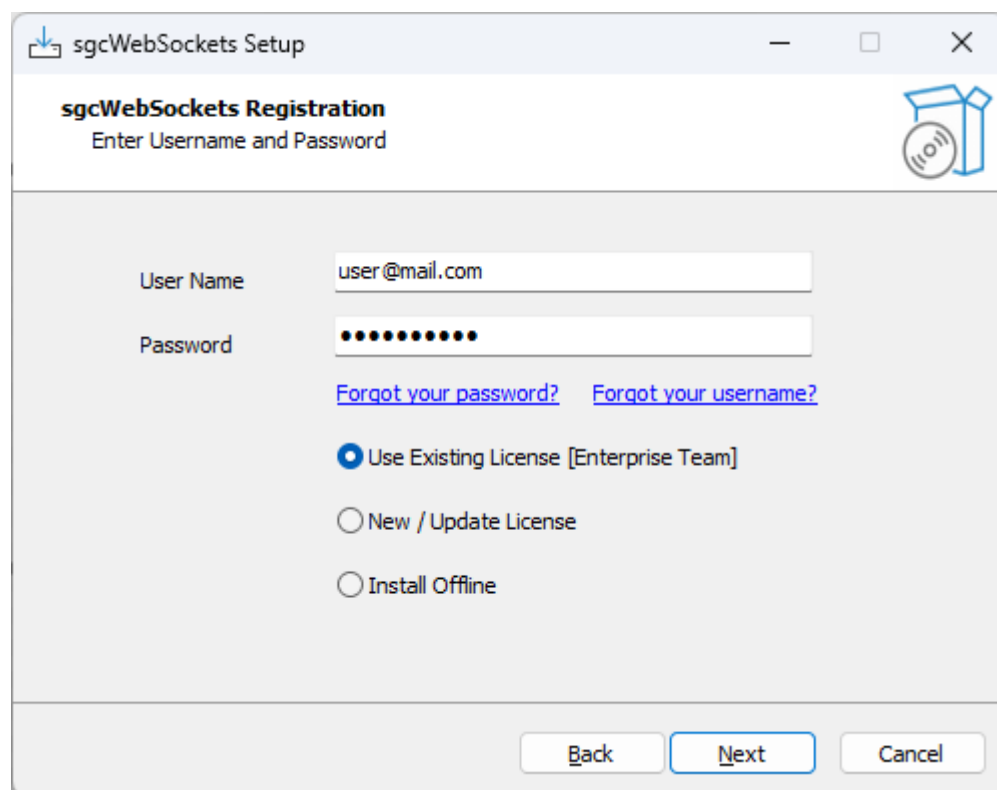


- Por fim, ele extrairá os arquivos, compilará e instalará o pacote e registrará os caminhos necessários na IDE.

Configuração de Clientes

Os usuários que adquiriram uma licença podem instalar a biblioteca sgcWebSockets usando o instalador. Encontre abaixo as instruções passo a passo para instalar o pacote.

- Execute o Instalador.
- O instalador é executado com os privilégios mais baixos (se for executado como admin, não pode ser instalado em drives de rede). Se o destino exigir privilégios de admin, execute a instalação como administrador.
- Primeiro você deve definir o nome de usuário/senha da sua conta privada eSeGeCe. Isso só precisa ser inserido uma vez; na próxima vez que você utilizar o setup, o instalador lerá o valor previamente salvo.

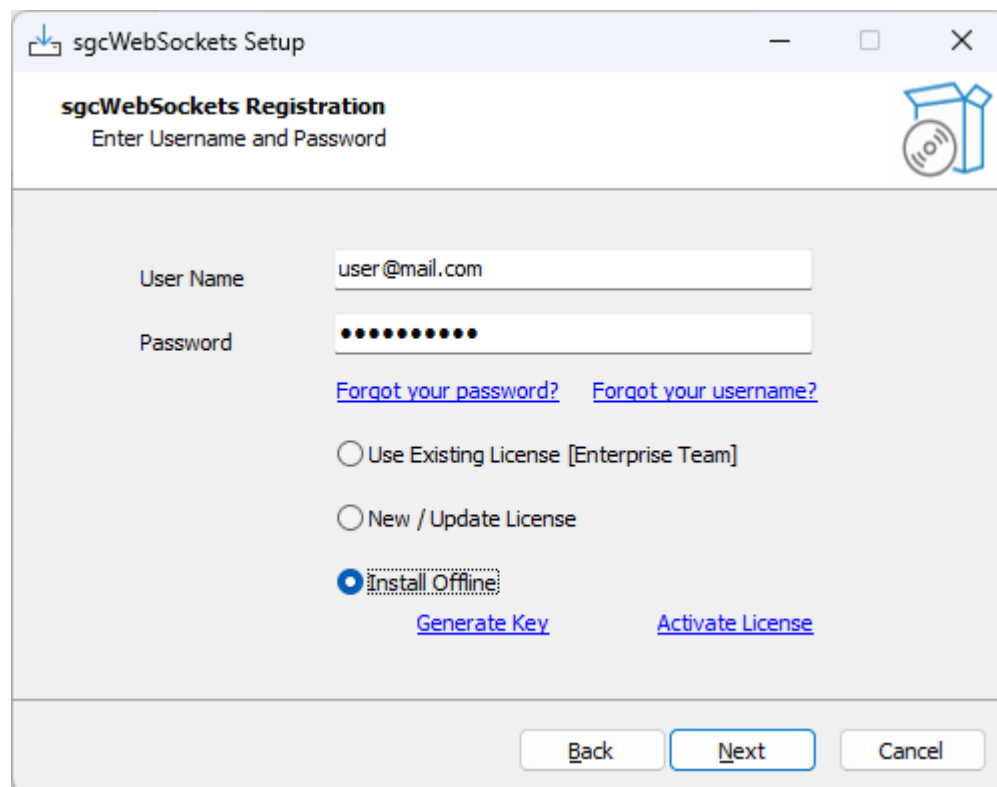


The screenshot shows the 'sgcWebSockets Setup' window. The title bar says 'sgcWebSockets Setup'. The main window has a header 'sgcWebSockets Registration' with the subtitle 'Enter Username and Password'. There is a blue icon of a box with a coin in the top right corner. The form contains two input fields: 'User Name' with the text 'user@mail.com' and 'Password' with masked characters. Below the password field are two links: 'Forgot your password?' and 'Forgot your username?'. There are three radio button options: 'Use Existing License [Enterprise Team]' (selected), 'New / Update License', and 'Install Offline'. At the bottom, there are three buttons: 'Back', 'Next' (highlighted with a blue border), and 'Cancel'.

- Existem 3 opções:
 - **Use Existing License:** se esta versão já tiver sido instalada, a opção será selecionada por padrão. Ela utilizará a configuração mais recente para esta versão.
 - **New / Update License:** se esta versão não tiver sido instalada anteriormente, esta opção será selecionada por padrão. Ela conectará ao Server License para obter as informações de licença. Se você atualizou sua licença recentemente, pode selecionar esta opção para atualizar a licença.
 - **Install Offline:** se a máquina não tiver acesso à internet, selecione esta opção para ativar sua licença.

Gerar chave

Esta opção gera uma chave que será utilizada para ativar a licença.



sgcWebSockets Registration
Enter Username and Password

User Name:

Password:

[Forgot your password?](#) [Forgot your username?](#)

☐ Use Existing License [Enterprise Team]
☐ New / Update License
☒ **Install Offline**

[Generate Key](#) [Activate License](#)

Back Next Cancel

Copie a chave e acesse sua conta online privada: www.eSeGeCe.com/my-account/subscriptions.



Generate Key - eSeGeCe Software (2023.8.0)

VjJ4b2MxTXlVWghpUm1oVltdHdZVlpXU2xOak1VNVlZMGhhYUZZd01UVlhhMIzVmpBeGMxZHVRbGRXUldzeFdWY3hSbVZXV25GUmJYQnNWbGQ0TmxVeFZrOVRNREZ6WTBac1ZHSnVRbTlXym5CelRWWk9XR05JV21GTlIzaDRWVEZvYTFSDfJuSk5WRVpWVFVvd2QxcEVRbk5qUjFGNVdrVndhVlI6YUhsVk1XUjNaR3h2ZDJKSvJ3sUlHSM2hvVld0YWQyTnNaRmhOVmtwT1RvAG9XVll4VWt0WlZscEdVbXBDVldFeFNsQlphMVV4VmtVeFdHskZOVmRTVlhCM1ZrVmFVMU5yTVZaTlNHeFRWa1phVUZsWE1VOUSNVkoxWTBoT2FWWlVSVEZWTVdoUfUyeEZkMk6YUzwbGEzQlIXbFprUzJSR1NuVlZiVpXVZzhMlF3VFRGU2NsVnJTazlTVkZaV1ZteG9hMVZHV2taU2FsSlZWbFpLZfZSc1ZYaFNWa1pWVld4Q1YxSXpVWHBxUkVaVFVXMVdjazFWVm1GbGEwcFBXVmQwUmsxR1VYaFZiSEJyVVFVSc1ZsVldVbGRVYkVwSVpFuk9WVTFHU2tOVWExVTFVbFpTV1ZwR1FsTlNNRFF4VmtaV1UySnJNWEpQVmxalU1ZrWktVRmxYTVU5Tk1WRjRwV3RhYTAXRVJrWlpWRTV2VkrKR2NsSlVSbFZXykVweFdrUkJOVlpXU2xWVGF6VIRVak5STUZaSE1YZFJheIZXVFZWV1dGZEhVbEZWYTFwv1RWWlNWbFZ0TlU1aGvSvldWVlpTVTFReVJuUmtSRTVWVFVaS1lWUnNWWGhTVm5BMLUyeENVMUpWVlhkV1J6RjNzbTFSZDA5VlZsaFdSWEJQVld0a2FtVldVWGRWYTbwUFVsUlZnBfZyYUd0VlJrcElWR3BPVkJaNVVqTlZSa1U1VUZFOVBRPT0=

Selecione a inscrição a ser ativada e cole a chave.

Offline License

TKVWSvpr0K3VVVH10ZtOVVEXVH1V0rptV1ZWkH1SHNNK114Vmtav10y3HnWLEpqVmxao1Z1VWkVkmX
YTVU5Tk1WRjRWV3RHYTAxRVJrWlpWRTV2VkrKR2NsSIVSbFZXyKVweFdrUKJOVlpXU2xWVGF6VIRVak5S
TUZaSE1YZFJhelZXVFZWV1dGZEHVbEZWYTFwV1RWWINWbFZ0TIU1aGVsWldWVlpTVTFReVJuUmtSRTV
WVfVaS1WUnNWWGhTVm5BMLUyeENVMUUpWVlhkV1J6RJNZbTFSZDA5VIZsaFdSWEJQVld0a2FtVldVblJqUI
VwUFVsUldRmxVVG10VJrcEIWR3BhVkZaNilVqTIZSa1U1VUZFOVBRPT0=

Activate License

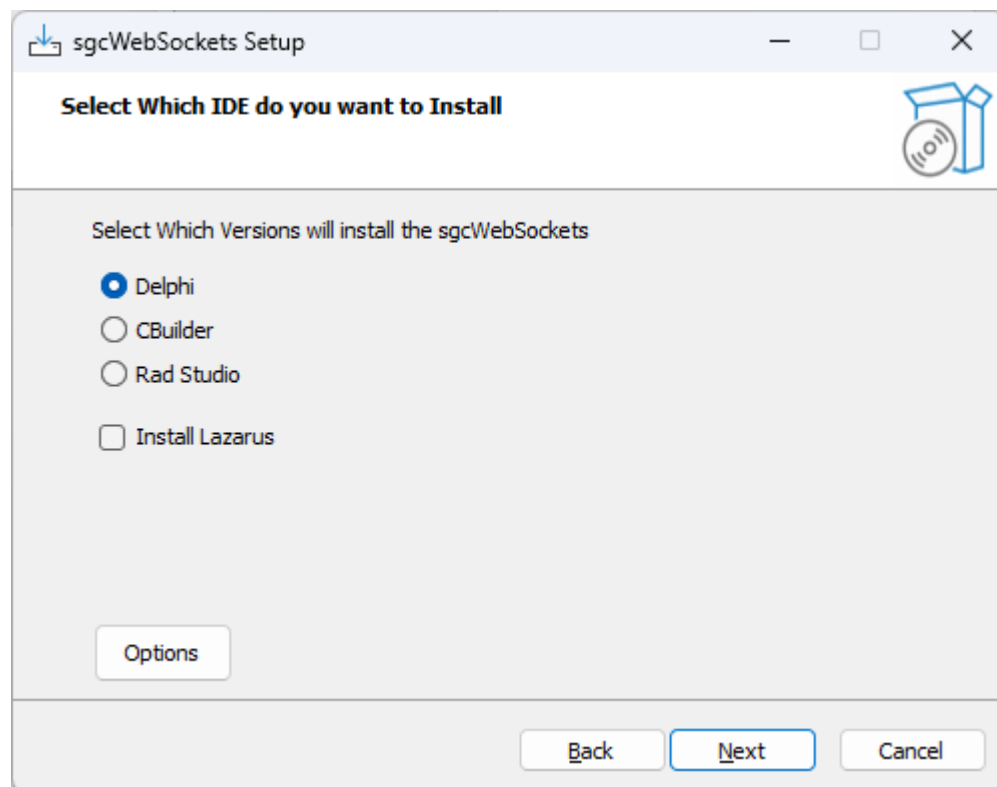
Ativar licença

Se a requisição estiver correta, ela retornará uma licença que deve ser copiada no setup.

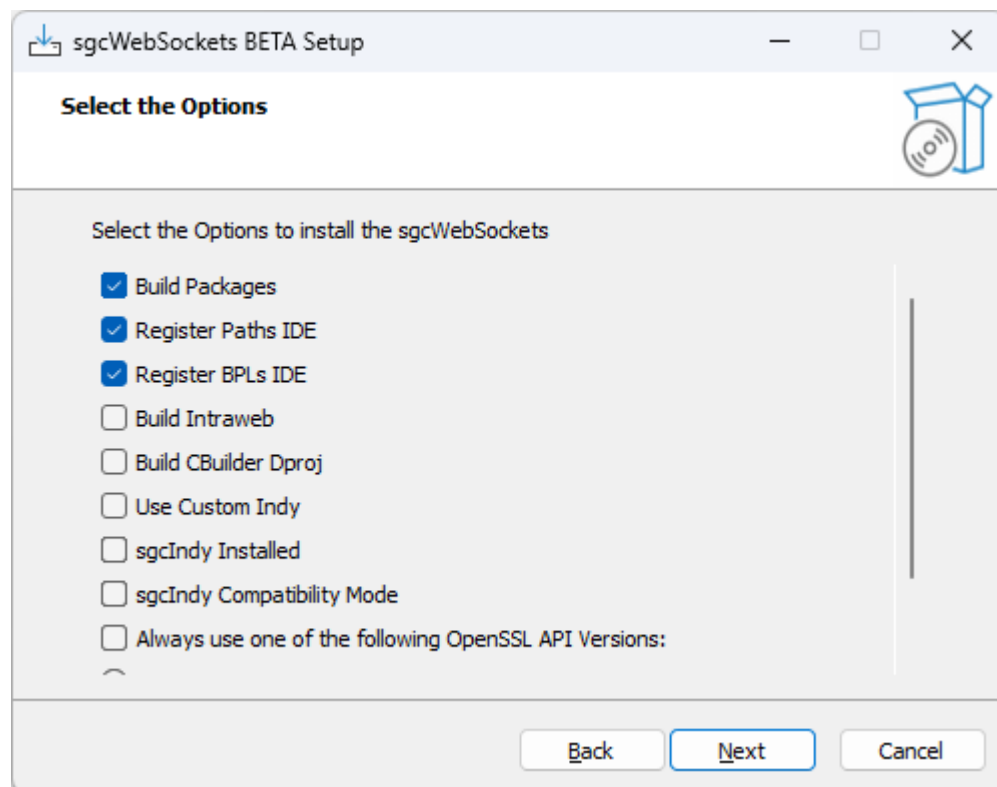
Activate License - eSeGeCe Software (2023.8.0)

VmRTVlhCM1ZrVmFVMU5yTVZaTlNHeFRWa1phVUZsWE1VOU5NVkoxWTBoT2FWWWVSVZEZWTvdofUfUyeEZkMkl6YUzwbGEzQlXbFprUzJSR1Nu
VIZiVpXVfVkemVWVXhWazlUTURGWVZGaHNWmkp0ZUZOVMJuQkhZakZzVjFwRVRtdFdiWFEXVkrGb1UxTnRSbGxhUnpWVWZucFdSRmxWWk
U1bGJGSnhWRzFzVGSxSVFubFZNV1IzWkd4dmQySkUbfZotW1oVFZGVldTMIzHYkhGVFZGWNJvakJ3U1ZWc1pITIVWa1YzVW01Q1lWSjZSa2ha
ZWtwSFYwWldkRmR0Um1saE0wRjVWMWQ0VDFock5YSmlSVkpVvJBkb1VGBFhNWHBOUmxKMFkwWmFIRlpVmxawFdIQLBVMjFHV1ZwSE5WUldlbFpRV1RCYWRtVIZOVmhoUjNSVvVteHZNvII5ZUU5VGF6VnlZa1ZTVkZkSFVsQmFWM1J5VFZaU1ZsVnROVTVTVkZaV1ZWWVnVMVJ0Vm5OU2Fr
NVZUVVpLVDFwVlZYaFNWbTk2WWvtVMUxSlZWWGRXUm1oM1ZtMVJkMDIwVmxWVFNFSIBWV3RrYTA1R1VYafZhmHBQVWxSV1JsZHFubXR
WUmtwR1RvaHdWVlpXU25WVWZVWVXhZMVpHVIZWck9WTINWbGw2VmtSQ1UxWnJNSGR0VIZaafXpXdfGVrRnBYZEVabGJGRjRWV3MxYTAXRVZ
USlpWRTV6Vkd4S1NGULVSBfZpUmtwRFZGukJOVlpXYTNwYVJUBHNZbGhSTWxaR1ZsTmliVkyZVGxVmRTVlhCM1ZrVmFVMU5yTVZaTlNHeFRWa
1phVUZsWE1VOU5NVkoxWTBoT2FWWWVSVZEZWTvdofUfUyeEZkMkl6YUzwbGEzQlXbFprUzJSR1NuVIZiVpXVfVkemVWVXhWazlUTURGWVZGa
HNWmkp0ZUZOVMJuQkhZakZzVjFwRVRtdFdiWFEXVkrGb1UxTnRSbGxhUnpWVWZucFdSRmxWWkU1bGJGSnhWRzFzVGSxSVFubFZNV1IzWkd
4dmQySkUbfZotW1oVFZGVldTMIzHYkhGVFZGWNJvakJ3U1ZWc1pITIVWa1YzVW01Q1lWSjZSa2haZWtwSFYwWldkRmR0Um1saE0wRjVWMW
Q0VDFock5YSmlSVkpVvJBkb1VGBFhNWHBOUmxKMFkwWmFIRlpVmxawFdIQLBVMjFHV1ZwSE5WUldlbFpRV1RCYWRtVIZOVmhoUjNSVvVteH
ZNvII5ZUU5VGF6VnlZa1ZTVkZkSFVsQmFWM1J5VFZaU1ZsVnROVTVTVkZaV1ZWWVnVMVJ0Vm5OU2FrNVZUVVpLVDFwVlZYaFNWbTk2WWvtVMU
xSlZWWGRXUm1oM1ZtMVJkMDIwVmxWVFNFSIBWV3RrYTA1R1VYafZhmHBQVWxSV1JsZHFubXRWUmtwR1RvaHdWVlpXU25WVWZVWVXhZM
VpHVIZWck9WTINWbGw2VmtSQ1UxWnJNSGR0VIZaafXpXdfGVrRnBYZEVabGJGRjRWV3MxYTAXRVZUSlpWRTV6Vkd4S1NGULVSBfZpUmtwRFZG
UkJOVlpXYTNwYVJUBHNZbGhSTWxaR1ZsTmliVkyZVGxVmRTVlhCM1ZrVmFVMU5yTVZaTlNHeFRWa1phVUZsWE1VOU5NVkoxWTBoT2FWWWVSV
EZWTvdofUfUyeEZkMkl6YUzwbGEzQlXbFprUzJSR1NuVIZiVpXVfVkemVWVXhWazlUTURGWVZGaHNWmkp0ZUZOVMJuQkhZakZzVjFwRVRtdF
diWFEXVkrGb1UxTnRSbGxhUnpWVWZucFdSRmxWWkU1bGJGSnhWRzFzVGSxSVFubFZNV1IzWkd4dmQySkUbfZotW1oVFZGVldTMIzHYkhGVFZ
GWNJvakJ3U1ZWc1pITIVWa1YzVW01Q1lWSjZSa2haZWtwSFYwWldkRmR0Um1saE0wRjVWMWQ0VDFock5YSmlSVkpVvJBkb1VGBFhNWHBOU
mxKMFkwWmFIRlpVmxawFdIQLBVMjFHV1ZwSE5WUldlbFpRV1RCYWRtVIZOVmhoUjNSVvVteHZNvII5ZUU5VGF6VnlZa1ZTVkZkSFVsQmFWM1J
5VFZaU1ZsVnROVTVTVkZaV1ZWWVnVMVJ0Vm5OU2FrNVZUVVpLVDFwVlZYaFNWbTk2WWvtVMUxSlZWWGRXUm1oM1ZtMVJkMDIwVmxWVFN
FSIBWV3RrYTA1R1VYafZhmHBQVWxSV1JsZHFubXRWUmtwR1RvaHdWVlpXU25WVWZVWVXhZMVpHVIZWck9WTINWbGw2VmtSQ1UxWnJNSGR
0VIZaafXpXdfGVrRnBYZEVabGJGRjRWV3MxYTAXRVZUSlpWRTV6Vkd4S1NGULVSBfZpUmtwRFZGukJOVlpXYTNwYVJUBHNZbGhSTWxaR1ZsTmli
VkyZVGxVmRTVlhCM1ZrVmFVMU5yTVZaTlNHeFRWa1phVUZsWE1VOU5NVkoxWTBoT2FWWWVSVZEZWTvdofUfUyeEZkMkl6YUzwbGEzQlXbFprU
zJSR1NuVIZiVpXVfVkemVWVXhWazlUTURGWVZGaHNWmkp0ZUZOVMJuQkhZakZzVjFwRVRtdFdiWFEXVkrGb1UxTnRSbGxhUnpWVWZucFdSR
mxWWkU1bGJGSnhWRzFzVGSxSVFubFZNV1IzWkd4dmQySkUbfZotW1oVFZGVldTMIzHYkhGVFZGWNJvakJ3U1ZWc1pITIVWa1YzVW01Q1lWS
jZSa2haZWtwSFYwWldkRmR0Um1saE0wRjVWMWQ0VDFock5YSmlSVkpVvJBkb1VGBFhNWHBOUmxKMFkwWmFIRlpVmxawFdIQLBVMjFHV1Z
wSE5WUldlbFpRV1RCYWRtVIZOVmhoUjNSVvVteHZNvII5ZUU5VGF6VnlZa1ZTVkZkSFVsQmFWM1J5VFZaU1ZsVnROVTVTVkZaV1ZWWVnVMVJ0V
m5OU2FrNVZUVVpLVDFwVlZYaFNWbTk2WWvtVMUxSlZWWGRXUm1oM1ZtMVJkMDIwVmxWVFNFSIBWV3RrYTA1R1VYafZhmHBQVWxSV1JsZ
HFubXRWUmtwR1RvaHdWVlpXU25WVWZVWVXhZMVpHVIZWck9WTINWbGw2VmtSQ1UxWnJNSGR0VIZaafXpXdfGVrRnBYZEVabGJGRjRWV3MxY
TAXRVZUSlpWRTV6Vkd4S1NGULVSBfZpUmtwRFZGukJOVlpXYTNwYVJUBHNZbGhSTWxaR1ZsTmliVkyZVGx

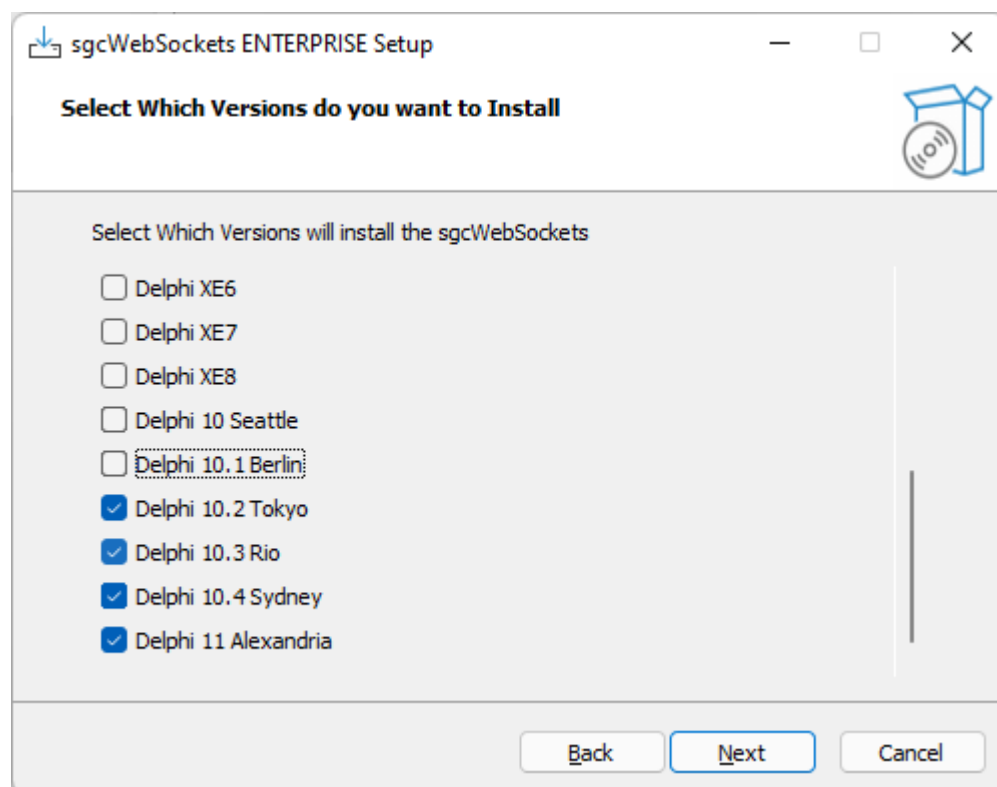
- Se a licença foi ativada com sucesso, selecione se você quer instalar na IDE do Delphi, CBuilder ou Rad Studio. Há uma opção para extrair os arquivos necessários do lazarus (o Lazarus requer a instalação manu- al do pacote).



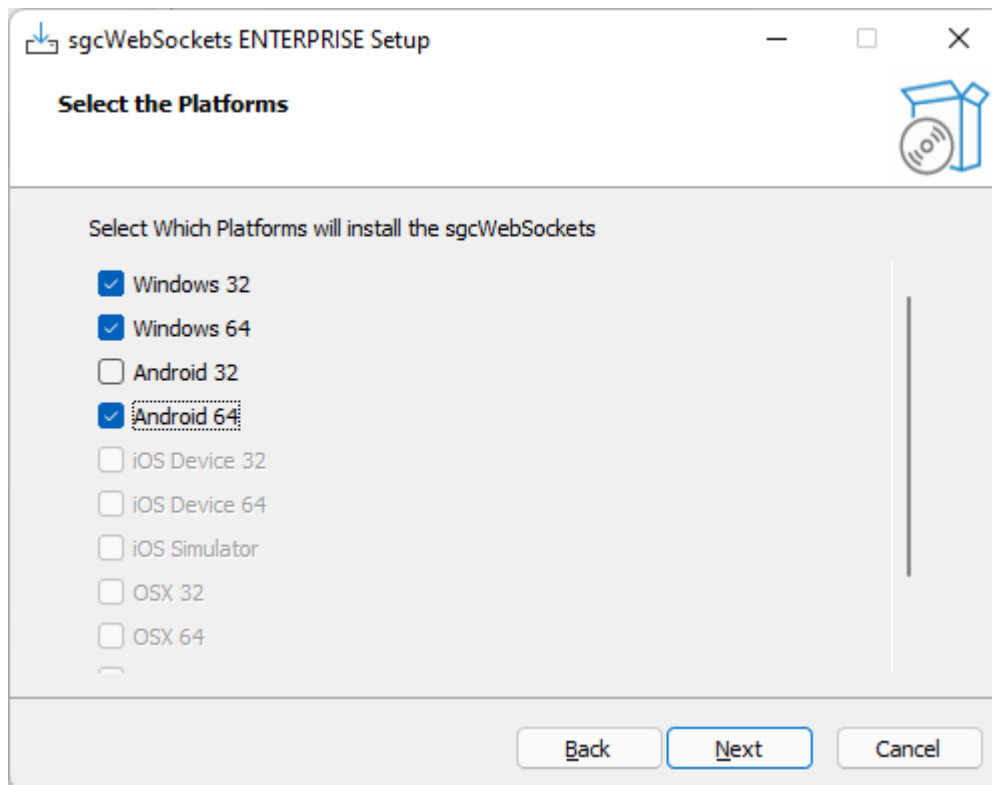
- Há algumas opções que podem ser personalizadas toda vez que você usa o instalador. Pressione o botão **Options** para acessar essas propriedades.
 - **Build Packages:** se selecionado, o instalador tentará compilar os pacotes.
 - **Register Paths IDE:** se selecionado, o instalador registrará os caminhos de bibliotecas necessários na IDE.
 - **Register BPLs IDE:** se selecionado e o instalador tiver compilado os pacotes com sucesso, o instalador registrará o pacote de tempo de design na IDE.
- As seguintes opções estão disponíveis apenas para licenças com código-fonte:
 - **Build IntraWeb:** se selecionado, o instalador instalará os arquivos IntraWeb necessários (desabilitado por padrão). Atualmente instala a versão IntraWeb XVI.
 - **Build CBuilder Dproj:** se selecionado, o instalador compilará o pacote CBuilder usando o pacote Delphi sgcWebSockets e gerando todos os arquivos CBuilder necessários.
 - **Use Custom Indy:** (apenas Enterprise), se selecionado, o sgcWebSockets usará a Custom Indy Version (com suporte a openssl 1.1 e 3.0, TLS 1.3, ALPN...)
 - **sgcIndy Already Installed:** se o pacote sgcIndy foi instalado e você deseja usar este pacote para compilar o pacote sgcWebSockets, marque esta opção.
 - **sgcIndy Compatibility Mode:** se o pacote sgcIndy foi instalado em Compatibility Mode (porque outros pacotes estão usando Indy, como o DevExpress), marque esta opção.
 - **Always use of the following OpenSSL API Versions:** marque esta opção se você deseja forçar o uso das APIs do OpenSSL 1.1.1 ou OpenSSL 3.0.0
 - **Debug Mode:** salva mensagens de depuração em um arquivo de log. Não use este modo em um ambiente de produção.
 - **Build Rad Studio IDE Win64:** compila o pacote de design-time para o Rad Studio IDE 12+ 64 bits.
 - **Include Resources:** habilitado por padrão. Embute o bundle do cliente JavaScript (`sgcResources.RES`) nos pacotes de runtime. O recurso embutido só é necessário se o seu servidor entregar o cliente para navegador do sgcWebSockets via HTTP a navegadores remotos. Desmarque esta opção para retirar a definição de `SGC_RESOURCES` em `sgcVer.inc` antes de os pacotes serem compilados, reduzindo o tamanho de cada executável que faz link com o código-fonte. Seguro para desabilitar em aplicações Delphi / C++Builder somente cliente e em servidores que não servem o cliente JS embutido. Veja [Configure Install](#) para a diretiva de compilador subjacente.



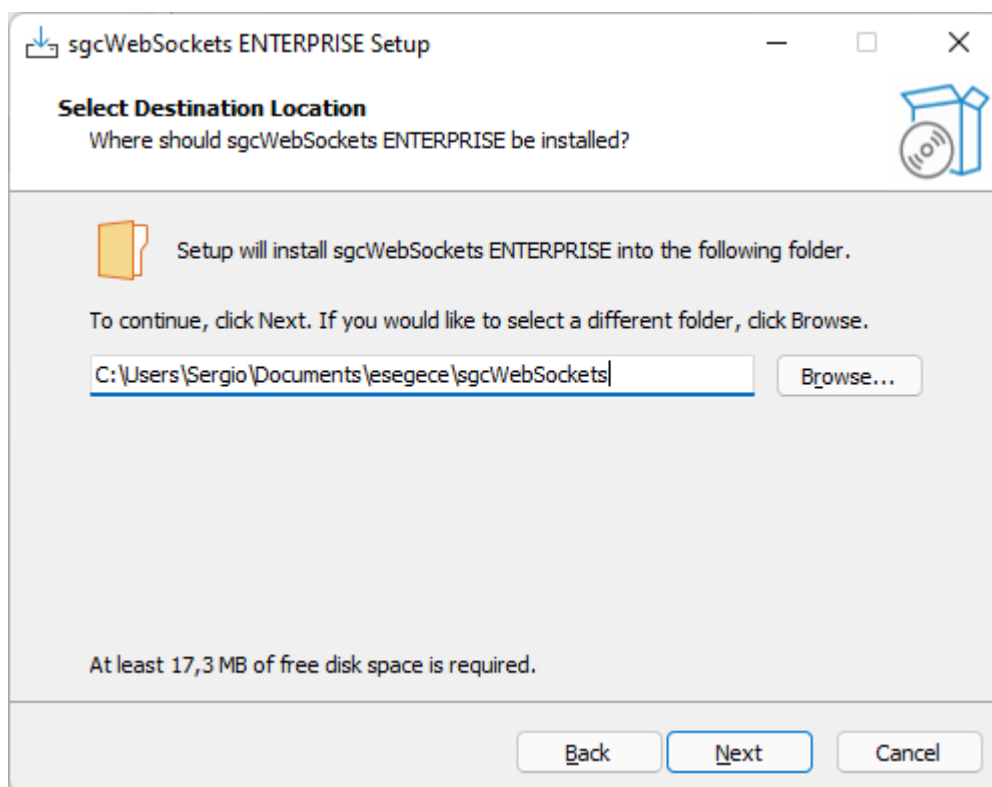
- Agora você pode selecionar quais versões do IDE deseja instalar. Apenas as versões do IDE que o instalador detectar como instaladas estarão disponíveis.



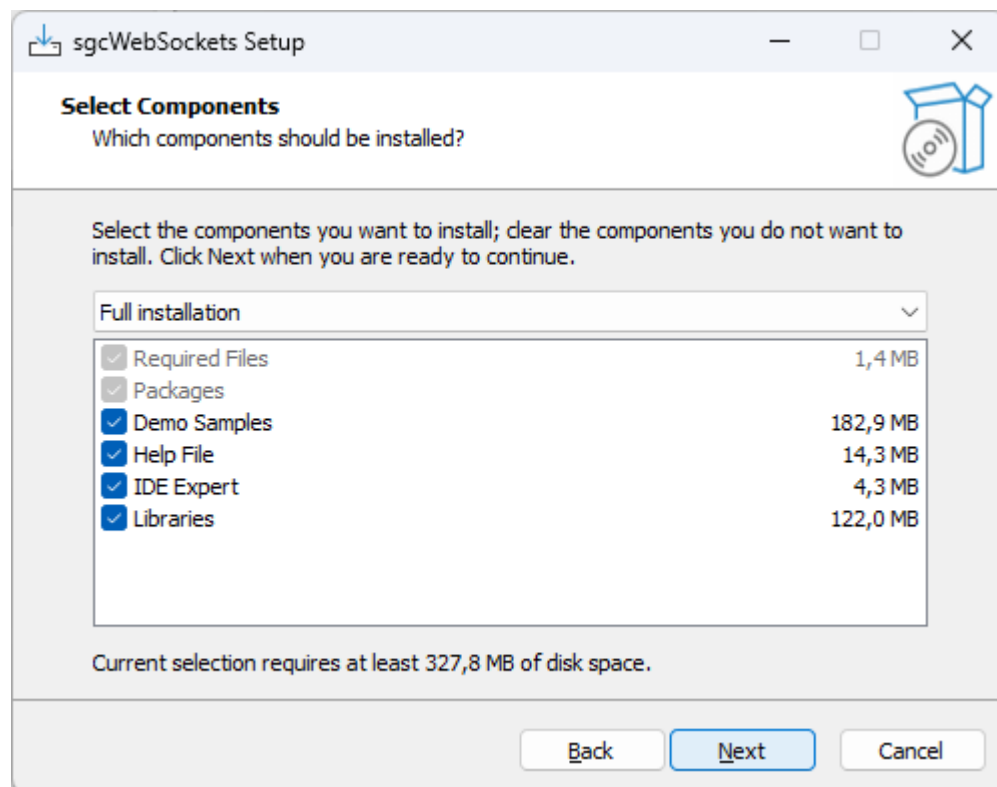
- O próximo passo é selecionar as plataformas.



- Selecione a pasta onde o pacote será instalado. Se você reinstalar o pacote, o instalador selecionará por padrão a mesma pasta selecionada na instalação anterior.



- Selecione quais componentes instalar. Os clientes registrados têm um **IDE expert** que permite conectar à conta eSeGeCe a partir da IDE, saber se há atualizações disponíveis, acesso direto ao helpdesk... e muito mais.



- Por fim, ele extrairá os arquivos, compilará e instalará o pacote e registrará os caminhos necessários na IDE.

Install Errors

- O MsBuild gera um erro se o comprimento do **Library Path for muito longo**. Para corrigir esse problema, tente excluir os caminhos não utilizados do library path. O MsBuild tem uma limitação de 32K caracteres.
- erro **F2110: 'tchar.h' file not found**. Este erro pode aparecer ao compilar para o CBuilder; para corrigi-lo, basta abrir o CBuilder, fechá-lo e tentar instalar novamente.

Parâmetros de linha de comando de instalação

Os seguintes comandos são suportados pelo instalador.

/SILENT

O assistente e a janela de fundo não são exibidos, mas o progresso da instalação é

/VERYSILENT

Quando um setup é muito silencioso, esta janela de progresso da instalação não é exibida.

/EXTRACT

O pacote não é instalado, apenas extraído. O caminho onde ele é instalado pode ser personalizado utilizando /EXTRACT=path-to-folder

Utilize este parâmetro e /SILENT se você quiser apenas extrair os arquivos sem interação do usuário.

/IDE

Este parâmetro permite que você defina qual IDE deseja instalar. Defina um dos seguintes:

- delphi
- cbuilder
- radstudio

Adicionalmente, você pode adicionar o Lazarus.

Exemplo: instalar delphi e lazarus.

/ide=delphi-lazarus.

/VERSIONS

Usando este parâmetro, você pode definir quais versões do RAD Studio deseja instalar. Múltiplas opções são permitidas:

- D7
- D2007
- D2009
- D2010
- DXE
- DXE2
- DXE3
- DXE4
- DXE5
- DXE6
- DXE7
- DXE8
- D10
- D10_1
- D10_2
- D10_3
- D10_4
- D11
- D12
- D13

Use o valor "All" para instalar todas as versões possíveis.

Exemplo: instale o Delphi 10 e o Delphi 13.

/versions=D10-D13

/PLATFORMS

Utilizando este parâmetro, você pode definir quais plataformas do RAD Studio deseja instalar. Múltiplas opções são permitidas:

- Win32
- Win64
- Win64x
- Android
- Android64
- iOSDevice32
- iOSDevice64
- iOSSimulator
- iOSSimARM64
- OSX32
- OSX64
- OSXARM64
- Linux64

Use o valor "All" para instalar todas as plataformas possíveis.

Exemplo: instalar Win32 e Win64.

/platforms=Win32-Win64

/USERNAME

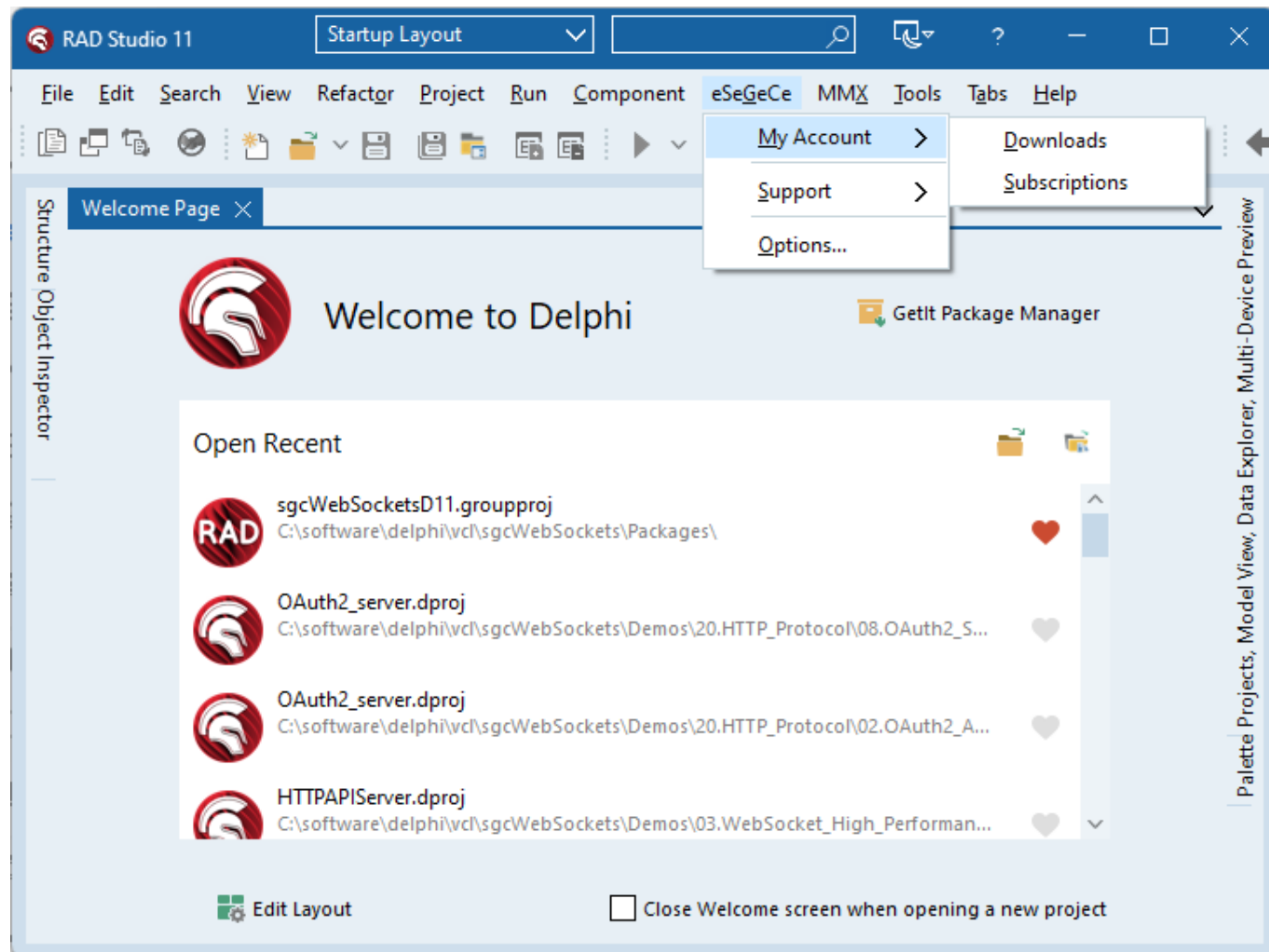
Define o nome de usuário da assinatura

/PASSWORD

Define a senha da assinatura

IDE Expert

Se o IDE Expert estiver instalado, você encontrará as seguintes opções de menu:

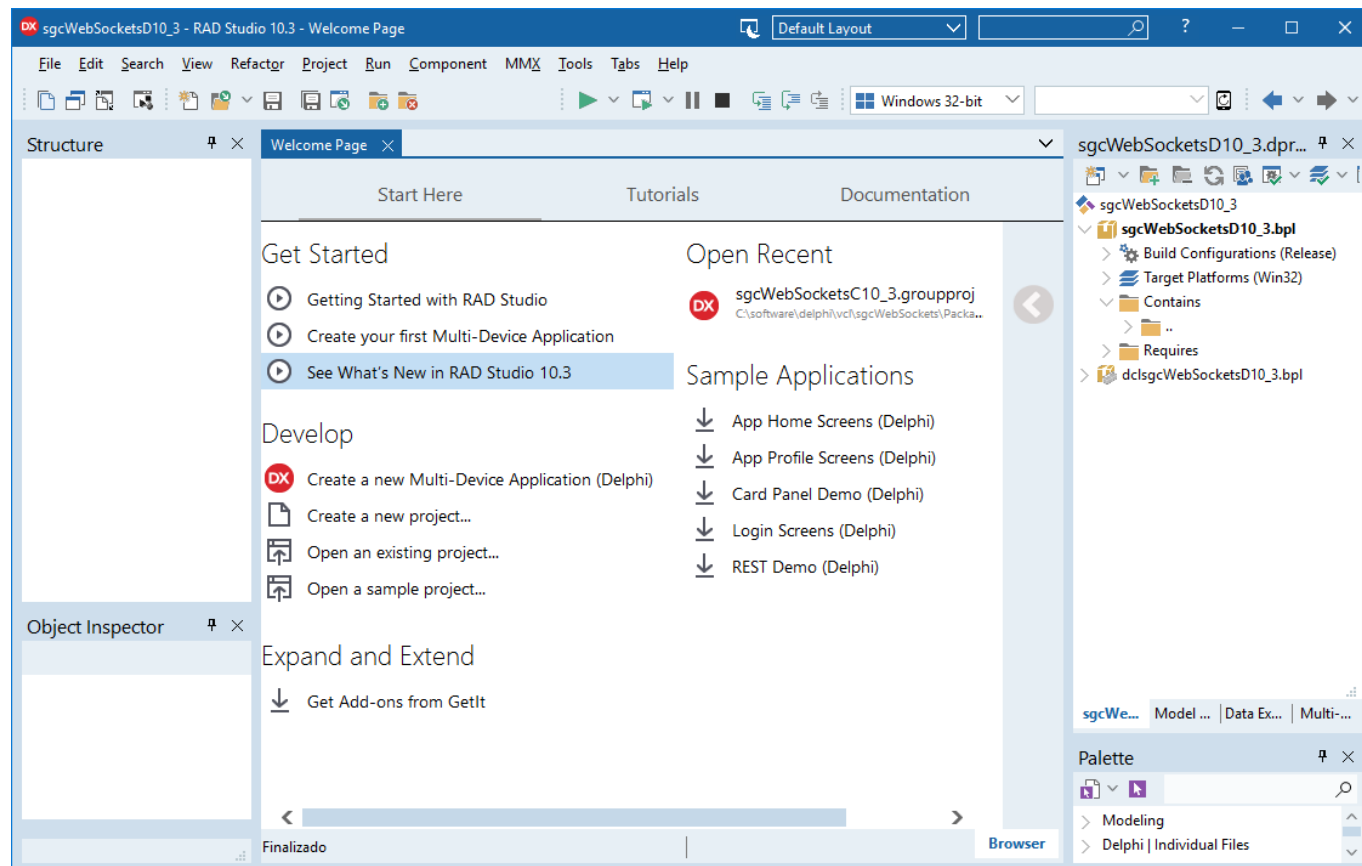


- **MyAccount:** acesso direto ao menu Downloads (onde você pode baixar a versão mais recente da beta) e às Assinaturas, para gerenciar sua licença ou renovar uma licença expirada.
- **Support:** acesso direto ao HelpDesk ou Forum com login automático. Documentação e formulário de Contato também estão disponíveis.
- **Options:** neste menu você pode configurar o nome de usuário/senha da sua conta. Selecione o navegador padrão e verifique se há atualizações disponíveis.

Instalar o Pacote Manualmente

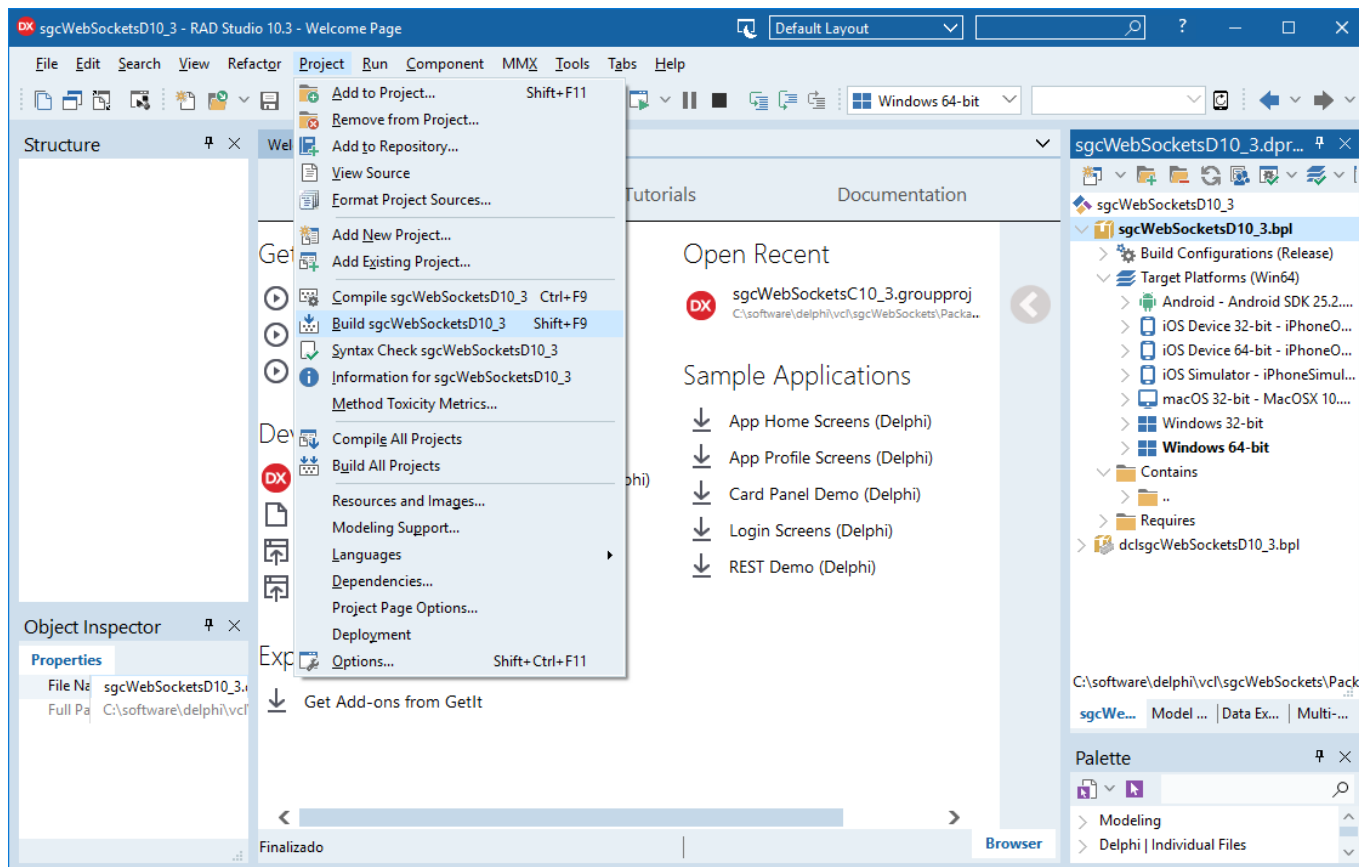
Siga os próximos passos para instalar o pacote sgcWebSockets, as capturas de tela usam a versão Delphi 10.3.

1. Abra o group project sgcWebSocketsD10_3.

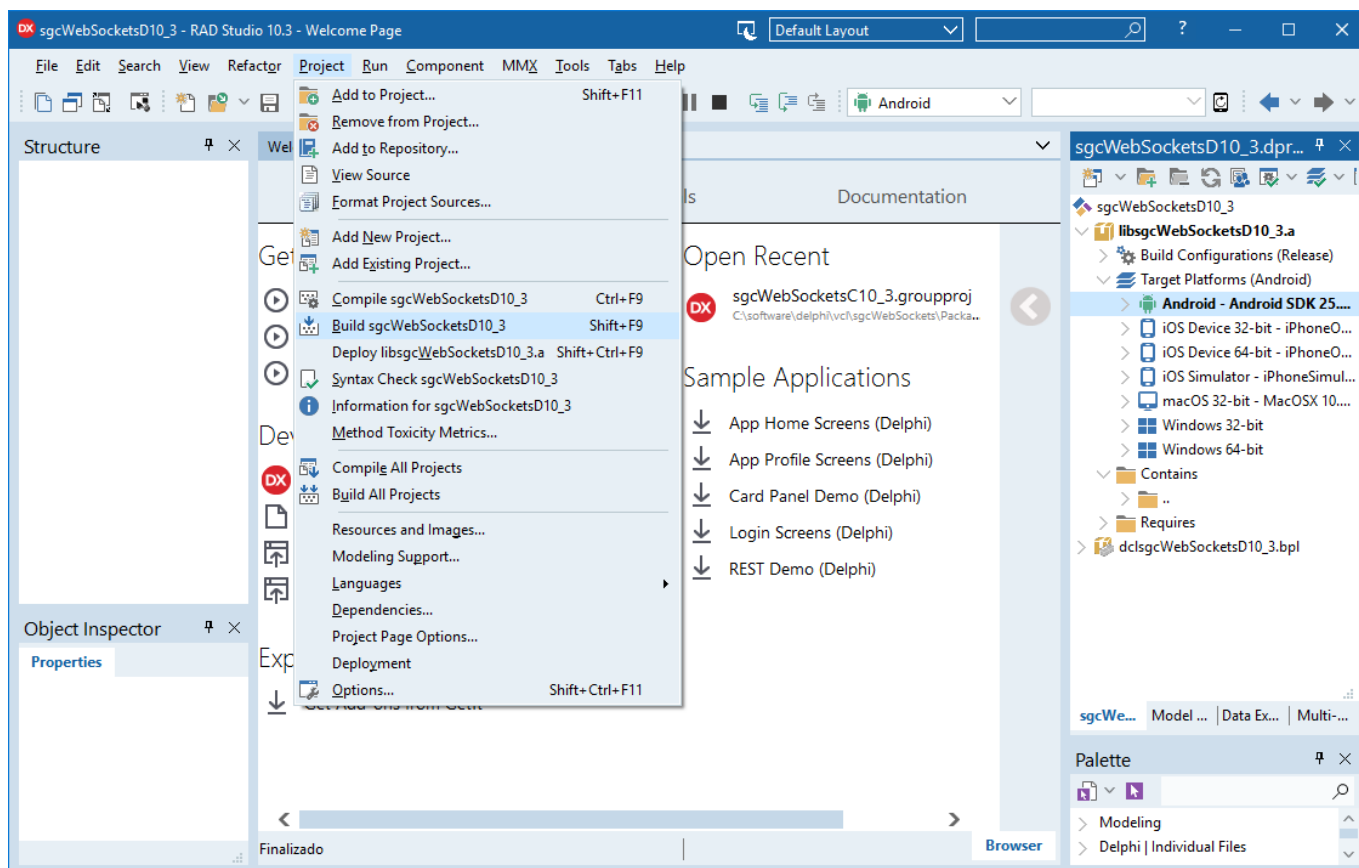


2. Agora devemos **compilar primeiro os pacotes de runtime** (o nome começa com sgcWebSockets). Há um pacote para cada plataforma de destino e isso depende da versão do Delphi, portanto **selecione a plataforma de destino uma a uma e compile** cada pacote.

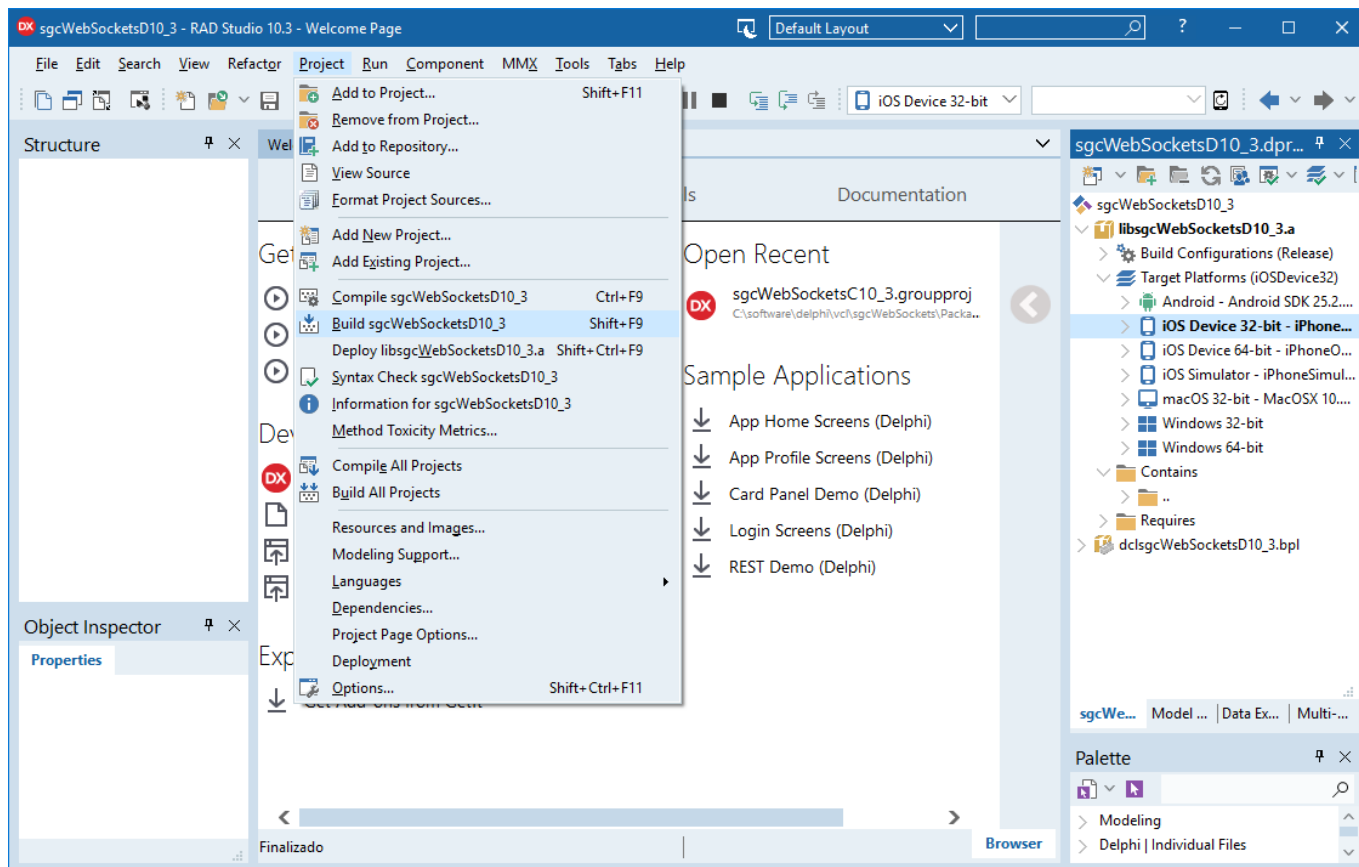
3. **Selecione win64** como plataforma de destino e compile o pacote.



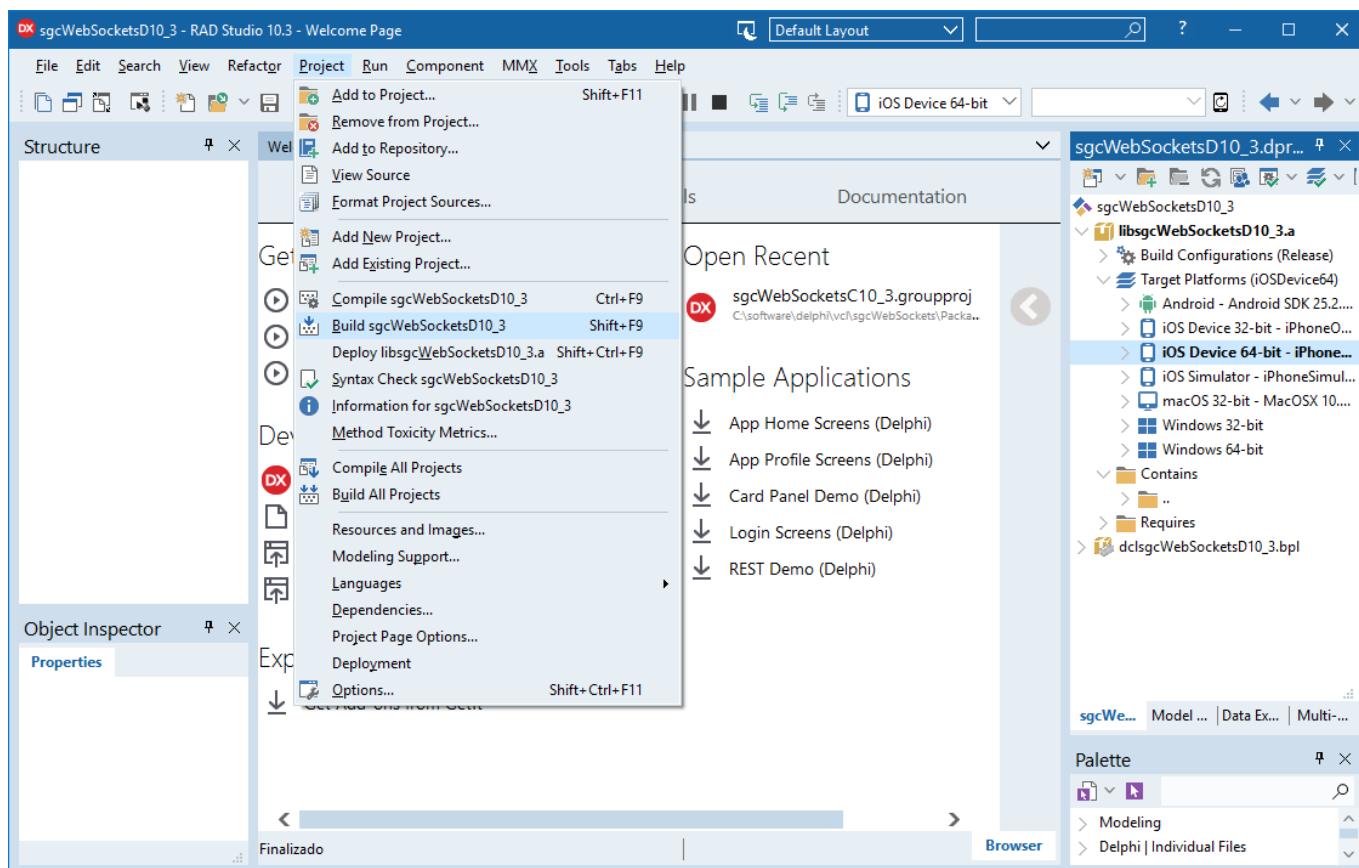
4. Selecione Android como Plataforma de Destino e compile o pacote.



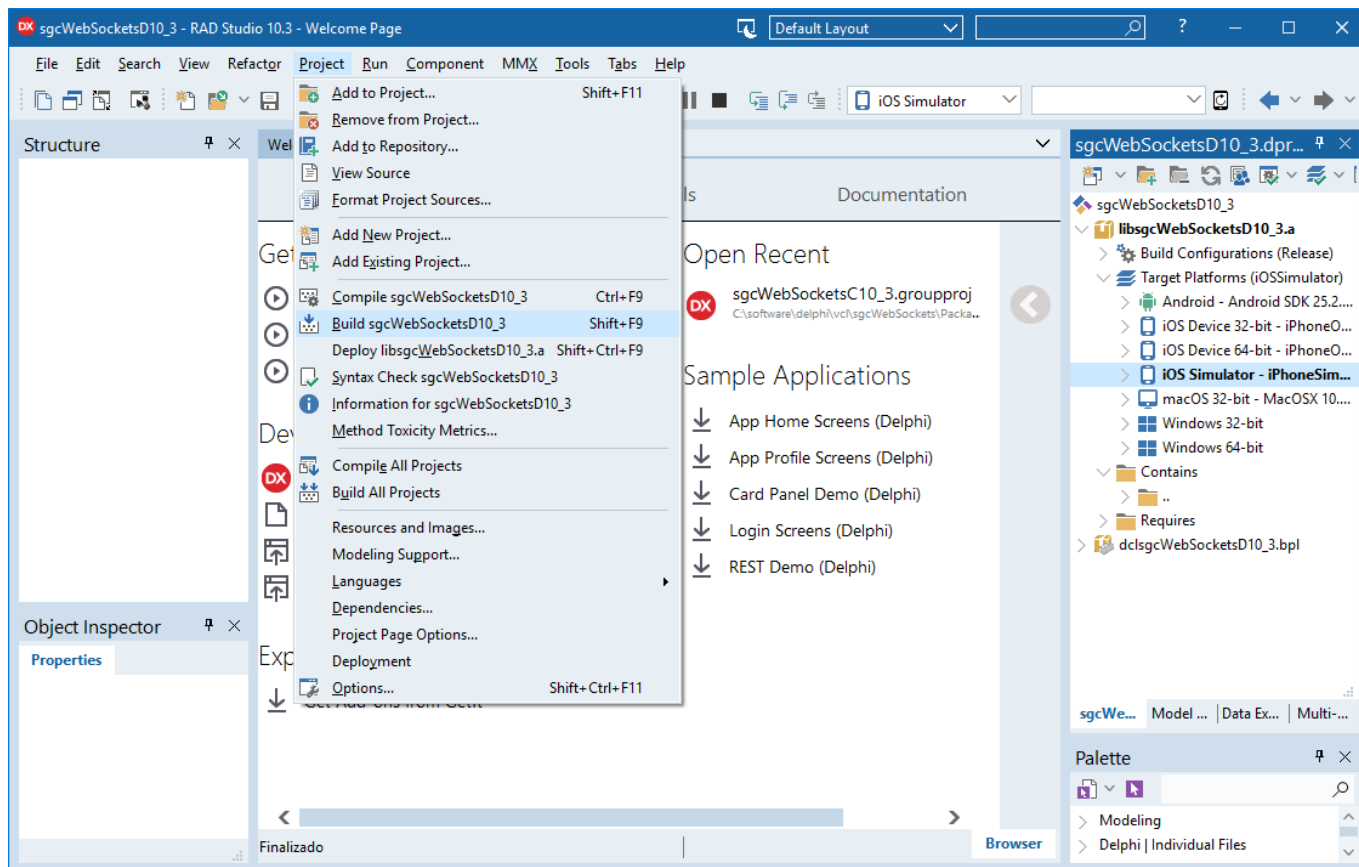
5. Selecione iOS Device 32 como Target Platform e compile o pacote.



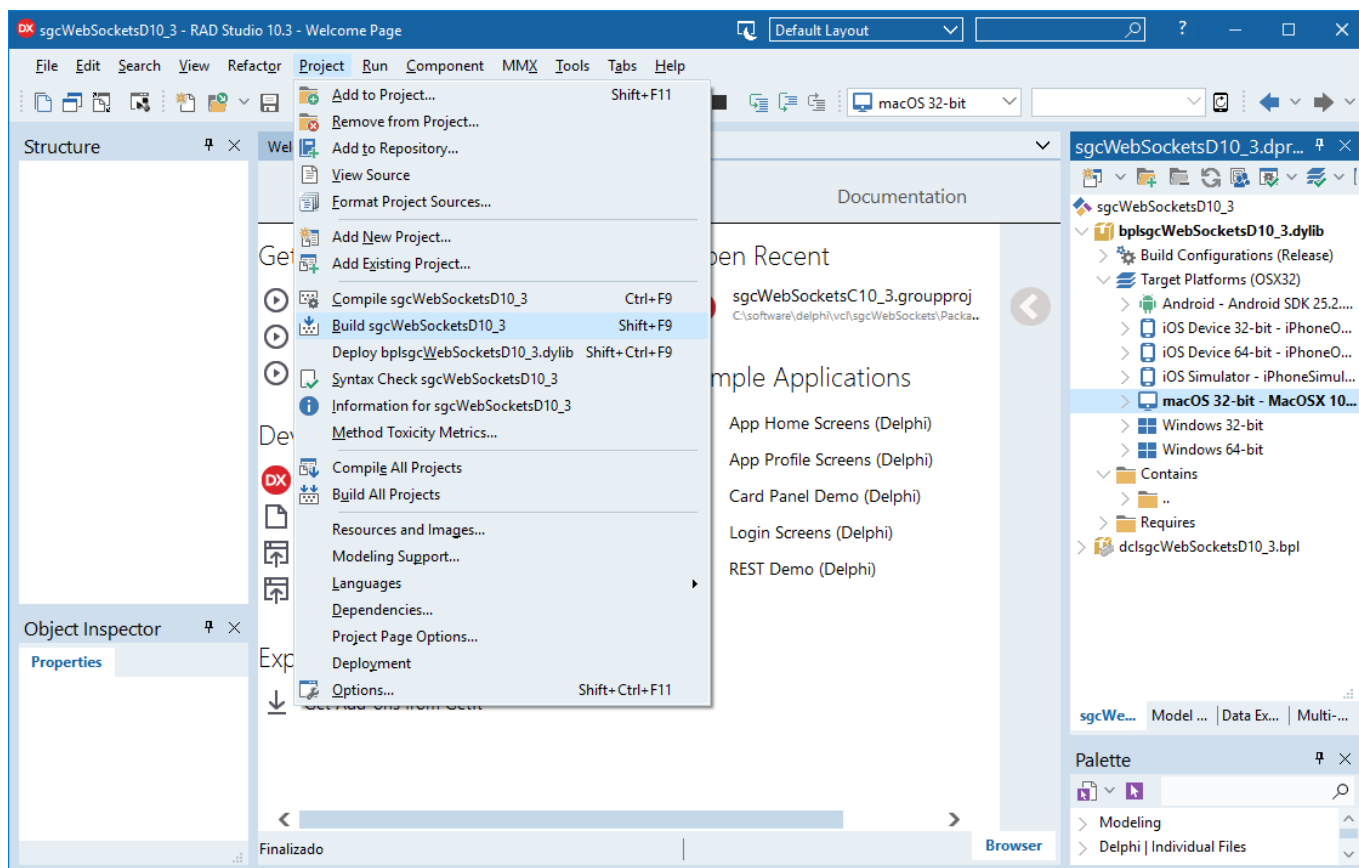
6. Selecione **iOS Device 64** como Plataforma de Destino e compile o pacote.



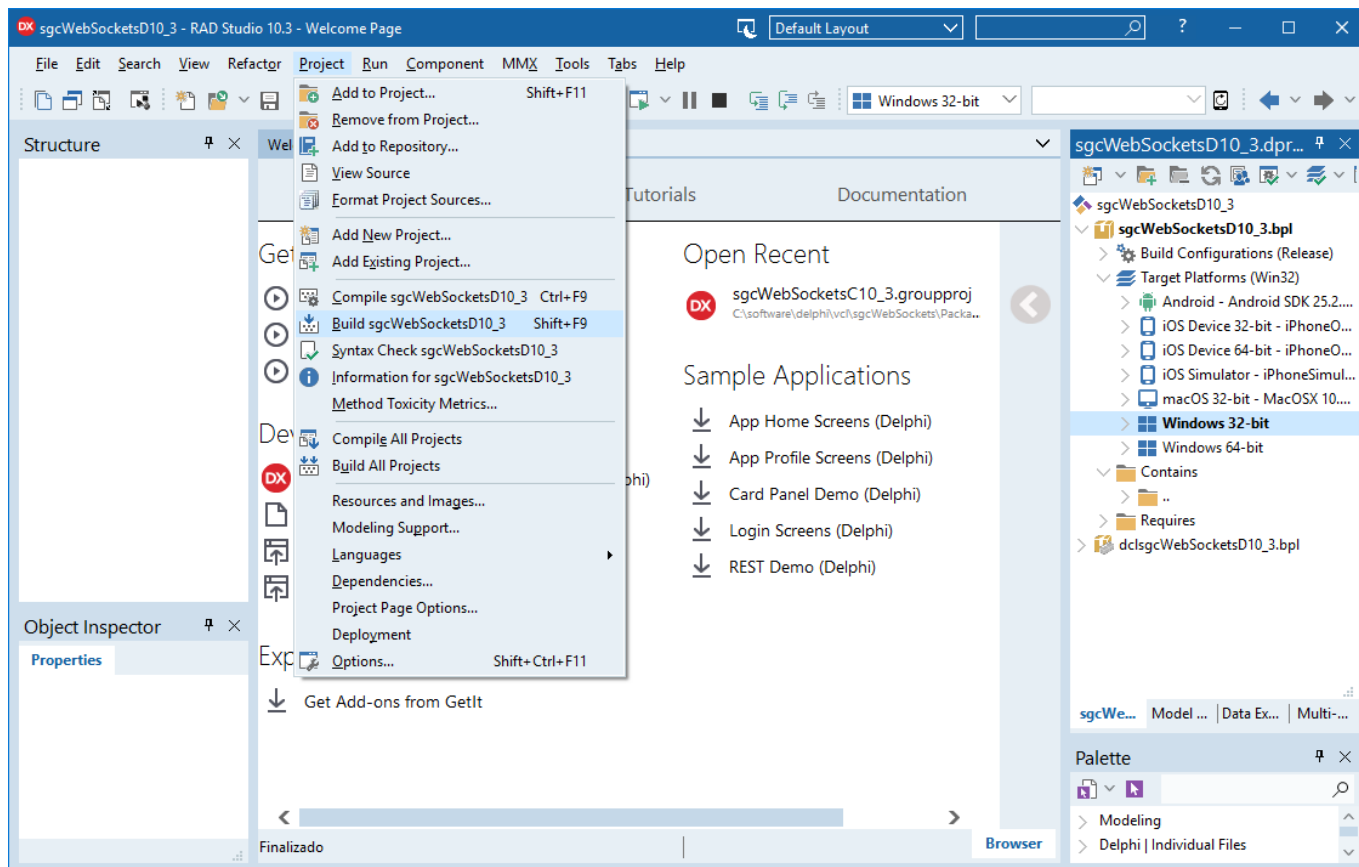
7. Selecione o **iOS Device Simulator** como Target Platform e compile o pacote.



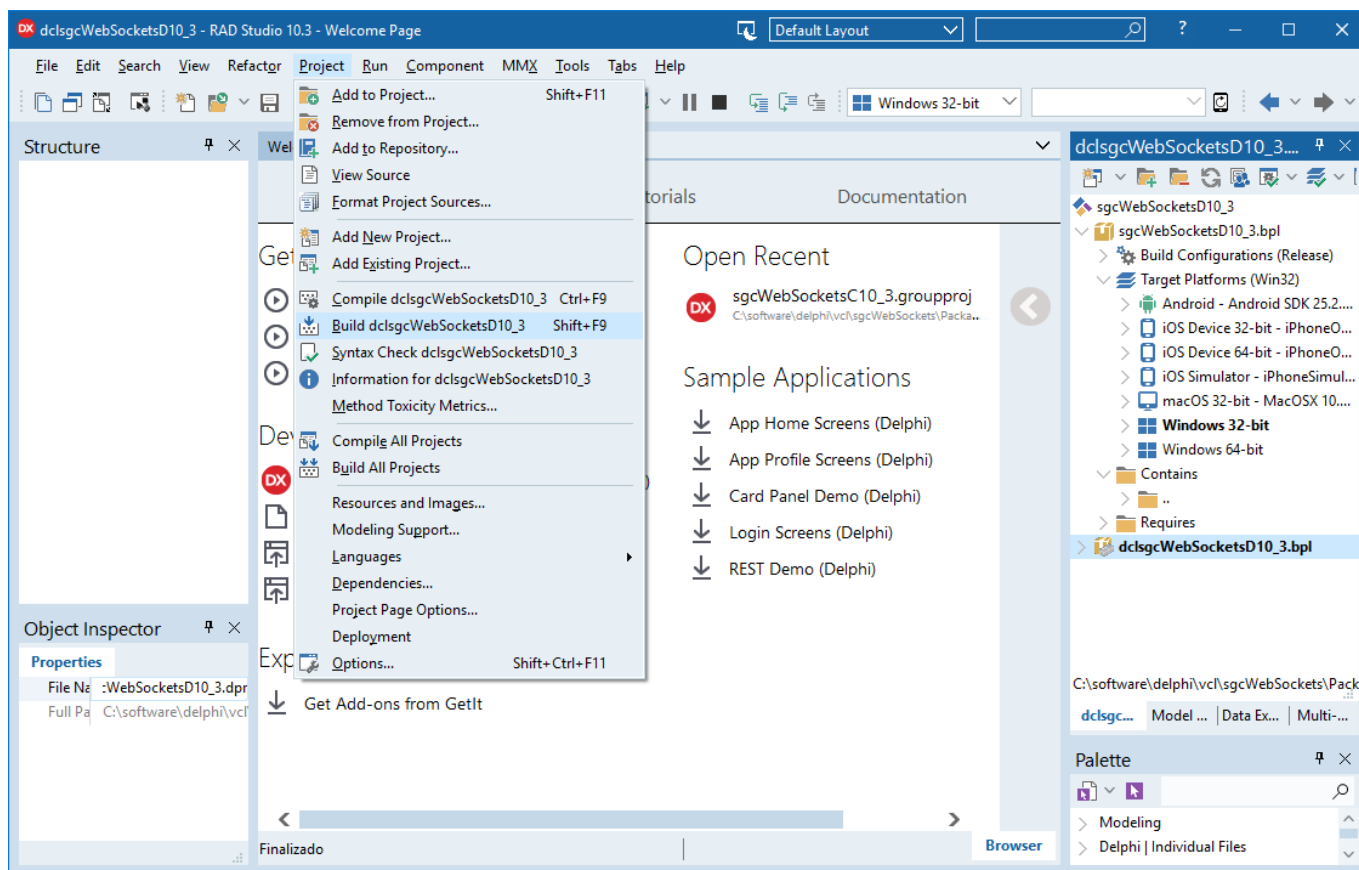
8. Selecione **MacOS 32** como Target Platform e compile o pacote.

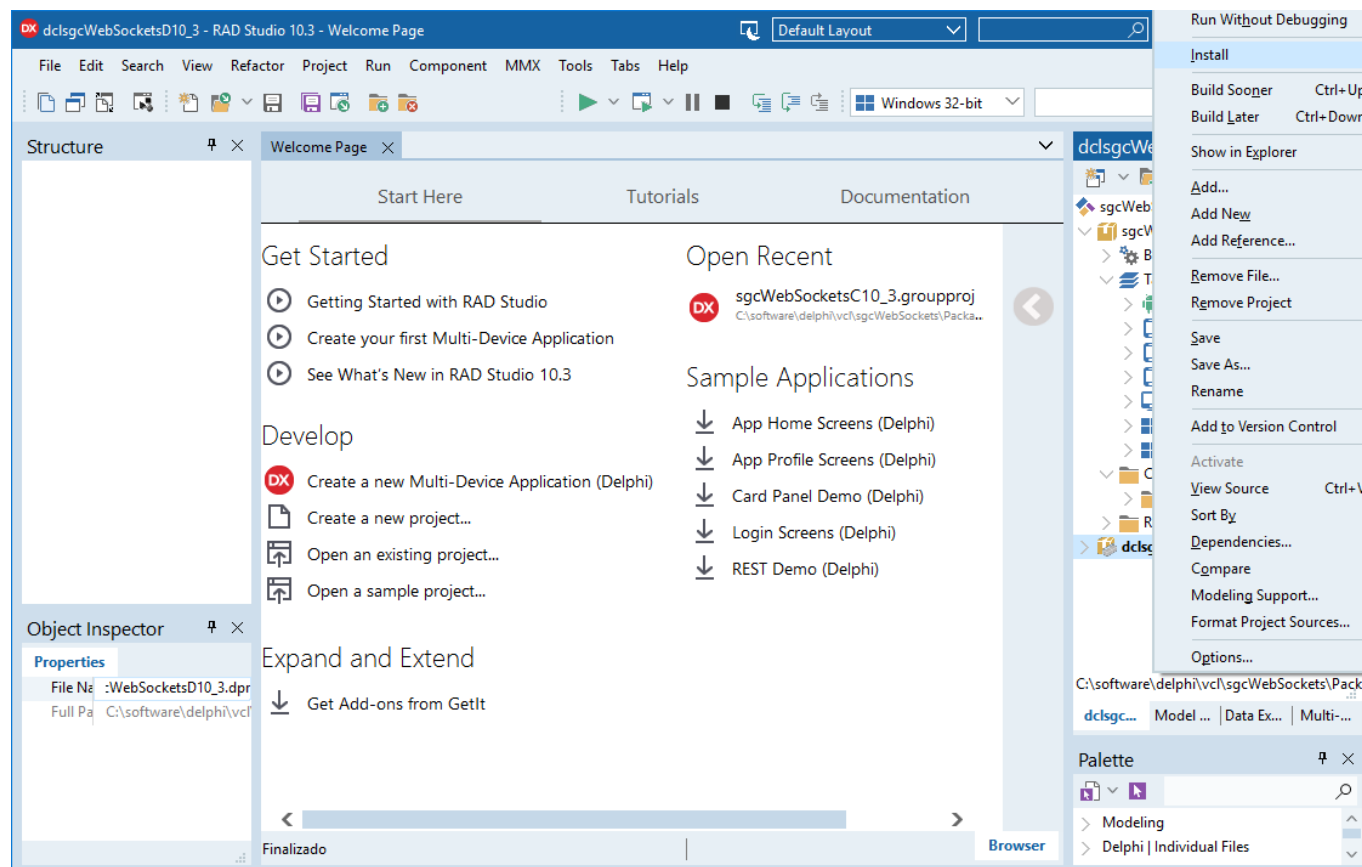


9. Selecione **Win32** como Target Platform e compile o pacote.

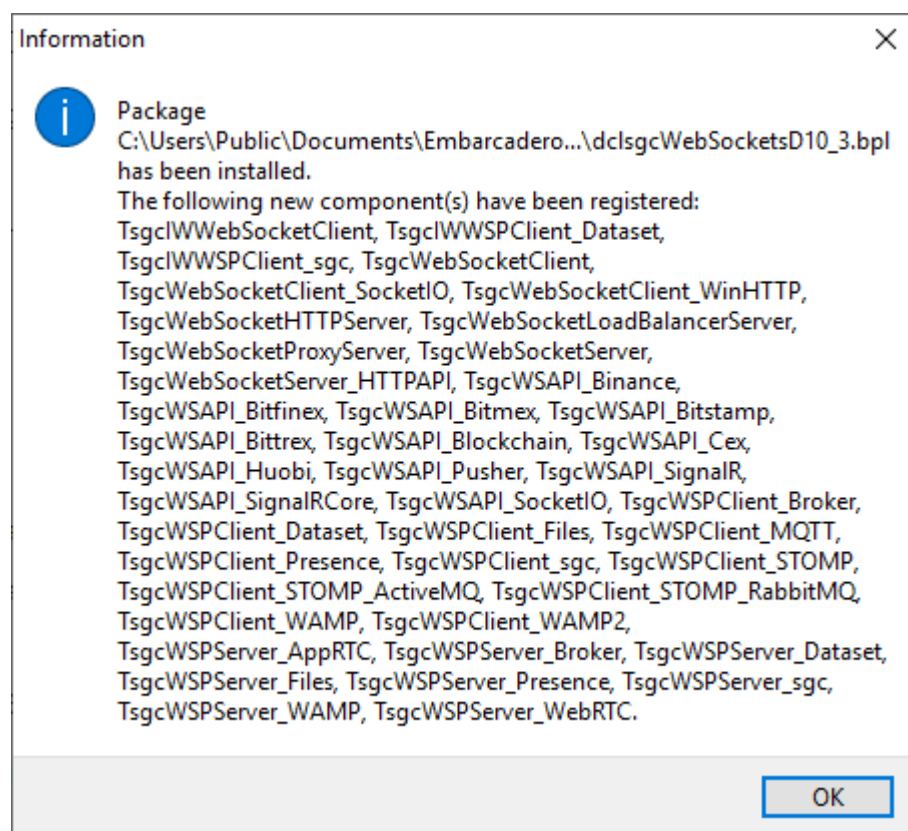


10. Uma vez que todos os runtime packages estejam compilados, selecione o **design time package** (o nome começa com dcl) e primeiro faça o **build** e depois o **install** (os design time packages têm apenas Win32 como plataforma de destino).

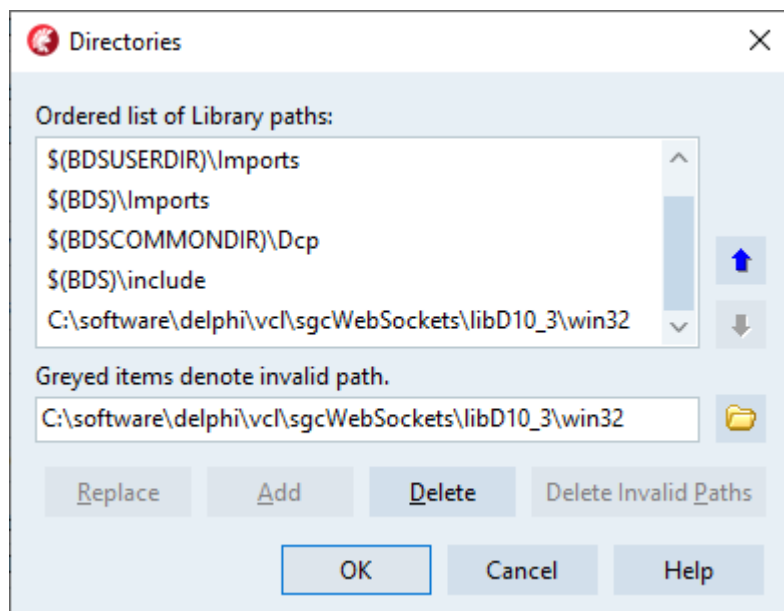




11. Se a instalação for **bem-sucedida**, você verá uma mensagem com todos os componentes instalados.



12. Então, você só precisa adicionar o diretório onde os arquivos compilados estão localizados ao seu **Rad Studio Library Path**. Você deve adicionar isso para cada Target Platform (win32, win64, osx64...)



* Se você estiver usando os servidores **Datasnap**, eles **NÃO estão incluídos** no pacote sgcWebSockets porque não podem ser instalados; são componentes somente de runtime. Neste caso, você deve adicionar também a pasta Source ao seu caminho de bibliotecas.

Install Errors

Às vezes, você pode obter alguns erros ao instalar os componentes.

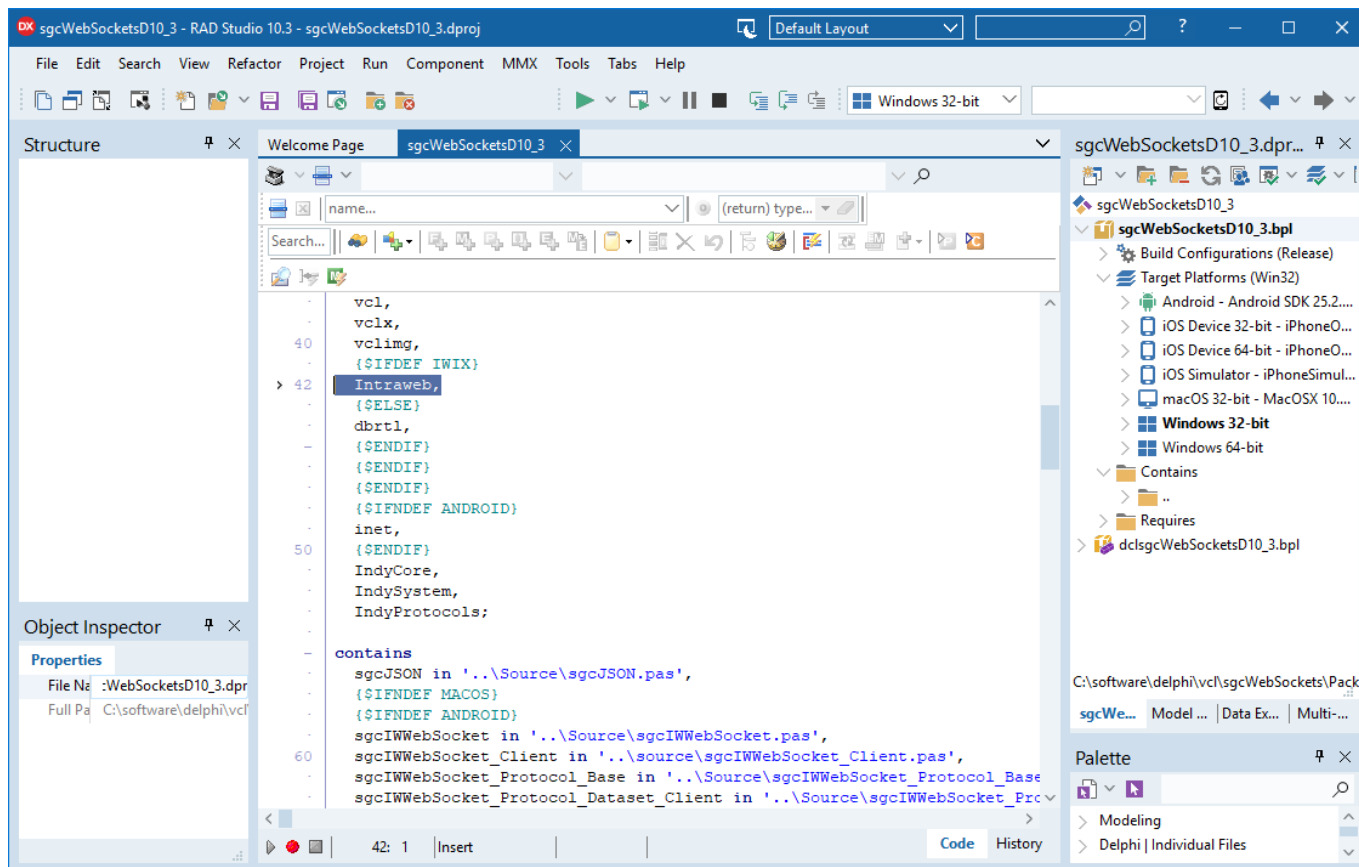
Intraweb package not found

O sgcWebSockets é compilado utilizando a **versão padrão do Intraweb** fornecida com o Delphi. Se você não tiver o Intraweb instalado, pode **modificar o arquivo sgcVer.inc** (localizado na pasta Source). Pesquise a sua versão do Delphi e **comente todos os compiler defines do Intraweb** (começam com IW). **Exemplo:** para o Delphi 10.4, comente todos os compiler defines do Intraweb

```
{$IFDEF VER340} { Delphi 10.4 }
{$DEFINE D2006}
{$DEFINE D2007}
{$DEFINE D2009}
{$DEFINE D2010}
{$DEFINE DXE}
{$DEFINE DXE2}
{$DEFINE DXE3}
{$DEFINE DXE4}
{$DEFINE DXE5}
{$DEFINE DXE6}
{$DEFINE DXE7}
{$DEFINE DXE8}
{$DEFINE D10}
{$DEFINE D10_1}
{$DEFINE D10_2}
{$DEFINE D10_3}
{$DEFINE D10_4}
{$DEFINE INDY10_1}
{$DEFINE INDY10_2}
{$DEFINE INDY10_5_5}
{$DEFINE INDY10_5_7}
{$DEFINE INDY10_5_8}
{$DEFINE INDY10_6}
{$DEFINE INDY10_6_0_5122}
{$DEFINE INDY10_6_0_5169}
{$DEFINE INDY10_6_2_5263}
{$DEFINE INDY10_6_2_5366}
{$DEFINE INDY10_6_2_D10_4}

{$IFNDEF BCB}
{$IFNDEF MACOS}
{$IFNDEF ANDROID}
{.$DEFINE IWIX}
{.$DEFINE IWXI}
{.$DEFINE IWXIV}
{.$DEFINE IWXV}
{$ENDIF}
{$ENDIF}
{$IFNDEF NEXTGEN}
{$DEFINE SGC_JSON_INTF}
{$ENDIF}
{$ENDIF}
{$ENDIF}
```

Se o Intraweb estiver instalado, mas for uma versão diferente da padrão que vem com o Delphi, talvez o seu pacote Intraweb tenha um nome diferente. Então abra o pacote runtime do sgcWebSockets e **altere o nome do Intraweb** no código-fonte do projeto.



Indy Package not found

O **sgcWebSockets** requer **Indy** para instalar componentes na sua IDE. A instalação Trial é compilada contra a biblioteca Indy fornecida com o Delphi / CBuilder, então, se você receber uma mensagem como esta:

[DCC Fatal Error] dclsgcWebSocketsDX.dpk(31): E2202 Required package 'IndyCore' not found

Muito provavelmente, você tem uma **versão mais recente do Indy**, portanto, para instalar a trial, você deve excluir esta versão e instalar a versão integrada do indy utilizando o setup do Delphi / CBuilder.

Se você tem o código-fonte completo, então basta verificar:

1. Pacotes Indy necessários: IndyCore, IndySystem e IndyProtocols. Se você tiver uma versão mais nova do Indy, os pacotes provavelmente têm um nome diferente (incluindo a versão), portanto acesse o menu **"Component / Install Packages"** e verifique qual nome têm os pacotes Indy e altere de acordo no projeto.

2. sgcWebSockets suporta várias versões do Indy, há defines de compilador para permitir a compilação para cada versão do Indy. Abra **sgcVer.inc**, localizado na pasta source, e altere de acordo com a sua versão do Indy (é o gslidVersion do arquivo Indy IdVers.inc). Alguns defines de compilador:

```

{$DEFINE INDY10_1}
{$DEFINE INDY10_2}
{$DEFINE INDY10_5_5}
{$DEFINE INDY10_5_7}
{$DEFINE INDY10_5_8}
{$DEFINE INDY10_6}
{$DEFINE INDY10_6_0_5122}
  {$DEFINE INDY10_6_0_5169}
{$DEFINE INDY10_6_2_5263}
{$DEFINE INDY10_6_2_5366}
{$DEFINE INDY10_6_2_D10_4}
    
```


c00000005 ACCESS_VIOLATION no CBuilder

Se você compilar um projeto usando o CBuilder e obtiver este erro, defina as seguintes opções no seu projeto:

Project > Options > C++ Linker
desmarque "Link with Dynamic RTL"

Project > Options > Packages > Runtime Packages
desmarque "Link with runtime packages"

Unable to find package import: sgcWebSocketsCXXX.bpi no CBuilder Win64

Ao compilar o runtime package para win64, você deve compilar Release e Debug.

Referência ambígua System.ZLib.hpp e IdZLib.hpp CBuilder

O sgcWebSockets Standard e Professional utiliza o Indy para alguns componentes, e o Indy não faz uso da unit ZLib, utiliza sua própria cópia de ZLib: IdZLib, IdZLibHeaders... o projeto está vinculando à ZLib e às units ZLib do indy, portanto, ao compilar, o compilador não sabe qual é a referência correta porque os nomes são os mesmos. Há 2 soluções:

1. Procure onde está incluído um link para System.ZLib.hpp e exclua-o ou mova-o para depois de IdZLibHeaders.hpp
2. Utilize os seguintes defines condicionais NO_USING_NAMESPACE_SYSTEM_ZLIB ou DELPHIHEADER_NO_IMPLICIT_NAMESPACE_USE nas opções dos seus projetos para evitar o uso de System.Zlib.hpp

Referência ambígua System.ZLib.hpp e sgclDZLib.hpp CBuilder

O sgcWebSockets Enterprise usa uma versão Indy personalizada para alguns componentes e o Indy não faz uso da unit ZLib, usa sua própria cópia de ZLib: sgclDZLib, sgclDZLibHeaders... o projeto está fazendo link com as units ZLib e indy ZLib, então, ao compilar, o compilador não sabe qual é a referência correta porque os nomes são os mesmos. Há 2 soluções:

1. Procure onde está incluído um link para System.ZLib.hpp e exclua-o ou mova-o para depois de sgclDZLibHeaders.hpp
2. Utilize os seguintes defines condicionais NO_USING_NAMESPACE_SYSTEM_ZLIB ou DELPHIHEADER_NO_IMPLICIT_NAMESPACE_USE nas opções dos seus projetos para evitar o uso de System.Zlib.hpp

Undefined reference to vTable for Sgcwebsocket... no CBuilder e Android

Use a seguinte solução alternativa para corrigir o erro. Adicione o arquivo libsgcwebsocketsC*.a, que está localizado na pasta padrão dcp/android, ao seu projeto usando o menu "Project/ Add to Project".

Exemplo: para o CBuilder 11, adicione ao seu projeto o arquivo "libsgcWebSocketsC11.a" que está localizado por padrão na pasta "C:\Users\Public\Documents\Embarcadero\Studio\22.0\DCP\Android\Release".

Checksum alterado no Lazarus

Este erro pode ocorrer ao tentar instalar os componentes no Lazarus se o profile para compilar a IDE não for "Optimized IDE". A versão trial é compilada com o profile "Optimized IDE".

Cannot find X used by Y, incompatible ppu

Tente a seguinte solução alternativa "Run / Clean up and rebuild" na opção de menu.

Configure Install

Na pasta source, há um arquivo chamado sgcVer.inc que inclui todos os defines de compilador para todas as IDEs Delphi, CBuilder e Lazarus.

Aqui você pode personalizar sua configuração para IntraWeb, Indy... **geralmente não há necessidade de fazer alterações**, a menos que você queira habilitar/desabilitar alguns recursos.

Altere com cuidado os defines de compilador e entre em contato conosco se precisar de assistência.

Para cada versão do Delphi, há uma seção onde você pode configurar todos os defines do compilador, um exemplo para Delphi 10.4

```
{$IFDEF VER340} { Delphi 10.4 }
{$DEFINE D2006}
{$DEFINE D2007}
{$DEFINE D2009}
{$DEFINE D2010}
{$DEFINE DXE}
{$DEFINE DXE2}
{$DEFINE DXE3}
{$DEFINE DXE4}
{$DEFINE DXE5}
{$DEFINE DXE6}
{$DEFINE DXE7}
{$DEFINE DXE8}
{$DEFINE D10}
{$DEFINE D10_1}
{$DEFINE D10_2}
{$DEFINE D10_3}
{$DEFINE D10_4}
{$DEFINE INDY10_1}
{$DEFINE INDY10_2}
{$DEFINE INDY10_5_5}
{$DEFINE INDY10_5_7}
{$DEFINE INDY10_5_8}
{$DEFINE INDY10_5_9}
{$DEFINE INDY10_6}
{$DEFINE INDY10_6_0_5122}
{$DEFINE INDY10_6_0_5169}
{$DEFINE INDY10_6_2_5263}
{$DEFINE INDY10_6_2_5366}
{$DEFINE INDY10_6_2_D10_4}

{$IFNDEF BCB}
{$IFNDEF MACOS}
{$IFNDEF ANDROID}
{.$DEFINE IWIX}
{.$DEFINE IWXI}
{.$DEFINE IWXIV}
{.$DEFINE IWXV}
{$ENDIF}
{$ENDIF}
{$IFNDEF NEXTGEN}
{$DEFINE SGC_JSON_INTF}
{$ENDIF}
{$ENDIF}
{$ENDIF}
```

Indy

Há alguns defines de compilador para a biblioteca Indy. Isto depende da versão do Indy instalada, por padrão é configurado para o pacote Indy incluído com o Delphi. A versão do Indy é o parâmetro `gsldVersion` do arquivo `IndyIdVers.inc`.

Intraweb

Se o Intraweb não estiver instalado, basta comentar os defines de compilador do Intraweb (aqueles que começam com `IW...`).

SGC_RESOURCES

Por padrão, o `sgcVer.inc` declara `{ $DEFINE SGC_RESOURCES }`, o que faz com que os pacotes de runtime embarquem o arquivo de recursos `sgcResources.RES`. Isto adiciona bytes extras a cada executável que faz link estático contra o código-fonte do `sgcWebSockets`.

O recurso contém o código de cliente JavaScript empacotado que um servidor `sgcWebSockets` pode entregar por HTTP a um navegador remoto (para que o navegador possa falar o subprotocolo `sgc`). Ele só é necessário quando:

- Você utiliza um servidor HTTP/WebSocket `sgcWebSockets` que entrega o bundle do cliente JS aos navegadores (típico para servidores baseados em `TsgcWSProtocol_JS_*`).

Ele **não** é necessário para:

- Clientes WebSocket Delphi / C++Builder (os dados são consumidos em código nativo, não em um navegador).
- Servidores que não servem o cliente JS empacotado (por exemplo, servidores somente WebSocket, servidores nativo-para-nativo).

Para excluir o recurso incorporado, comente a linha em `sgcVer.inc`:

```
{.$DEFINE SGC_RESOURCES} { RESOURCES }
```

Alternativamente, ao executar o [Install Setup](#) da edição de código-fonte, desmarque a opção **Include Resources** na página Options; o instalador reescreverá o `sgcVer.inc` antes de compilar os pacotes.

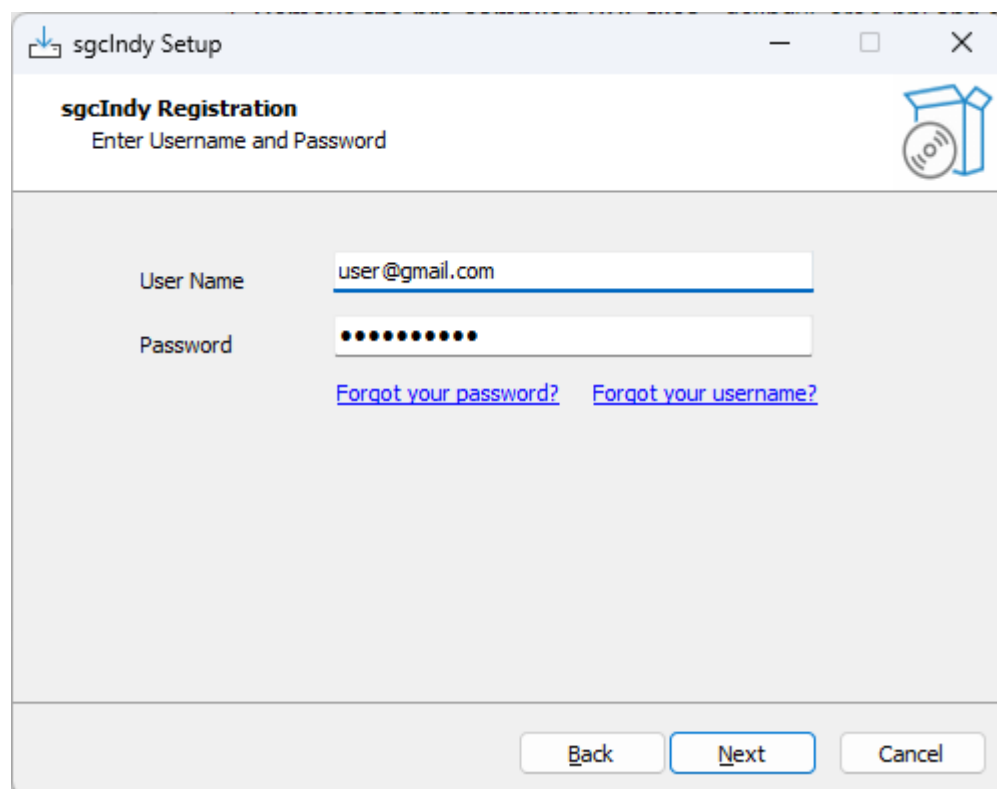
Instalar o Pacote sgclndy

Setup Installation

*Requer Windows Vista no mínimo (Windows 2000, XP e Server 2003 não são suportados).

Usuários que adquiriram uma licença podem instalar o pacote sgclndy utilizando o setup. Encontre abaixo as instruções passo a passo para instalar o pacote.

- Execute o Instalador.
- Primeiro você deve definir o nome de usuário/senha da sua conta privada eSeGeCe. Isto só precisa ser inserido uma vez; na próxima vez que você utilizar o setup, o instalador lerá o último valor.



sgclndy Setup

sgclndy Registration
Enter Username and Password

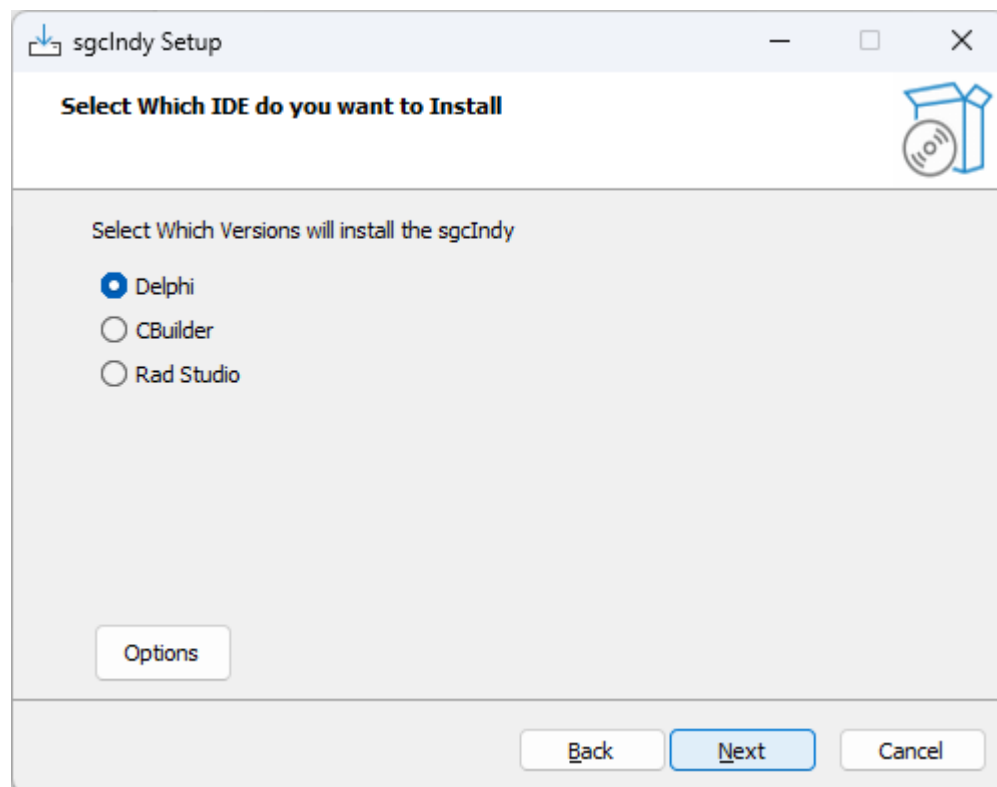
User Name: user@gmail.com

Password:

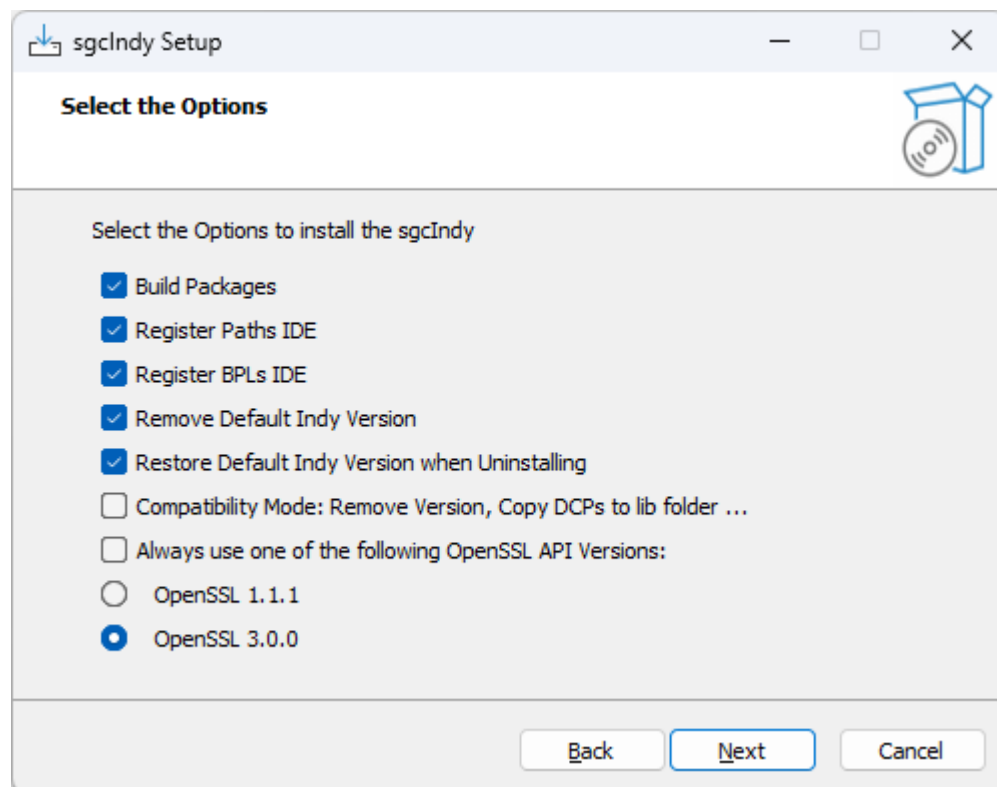
[Forgot your password?](#) [Forgot your username?](#)

Back Next Cancel

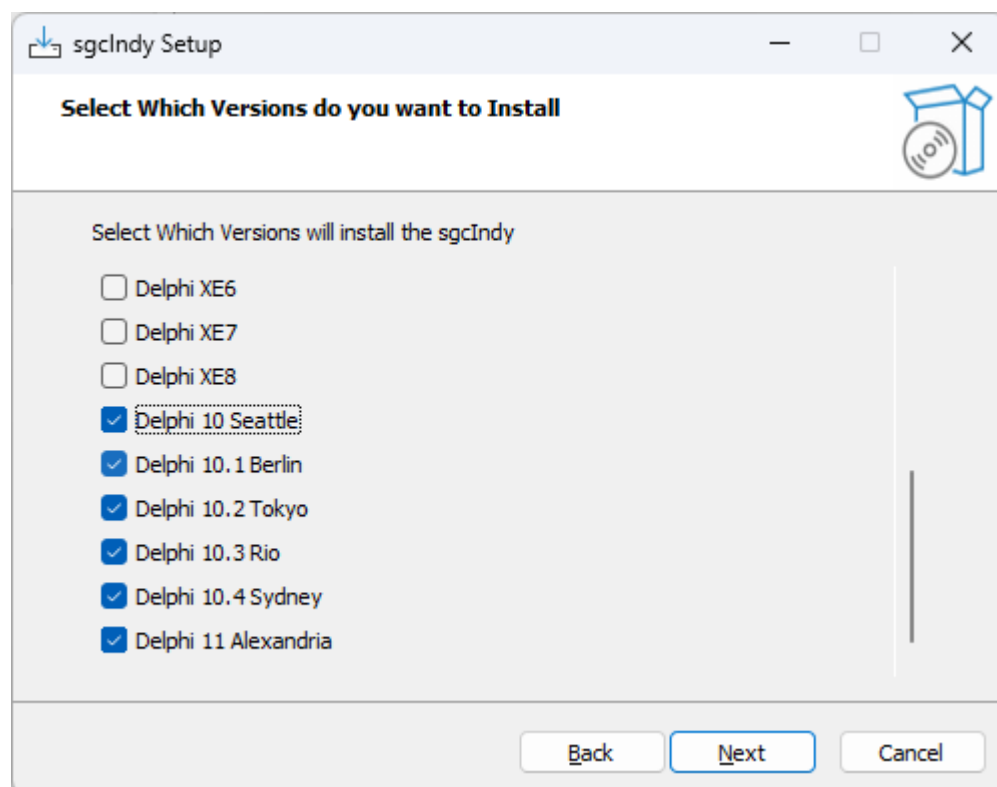
- Se o usuário fez login com sucesso, selecione se você quer instalar na IDE Delphi, CBuilder ou Rad Studio.



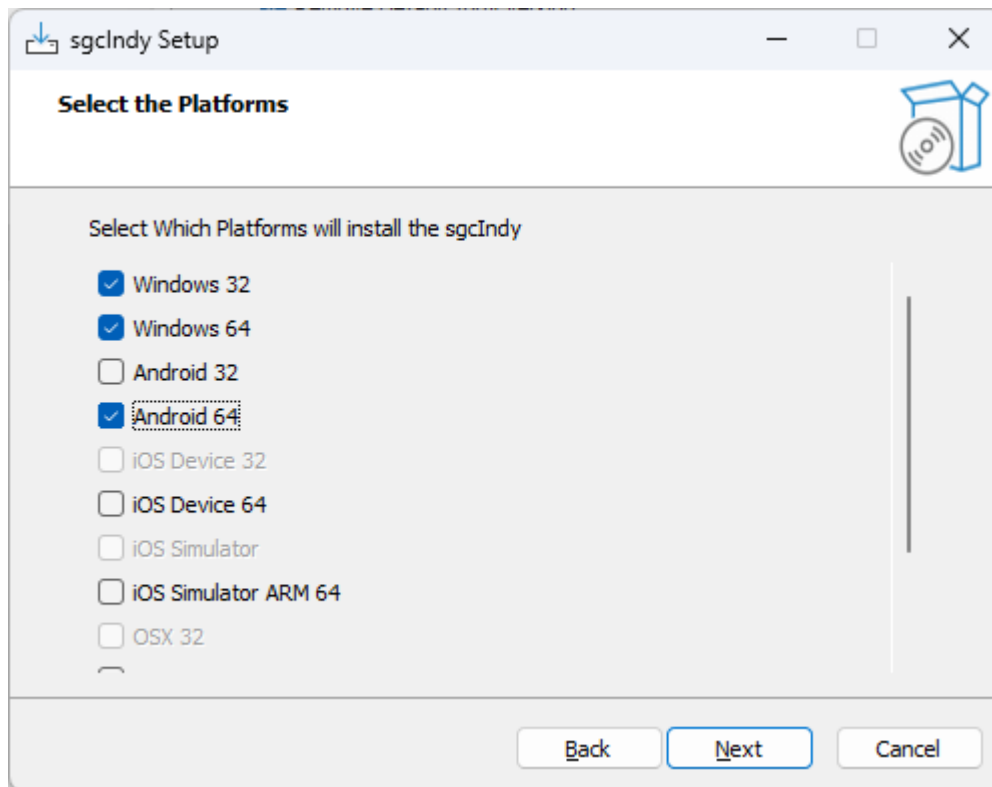
- Há algumas opções que podem ser personalizadas cada vez que você usa o instalador, pressione o botão **Options** para acessar essas propriedades.
 - **Build Packages:** se selecionado, o instalador tentará compilar os pacotes.
 - **Register Paths IDE:** se selecionado, o instalador registrará os caminhos de biblioteca necessários na IDE.
 - **Register BPLs IDE:** se selecionado e o instalador tiver compilado os pacotes com sucesso, o instalador registrará o pacote de tempo de design na IDE.
 - **Remove Default Indy Version:** se selecionado, o instalador desinstalará primeiro a versão Indy padrão que vem com o Rad Studio.
 - **Restore Default Indy Version when Uninstalling:** se selecionado, o instalador restaura a versão Indy padrão desinstalada quando o pacote é desinstalado.
 - **Compatibility Mode:** se selecionado, os arquivos dcp são compilados sem versão e copiados para a pasta Embarcadero/lib. Marque esta opção se outros pacotes estiverem fazendo uso dos pacotes Indy, como o DevExpress. Além disso, ela compila as unidades Embarcadero IP Abstraction com a versão sgclndy instalada.
 - **Always use of the following OpenSSL API Versions:** marque esta opção se você quiser forçar o uso das APIs OpenSSL 1.1.1 ou OpenSSL 3.0.0



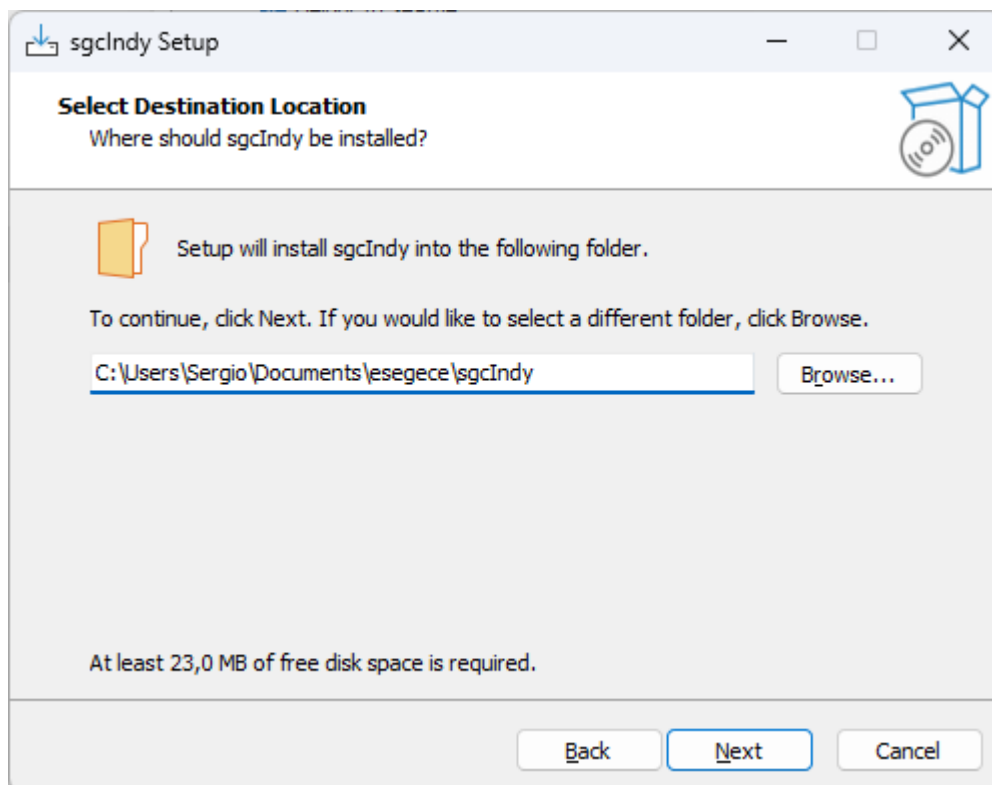
- Agora você pode selecionar quais Versões da IDE deseja instalar. Apenas as versões da IDE que o instalador detectar como instaladas estarão disponíveis.



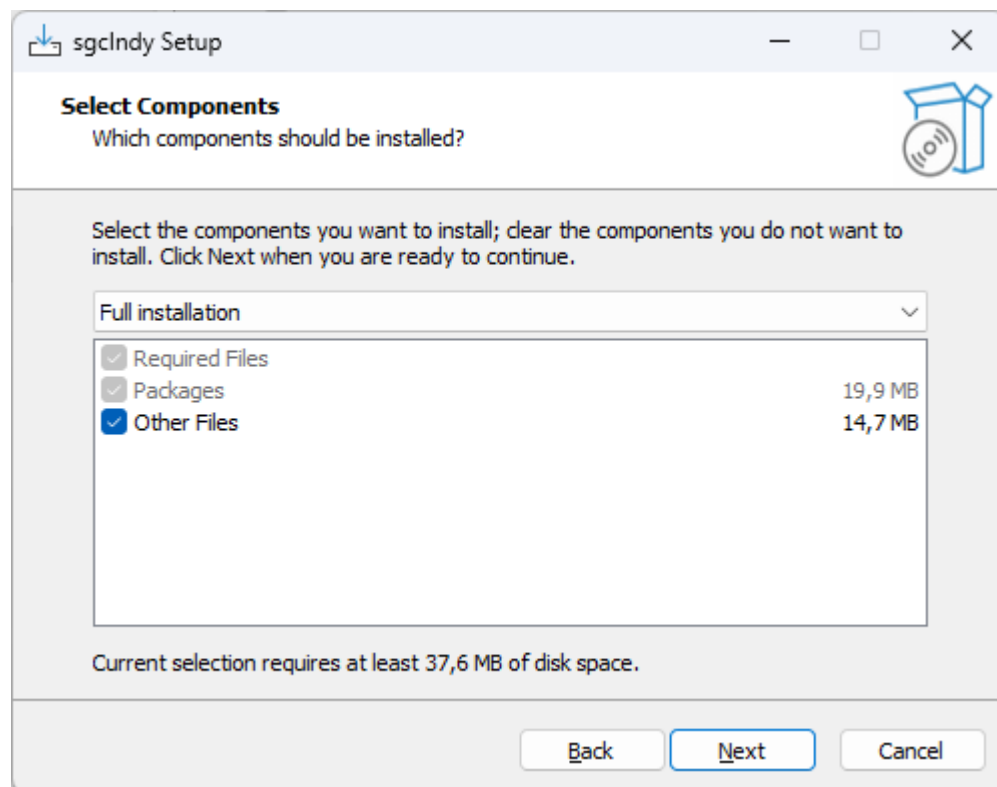
- A próxima etapa é selecionar as Platforms.



- Selecione a pasta onde o pacote será instalado. Se você reinstalar o pacote, o instalador selecionará por padrão a mesma pasta selecionada na instalação anterior.



- Selecione quais componentes instalar.



- Por fim, ele extrairá os arquivos, compilará e instalará o pacote e registrará os caminhos necessários na IDE.

Install Errors

- O MsBuild gera um erro se o comprimento do **Library Path for muito longo**. Para corrigir esse problema, tente excluir os caminhos não utilizados do library path. O MsBuild tem uma limitação de 32K caracteres.

Instalação manual

Se o Indy já estiver **instalado**, ele precisa ser **desinstalado** primeiro.

1. **Remova os arquivos BPL pré-compilados** - dclIndyCoreX.bpl e dclIndyProtocolsX.bpl - da IDE através da caixa de diálogo "Components > Install Packages".
2. Em seguida, **exclua todos os binários existentes** (IndySystemX., IndyCoreX., IndyProtocolsX., dclIndyCoreX. e dclIndyProtocolsX.*), bem como exclua quaisquer arquivos de código-fonte do Indy 10, se presentes.
3. Não deixe de **verificar a existência de arquivos** nas pastas \bin, \lib e \source da IDE, nas subpastas \Indy e nas pastas de sistema do SO."

Para **compilar o pacote sgclndy**, você pode

1. **(Apenas CBuilder)** Utilize o script de linha de comando **FULLC#.BAT** que corresponde à sua versão do CBuilder.
2. **Abra os arquivos DPK individuais** na IDE e **compile-os**, na seguinte ordem:
 1. IndySystemX.dpk (em Lib\System)
 2. IndyCoreX.dpk (em Lib\Core)
 3. IndyProtocolsX.dpk (em Lib\Protocols)
 4. dclIndyCoreX.dpk (em Lib\Core)
 5. dclIndyProtocolsX.dpk (em Lib\Protocols)

Uma vez que os pacotes Indy tenham sido compilados, vá ao menu **Components / Install Packages** e instale os pacotes design-time do Indy

1. dclIndyCore*.bpl
2. dclIndyProtocols*.bpl

Por fim, defina os caminhos na sua IDE para os pacotes sgclIndy.

Configure ZLib

ZLib version: 1.2.12

O sgcWebSockets utiliza a compressão ZLib quando a Extensão WebSocket [Compression PerMessage Deflate](#) está habilitada. Por padrão, o ZLib é vinculado estaticamente à sua aplicação, então não há necessidade de distribuir a biblioteca ZLib.

Se você quiser usar uma biblioteca específica, adicione o seguinte Conditional Define ao seu projeto:

```
SGC_DYNAMICLOAD_ZLIB
```

Como alternativa, você pode editar o arquivo sgclndy.inc (localizado na pasta de código-fonte) e adicionar a seguinte linha

```
{ $DEFINE SGC_DYNAMICLOAD_ZLIB }
```

Por fim, você deve definir o local onde está a biblioteca ZLib; para isso, use o método a seguir e passe o Caminho Completo (sem o nome da biblioteca) onde ela está localizada

```
sgcldZLibHeaders.IdZLibSetLibPath('c:\software\zlib');
```

**Esta configuração só é válida para a sgcWebSockets Enterprise Edition com código-fonte.*

QuickStart

WebSockets Components

Criar um novo servidor WebSocket ou cliente WebSocket é muito simples, basta criar uma nova instância da classe, configurar Host / Port e definir a propriedade Active = true para iniciar o processo.

[Início rápido WebSockets](#)

Componentes HTTP

O protocolo HTTP/2 permite que você crie servidores / clientes HTTP muito mais rápidos do que utilizando o protocolo HTTP/1. O servidor HTTP/2 está incluído no servidor WebSocket, enquanto o cliente HTTP/2 é um componente dedicado que implementa o protocolo HTTP/2.

[QuickStart HTTP](#)

Fluxo de threads

Os componentes do sgcWebSockets são threaded, o que significa que **as conexões são executadas em threads secundárias**. Por **padrão**, os principais **eventos são despachados na thread principal**, o que é útil quando o número de eventos a despachar é baixo, mas para **melhor desempenho** você pode configurar os componentes onde os **eventos são despachados no contexto da thread da conexão**. Leia o seguinte artigo que explica como configurar o fluxo de threads:

[Como Configurar NotifyEvents](#)

Como Compilar Aplicações

Construir aplicações com a biblioteca sgcWebSockets é muito fácil. Basta seguir as próximas dicas, que ajudarão você a **construir sua aplicação com sucesso**.

[Compilação](#)

Servidor de Alto Desempenho

O sgcWebSockets tem **2 implementações de servidor**: 1 baseada no **servidor Indy** e outra baseada no **Servidor HTTP.SYS da Microsoft**. A última é recomendada para servidores de alto desempenho que precisam tratar milhares de conexões concorrentes. Confira o artigo a seguir, que explica como melhorar o desempenho do servidor.

[Servidor de Alto Desempenho](#)

Gerenciador de Memória

Escolher um gerenciador de memória adequado pode melhorar o desempenho da sua aplicação. Confira o artigo a seguir, que mostra uma comparação entre vários gerenciadores de memória.

[Memory Manager](#)

OpenSSL

Quando sua aplicação requer conexões seguras, geralmente as **bibliotecas openssl** são necessárias para **criptografar as comunicações**, siga os próximos passos para configurar com sucesso sua aplicação com as bibliotecas openssl.

[Configure OpenSSL](#)

Indy

A biblioteca Indy é usada como base em alguns componentes do sgcWebSockets, a edição Enterprise do sgcWebSockets inclui uma versão personalizada do indy que permite usar openssl 1.1.1 e openssl 3.0.0, ALPN...

[Indy](#)

Linux (Lazarus)

Se você compilar um projeto Lazarus para Linux e receber esta mensagem:

```
Falha na inicialização do semáforo (possivelmente threads concorrentes em excesso)
```

Basta adicionar a unit **cthreads** ao arquivo do seu projeto.

QuickStart | WebSockets

Vamos começar com um exemplo básico onde precisamos criar um Servidor WebSocket e 2 tipos de cliente WebSocket: Cliente de Aplicação e Cliente Navegador Web.

WebSocket Server

1. Crie uma nova Aplicação Windows Forms
2. Coloque um TsgcWebSocketServer em um Formulário.
3. Na aba Events, dê um duplo clique no Evento OnMessage e digite o seguinte código:

```
void OnMessage(TsgcWSConnection *Connection, const string Text)
{
    ShowMessage("Message Received From Client: " + Text);
}
```

4. Solte um Button no Form, dê um duplo clique e digite este código:

```
TsgcWebSocketServer1->Active = true;
```

WebSocket Client

1. Crie uma nova Aplicação Windows Forms
2. Solte um TsgcWebSocketClient em um Formulário e configure as Propriedades Host e Port para conectar ao Servidor.
3. Coloque um TButton em um Form, dê um duplo clique e digite este código:

```
TsgcWebSocketClient1->Active = true;
```

4. Solte um Button no Form, dê um duplo clique e digite este código:

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

Web Browser Client

1. Crie um novo arquivo HTML
2. Abra o arquivo com um editor de texto e copie o seguinte código:

```
<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>
```

Você precisa substituir host e port neste arquivo pelo seu Host e Port personalizados!!

3. Salve o arquivo e pronto, você configurou um cliente WebSocket básico para navegador web.

How To Use

1. Inicie a Server Application e pressione o botão para iniciar o WebSocket Server para escutar novas conexões.
2. Inicie a Aplicação Cliente e pressione button1 para conectar ao servidor e pressione button2 para enviar uma mensagem. No lado do Servidor, você verá uma mensagem com o texto enviado pelo Cliente.
3. Abra então o arquivo HTML com seu Navegador Web (Chrome, Firefox, Safari ou Internet Explorer 10+), pressione Open para abrir uma conexão e pressione send para enviar uma mensagem ao servidor. No lado do Servidor, você verá uma mensagem com o texto enviado pelo Cliente do Navegador Web.

Linux Compiler

Exemplo de servidor simples (escutando na porta 5000).

```
program sgcWebSockets_linux;
{$APPTYPE CONSOLE}
{$R *.res}

uses
  System.SysUtils, sgcWebSocket, sgcWebSocket_Types;

var
  oServer: TsgcWebSocketServer;

begin
  try
    oServer := TsgcWebSocketServer.Create(nil);
    oServer.Port := 5000;
    oServer.NotifyEvents := neNoSync;
    oServer.Active := True;

    while oServer.Active do
      Sleep(10);
    except
      on E: Exception do
        Writeln(E.ClassName, ': ', E.Message);
      end;
    end.
end.
```

Linux (Lazarus)

Se você compilar um projeto Lazarus para Linux e receber esta mensagem:

Falha na inicialização do semáforo (possivelmente threads concorrentes em excesso)

Basta adicionar a unit **cthreads** ao arquivo do seu projeto.

QuickStart | HTTP

Vamos começar com um exemplo básico onde precisamos criar um Servidor HTTP/2 e um cliente HTTP/2.

HTTP/2 Server

1. Crie uma nova Aplicação Windows Forms
2. Coloque um TsgcWebSocketHTTPServer em um Formulário.
3. Na aba Events, clique duas vezes no evento OnCommandGet e digite o seguinte código:

```
void OnCommandGet(TidContext *AContext, TidHTTPRequestInfo *ARequestInfo,
    TidHTTPResponseInfo *AResponseInfo)
{
    if (ARequestInfo->Document == "/" )
    {
        AResponseInfo->ContentText = "<html><head><title>Test Page</title></head><body></body></html>";
        AResponseInfo->ContentType = "text/html";
        AResponseInfo->ResponseNo = 200;
    }
}
```

4. Por padrão, o servidor só habilita conexões HTTP/1, portanto habilite as opções HTTP/2 na propriedade **HTTP2Options.Enabled = true** e, em seguida, configure as SSL Options. Conexões seguras requerem [bibliotecas 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 = sslAPI_1_1;
TsgcWebSocketHTTPServer1->SSLOptions->Port = 443;
TsgcWebSocketHTTPServer1->SSLOptions->Version = tls1_3;
```

5. Coloque um Button no Form, faça duplo clique e digite este código:

```
TsgcWebSocketHTTPServer1->Active = true;
```

HTTP/1 Client

1. Crie uma nova Aplicação Windows Forms
2. Coloque um TButton em um Form, dê Duplo Clique e digite este código:

```
TsgcHTTP1Client *oHTTP1 = new TsgcHTTP1Client();
try
{
    ShowMessage(oHTTP1->Get("https://127.0.0.1"));
}
finally
{
    oHTTP1->Free();
}
```


Cliente HTTP/2

1. Crie uma nova Aplicação Windows Forms
2. Coloque um TButton em um Form, dê Duplo Clique e digite este código:

```
TsgcHTTP2Client *oHTTP2 = new TsgcHTTP2Client();  
try  
{  
    ShowMessage(oHTTP2->Get("https://127.0.0.1"));  
}  
__finally  
{  
    oHTTP2->Free();  
}
```

QuickStart | Threading Flow

Os componentes `sgcWebSockets` são baseados em threads; por exemplo, `TsgcWebSocketHTTPServer` (baseado na biblioteca Indy) cria uma thread para cada conexão, enquanto `TsgcWebSocketServer_HTTPAPI` (baseado no HTTP.SYS da Microsoft) executa um pool de threads e as conexões são tratadas por esse pool de threads (máximo de 64 threads) e `TsgcWebSocketClient` executa sua própria thread para processar de forma assíncrona as respostas do servidor WebSocket.

Por padrão, há uma propriedade chamada **NotifyEvents**, que tem o valor `neAsynchronous`. Isso significa que, quando um cliente WebSocket recebe uma mensagem, esta mensagem é enfileirada e despachada na thread principal pelo SO posteriormente. Isso funciona bem para clientes que não recebem muitas mensagens e para facilitar o uso, porque não exige sincronização com a thread principal quando você deseja, por exemplo, atualizar um controle no seu formulário.

Mas quando o servidor / cliente precisa processar várias mensagens em um curto período de tempo, é melhor alterar este fluxo de threads para um em que os eventos sejam despachados no contexto da thread de conexão. Para fazer isso, basta definir a propriedade `NotifyEvents` como **neNoSync**, dessa forma, quando por exemplo um cliente recebe uma mensagem do servidor, esta mensagem será despachada no contexto de uma thread secundária, portanto se você precisar atualizar um controle do seu formulário, primeiro sincronize com a main thread e então atualize o controle do formulário (porque os controles de formulário não são thread safe). O mesmo se aplica se você quiser acessar um objeto compartilhado, você precisa implementar seus próprios métodos de sincronização.

Threading Flow Easy Mode (`NotifyEvents = neAsynchronous`) e Baixo Desempenho

Este é o fluxo de threads por padrão e normalmente é utilizado em exemplos de demonstração. Selecione este modo se você não espera tratar várias mensagens por segundo e precisa atualizar controles de formulário ou acessar objetos compartilhados.

`NotifyEvents = neAsynchronous`

Fluxo de threads de melhor desempenho (`NotifyEvents = neNoSync`)

Defina este fluxo de threads para componentes servidor e para clientes que precisam de alto desempenho porque você espera tratar várias mensagens. Utilizando esta configuração, os eventos são despachados no contexto da thread de conexão, portanto, para atualizar um controle de Formulário, primeiro sincronize com a main thread.

`NotifyEvents = neNoSync`

Como sincronizar a thread principal

Você pode sincronizar com a Thread Principal chamando `TThread.Synchronize` ou `TThread.Queue`; ambos os métodos podem ser utilizados e a escolha de um ou outro depende de como você quer implementar a sincronização.

`TThread.Synchronize`

Este método é bloqueante, o que significa que, quando você chama `Synchronize`, o código bloqueia até que se sincronize com a thread principal.

`TThread.Queue`

Este método não é bloqueante, então, quando você chama `queue`, a mensagem é enfileirada e será despachada posteriormente.

Código de Exemplo

Atualiza um Memo com as mensagens recebidas do Cliente WebSocket.

```
void OnClientMessage(TsgcWSConnection *Connection, string aText)
{
    TThread::Queue(NULL,
        [&]() {
            memo1->lines->add(aText);
        });
}
```

QuickStart | Build

Construir uma aplicação com a biblioteca sgcWebSockets é muito fácil. Apenas tenha em mente se os seus componentes requerem ou não as bibliotecas OpenSSL. Se as suas aplicações requerem conexões seguras, as bibliotecas openSSL devem ser implantadas (exceto se você utilizar [SChannel para windows](#) em componentes cliente).

Para **aplicações Windows**, basta implantar as bibliotecas OpenSSL na mesma pasta onde a aplicação está localizada.

Para outras personalities, confira os seguintes artigos:

- [Build OSX Application](#)
- [Build Android Application](#)
- [Build iOS Application](#)

CBuilder DEBUG

Se você está usando CBuilder e deseja depurar a biblioteca sgcWebSockets, siga os próximos passos:

- Acesse as Opções do Projeto.
- Habilite Use debug .dcus em Delphi Compiler > Compiling.
- Desabilite Link with Dynamic RTL em C++ Linker.
- Desabilite Link with Runtime Packages em Packages > Runtime Packages.

Você poderá então entrar no código-fonte da VCL/RTL.

Build | OSX Application

Para construir uma Aplicação OSX com a biblioteca sgcWebSockets, você deve seguir os passos do site da Embarcadero para construir uma Aplicação OSX.

Install PASServer in MacOS

http://docwiki.embarcadero.com/RADStudio/Sydney/en/PAServer,_the_Platform_Assistant_Server_Application

Obter um Developer ID Certificate

Faça login com uma Apple Developer Account válida em <https://developer.apple.com> e crie um novo "Developer ID Application" a partir do menu Certificates.

http://docwiki.embarcadero.com/RADStudio/Rio/en/MacOS_Notarization

Criar um novo Apple Id

Em seguida, vá para <https://appleid.apple.com/account> para criar um novo Apple Id

Configurar provisionamento

Por fim, abra o menu **Project / Options / Provisioning** e preencha os dados necessários para notarizar uma aplicação OSX.

Provisioning

Target
Release configuration - macOS 64-bit platform

Build type
macOS 64-bit - Developer ID

Apple ID
your@email.com

App-specific Password
.....

Developer ID Application Certificate
Developer ID Application: *****

Additional options to pass to the notarization command-line tool

☐ Attach a ticket to the notarized application to allow it to run offline

Se o seu projeto requer algumas bibliotecas, não se esqueça de incluí-las no menu **Project / Deployment**. Defina o Remote Path como "Contents\MacOS\"

Local Path	Local Name	Type	Configurat...	Platforms	Remote Path	Remote Name	Remote Status	Overwrite
☒	libcrypto.1.0.0.dylib	File	Release	[OSX64]	Contents\MacOS\	libcrypto.1.0.0.dylib	Not Connected	Always
☒	OSX64\Release\sgcClientMobile	ProjectOutput	Release	[OSX64]	Contents\MacOS\	sgcClientMobile	Not Connected	Always
☒	\$(BDS)\bin\delphi_PROJECTICNS.icns	ProjectOSXRes...	Release	[OSX64]	Contents\Resources\	sgcClientMobile.icns	Not Connected	Always
☒	OSX64\Release\sgcClientMobile.dSYM	ProjectOSXDe...	Release	[OSX64]	..\\$(PROJECTNAME).ap...	sgcClientMobile	Not Connected	Always
☒	OSX64\Release\sgcClientMobile.info.plist	ProjectOSXInf...	Release	[OSX64]	Contents\	Info.plist	Not Connected	Always
☒	libssl.1.0.0.dylib	File	Release	[OSX64]	Contents\MacOS\	libssl.1.0.0.dylib	Not Connected	Always
☒	OSX64\Release\sgcClientMobile.entitle...	ProjectOSXEnt...	Release	[OSX64]	..\	sgcClientMobile.entitle...	Not Connected	Always

Essas bibliotecas serão assinadas automaticamente quando a aplicação for notarizada, você pode verificar se a biblioteca foi assinada utilizando o seguinte comando:

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

Leia mais sobre como [Configurar openssl OSX](#).

Build | Aplicativo Android

Para compilar uma Aplicação Android com a biblioteca sgcWebSockets, você deve seguir os passos do site da Embarcadero para compilar uma Aplicação Android.

Criando um App Android

http://docwiki.embarcadero.com/RADStudio/Sydney/en/Creating_an_Android_App

Project Deployment

Se o seu projeto requer algumas bibliotecas, não esqueça de incluí-las no menu **Project / Deployment**. Defina Remote Path como ".\assets\internal"

Local Path	Local Name	Type	Configurat...	Platforms	Remote Path	Remote Name
☒ \$(BDS)\bin\Artwork\...	FM_LauncherIcon_72x7...	Android_Laun...	Release	[Android64]	res\drawable-hdpi\	ic_launcher.png
☒ \$(BDS)\bin\Artwork\...	FM_SplashImage_960x7...	Android_Splas...	Release	[Android64]	res\drawable-xlarge\	splash_image.png
☒ ..\..\apps\Third-pa...	libssl.so	File	Release	[Android64]	.\assets\internal	libssl.so
☒ \$(BDS)\lib\android\...	libnative-activity.so	AndroidLibnat...	Release	[Android64]	library\lib\armeabi\	libsgcClientMobile.so
☒ Android64\Release\	strings.xml	Android_Strings	Release	[Android64]	res\values\	strings.xml
☒ \$(BDS)\bin\Artwork\...	FM_SplashImage_640x4...	Android_Splas...	Release	[Android64]	res\drawable-large\	splash_image.png
☒ Android64\Release\	classes.dex	AndroidClasse...	Release	[Android64]	classes\	classes.dex
☒ Android64\Release\	styles-v21.xml	AndroidSplash...	Release	[Android64]	res\values-v21\	styles.xml
☒ \$(BDS)\bin\Artwork\...	FM_LauncherIcon_48x4...	Android_Laun...	Release	[Android64]	res\drawable-mdpi\	ic_launcher.png
☒ \$(BDS)\bin\Artwork\...	FM_SplashImage_470x3...	Android_Splas...	Release	[Android64]	res\drawable-normal\	splash_image.png
☒ \$(BDS)\bin\Artwork\...	FM_SplashImage_426x3...	Android_Splas...	Release	[Android64]	res\drawable-small\	splash_image.png
☒ \$(BDS)\lib\android\...	libnative-activity.so	AndroidLibnat...	Release	[Android64]	library\lib\armeabi-v7a\	libsgcClientMobile.so
☒ Android64\Release\	libsgcClientMobile.so	ProjectOutput	Release	[Android64]	library\lib\arm64-v8a\	libsgcClientMobile.so
☒ ..\..\apps\Third-pa...	libcrypto.so	File	Release	[Android64]	.\assets\internal	libcrypto.so
☒ Android64\Release\	styles.xml	AndroidSplash...	Release	[Android64]	res\values\	styles.xml

Leia mais sobre como [configurar o openssl no Android](#).

Build | iOS Application

Para compilar uma aplicação iOS com a biblioteca sgcWebSockets, você deve seguir os passos do site da Embarcadero para compilar uma aplicação iOS.

Install PASServer in MacOS

http://docwiki.embarcadero.com/RADStudio/Sydney/en/PAServer_the_Platform_Assistant_Server_Application

Obter um iOS Development Certificate

Faça login com uma Apple Developer Account válida em <https://developer.apple.com> e crie um novo "iOS Development Certificate" no menu Certificates.

Crie um novo Identifier para seus apps iOS e um novo provisioning profile.

http://docwiki.embarcadero.com/RADStudio/Sydney/en/iOS_Mobile_Application_Development

Configurar o Bundle Identifier

Abra o menu **Project / Options / Application / Version Info** e defina seu Bundle Identifier

Version Info

Target
Release configuration - iOS Device 64-bit platform

☒ Include version information in project

Module version number

Major version	Minor version	Build
1	0	0

Build number options
Do not change build number

Key	Value
CFBundleName	\$(ModuleName)
CFBundleDevelopmentRegion	en
CFBundleDisplayName	\$(ModuleName)
CFBundleIdentifier	com.esegece.\$(ModuleName)
CFBundleInfoDictionaryVersion	6.0
CFBundleVersion	1.0.0
CFBundlePackageType	APPL
CFBundleSignature	????
LSRequiresiPhoneOS	true

Deployment

Se o seu projeto requer algumas bibliotecas estáticas, copie essas bibliotecas para a pasta lib/iosDevice64 do Embarcadero:

- C:\Program Files (x86)\Embarcadero\Studio\<IDE Version>\lib\iosDevice64\debug
- C:\Program Files (x86)\Embarcadero\Studio\<IDE Version>\lib\iosDevice64\release

Leia mais sobre Como [Configurar o openssl iOS](#)

Provisioning

Por fim, verifique no menu **Project / Options / Deployment**, se o certificado foi carregado com sucesso.

Provisioning

Target

Release configuration - iOS Device 64-bit platform

Apply...

Save...

<Use environment options (Auto)>

Developer Certificate:

<Auto> - To select a certificate, first select a valid provision profile

Provision Profile

Name:

provisioning esegece

File Path:

/Users/sergio/Library/MobileDevice/Provisioning Profiles/

Application Identifier:

Developer program Name:

Sergio Gomez

Developer Certificate

iPhone Developer: Sergio Gomez

Current Bundle Identifier:

com.esegece.sgcClientMobile

Servidores de Desempenho Rápido

Servidores baseados na biblioteca Indy

[TsgcWebsocketServer](#) e [TsgcWebsocketHTTPServer](#) são baseados na biblioteca Indy, portanto cada conexão é tratada por uma thread. Assim, se você tiver 1000 conexões simultâneas, terá, no mínimo, 1000 threads para tratar essas conexões. Quando o desempenho é importante, você deve fazer alguns "ajustes" para aumentar o desempenho e melhorar o trabalho do servidor. **A partir do sgcWebSockets 4.3.3, os servidores Indy também suportam IOCP**, você pode [ler mais](#).

Utilize as dicas a seguir para aumentar o desempenho do servidor.

1. Defina na propriedade do componente Server **NotifyEvents := neNoSync**. Isso significa que os eventos são gerados no contexto da thread de conexão, portanto não há mecanismo de sincronização. Se você precisar acessar controles VCL ou objetos compartilhados, utilize seus próprios mecanismos de sincronização.

2. Defina na propriedade do componente Server **Optimizations.Connections.Enabled := True**. Se você planeja ter mais de 1000 conexões simultâneas no seu servidor, e chama o método `Server.WriteData` com frequência, habilite esta propriedade. Basicamente, ela salva as conexões em uma lista de cache onde as buscas são mais rápidas do que acessar a lista de conexões do Indy.

2.1 CacheSize: é o número de conexões armazenadas em um cache rápido. Valor padrão = 100.

2.2 GroupLevel: cria internamente várias listas divididas pelo primeiro caractere, então, se você tiver muitas conexões, as buscas são mais rápidas. Valor padrão = 1.

3. Defina na propriedade do componente Server **Optimizations.Channels.Enabled := True**. Ao habilitar esta propriedade, os canais são salvos em uma lista onde as buscas são mais rápidas que no método anterior.

4. Defina na propriedade do componente Server **Optimizations.ConnectionsFree.Enabled := True**. Se esta propriedade estiver habilitada, toda vez que houver uma desconexão, em vez de destruir o `TsgcWSCConnection`, o objeto é armazenado em uma Lista e, a cada X segundos, todos os objetos armazenados nessa lista são destruídos. Ao habilitar esta propriedade, o consumo de memória será maior, portanto, se você tiver muitas desconexões em um curto período de tempo, defina esta propriedade como false.

4.1 Interval: número de segundos em que todas as conexões desconectadas armazenadas em uma lista são destruídas. Por padrão é 60.

5. Por padrão, o `sgcWebSockets` utiliza **Critical Sections** para proteger o acesso a objetos compartilhados. Mas você pode utilizar `TMonitor` ou `SpinLocks` em vez de critical sections. Basta compilar seu projeto com um dos seguintes defines de compilador

3.1 `{ $DEFINE SGC_SPINLOCKS }`

3.2 `{ $DEFINE SGC_TMONITOR }`

6. Use o **FastMM4** mais recente, você pode baixá-lo em: <https://github.com/pleriche/FastMM4>

O `FastMM4` é um gerenciador de memória muito bom, mas às vezes não escala bem com aplicações multithread. Utilize o seguinte define de compilador em sua aplicação:

`{ $DEFINE UseReleaseStack }`

Em seguida, adicione `FastMM4` como a primeira unit no uses do seu projeto e compile novamente. Para um servidor de alta concorrência, você notará um aumento no desempenho.

Este ajuste faz o seguinte: Se um bloco não puder ser liberado imediatamente durante uma chamada `FreeMem`, o bloco será adicionado a uma lista de blocos que serão liberados posteriormente, seja na thread de limpeza em segundo plano ou durante a próxima chamada a `FreeMem`.

7. Melhor que o `FastMM4`, use o **FastMM5** mais recente, que você pode baixar de: <https://github.com/pleriche/FastMM5>

Esta é uma nova versão do mesmo desenvolvedor do `FastMM4`, suporta Delphi XE3 e posteriores e pode ser usada no `Windows32` e `Windows64`.

O FastMM5 tem licença dupla, portanto existem 2 licenças: GPL e Comercial. Assim, se você quiser utilizá-lo em projetos comerciais, deve adquirir uma licença.

Veja abaixo um grid que compara o desempenho entre FastMM4 e FastMM5, fazendo 100.000 requisições e respostas websocket utilizando 1, 10, 100, 500 e 1000 clientes concorrentes. O desempenho sob FastMM5 é muito melhor, em aplicações multithread, do que utilizando FastMM4.

Cientes	Windows	FMM4	FMM5	Diferença
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

A Edição Enterprise do sgcWebSockets suporta **IOCP** no Windows, isso significa que, em vez de criar 1 thread para cada conexão, um pool de threads trata todas as conexões. Para habilitar o IOCP, basta definir o IOHandler como IOCP.

```
IOHandlerOptions.IOHandlerType = iohIOCP
```

A propriedade IOHandlerOptions.IOCP permite que você personalize as propriedades do IOCP.

- **IOCPThreads:** estas são as threads utilizadas para tratar as conexões; por padrão, o valor é zero, o que significa que as threads serão calculadas automaticamente utilizando o número de processadores (para Delphi 7 a Delphi 2007, este valor é definido como 32 porque a função de contagem de CPU não é suportada).
- **WorkOpThreads:** defina um valor maior que zero se você quiser que as requisições de cada conexão sejam sempre tratadas pela mesma thread. Por padrão, as requisições IOCP são tratadas por threads aleatórias; se você quiser que as conexões sejam sempre tratadas pela mesma thread, defina um valor maior que zero. Exemplo: se você definir WorkOpThreads = 32, o servidor criará 32 threads e, toda vez que houver uma nova requisição, se a conexão já tiver sido processada anteriormente, ela será enfileirada na mesma thread.

O IOCP é recomendado quando você deseja tratar milhares de conexões simultâneas.

Indy Server Linux

A `sgcWebSockets Enterprise Edition` suporta **EPOLL** no Linux, o que significa que, em vez de criar 1 thread para cada conexão, um pool de threads trata todas as conexões. Para habilitar o EPOLL, basta definir o `IOHandler` como EPOLL.

```
IOHandlerOptions.IOHandlerType = iohEPOLL
```

A propriedade `IOHandlerOptions.EPOLL` permite personalizar as propriedades do EPOLL.

- **EPOLLThreads:** estas são as threads usadas para tratar as conexões; por padrão, o valor é zero, o que significa que as threads serão calculadas automaticamente usando o número de processadores.
- **WorkOpThreads:** defina um valor maior que zero se você quiser que as requisições de cada conexão sejam sempre tratadas pela mesma thread. Por padrão, as requisições EPOLL são tratadas por threads aleatórias; se você quiser que as conexões sejam tratadas sempre pela mesma thread, defina um valor maior que zero. Exemplo: se você definir `WorkOpThreads = 32`, o servidor criará 32 threads e, toda vez que houver uma nova requisição, se a conexão já foi processada anteriormente, ela será enfileirada na mesma thread.

EPOLL é recomendado quando você deseja tratar milhares de conexões simultâneas.

Servidor Baseado em HTTP.SYS

O componente `TsgcWebSocketServer_HTTPAPI` é baseado na Microsoft HTTP API e foi projetado para funcionar com IOCP, portanto é recomendado quando o servidor deve tratar milhares de conexões, mas tem a limitação de que só pode rodar no Windows.

O servidor pode tratar os protocolos **WebSocket** e **HTTP/2** na mesma porta e pode trabalhar com outras implementações, pois pode ser configurado para tratar apenas alguns endpoints.

Exemplo: você pode configurar este servidor para tratar conexões websocket com nossa biblioteca `sgcWebSockets` e deixar que outras implementações / terceiros ou o que for utilizem outros endpoints.

- Endpoint: `https://server/ws` tratará as conexões que utilizam o protocolo WebSocket utilizando o `sgcWebSockets`
- Endpoint: `https://server/other` tratará as conexões usando outra biblioteca.

Utilize o mais recente **FastMM5**, você pode baixar de: <https://github.com/plieriche/FastMM5>

Esta é uma nova versão do mesmo desenvolvedor do FastMM4, suporta Delphi XE3 e posteriores e pode ser usada no Windows32 e Windows64.

O FastMM5 tem licença dupla, portanto existem 2 licenças: GPL e Comercial. Assim, se você quiser utilizá-lo em projetos comerciais, deve adquirir uma licença.

Veja abaixo um grid que compara o desempenho entre FastMM4 e FastMM5, fazendo 100.000 requisições e respostas websocket utilizando 1, 10, 100, 500 e 1000 clientes concorrentes. O desempenho sob FastMM5 é muito melhor, em aplicações multithread, do que utilizando FastMM4.

Clientes	Windows	FMM4	FMM5	Diferença
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%

Gerenciador de Memória

Recentemente, uma nova versão do FastMM, desenvolvida por Pierre le Riche, foi lançada. A nova versão se chama **FastMM5** e foi reescrita para melhorar o desempenho em aplicações multithread. Pode ser configurada para melhor velocidade ou menor uso de memória, entre outros.

Suporte a partir do **Compilador XE3** do Delphi e pode ser utilizado em **Windows32** e **Windows64**.

O FastMM5 é **licenciado de forma dupla**, portanto há 2 licenças: **GPL** e **Commercial**. Portanto, se você quiser usar em projetos comerciais, deve adquirir uma licença. Mais detalhes aqui

<https://github.com/pleriche/FastMM5>

O FastMM4 tem um novo fork, chamado **FastMM4-AVX**, desenvolvido por Maxim Masiutin, que adiciona recursos muito interessantes como: sincronização mais eficiente, instruções AVX para cópia de memória mais rápida, melhorias de velocidade e mais. O FastMM4-AVX possui licença dupla: MPL e GPL. Mais detalhes aqui:

<https://github.com/maximmasiutin/FastMM4-AVX>

Configuração

Para testar o desempenho com nossos componentes, foi criada uma nova aplicação de console windows, **sgcBenchmark.exe**, que será utilizada para medir o desempenho de cada memory manager utilizando nossos componentes sgcWebSockets.

O teste é muito simples, um cliente (ou mais de um cliente) conecta a um servidor, envia uma mensagem ao servidor e o servidor responde com a mesma mensagem ao cliente. Isso é repetido 100.000 vezes. Os testes são repetidos alterando o número de clientes concorrentes, primeiro 1, depois 10, 100... o tempo medido é o tempo decorrido entre a primeira mensagem enviada pelo cliente e a última mensagem recebida do servidor (portanto o tempo usado para conectar ao servidor não é medido).

O benchmark comparará o desempenho usando o Default Memory Manager que vem com o Delphi 10.4.1, o FastMM5 e o FastMM4-AVX

Benchmark do Servidor WebSocket Indy

No primeiro Benchmark, o Servidor usado é o [Indy WebSocket Server](#), este servidor é baseado no Indy TCP Server, portanto cada conexão cria 1 thread.

Os valores são medidos em milissegundos, então, por exemplo, o primeiro teste feito com 1 cliente em plataformas Windows32, utilizando o gerenciador de memória padrão leva 4135 milissegundos, utilizando o FastMM5 leva 4214 milissegundos e utilizando o FastMM4-AVX leva 4823 milissegundos. A porcentagem calculada é em relação ao valor de referência, neste caso em relação ao gerenciador de memória Default que vem com o delphi; quanto menor a porcentagem, melhor o desempenho obtido.

O Benchmark foi feito 3 vezes e os valores exibidos são a soma dos benchmarks / 3.

Para o benchmark, o servidor usado foi:

- Windows 2016 Server Datacenter
- 16 Processadores Virtuais
- 32 GB RAM
- 2.2 GHz

A versão do Delphi usada foi a Delphi 10.4.1, e as versões mais recentes do FastMM5 e FastMM4-AVX dos servidores do github.

Encontre abaixo o resultado do benchmark.

Clientes	Plataforma	Padrão (ms)	FMM5 (ms)	FMM5 (%)	FMM4-AVX (ms)	FMM4-AVX (%)
1	Win32	4135	4214	1.91%	4823	16.64%
1	Win64	4052	4520	11.55%	4328	6,81%
10	Win32	4214	1729	-58.97%	1828	-56.62%
10	Win64	4104	1875	-54.31%	1651	-59,77%
100	Win32	3958	1604	-59.47%	1583	-60.01%
100	Win64	3958	1614	-59.22%	1635	-58.69%
500	Win32	4098	1723	-57.96%	1854	-54.76%
500	Win64	5333	1791	-66.42%	1833	-65.63%
1000	Win32	5927	2208	-62.75%	2328	-60.72%
1000	Win64	8166	2229	-72.70%	2234	-72,64%

Benchmark HTTP.SYS Server

No segundo Benchmark, o Servidor utilizado é o [HTTP.SYS WebSocket Server](#); este servidor é baseado no HTTP API Microsoft Framework e as conexões são tratadas por um pool de threads.

Os valores são medidos em milissegundos, então, por exemplo, o primeiro teste que é feito com 1 cliente em plataformas Windows32, utilizando o gerenciador de memória padrão leva 5364 milissegundos, utilizando FastMM5 leva 5182 milissegundos e utilizando FastMM4-AVX leva 5838 milissegundos. O percentual calculado é em relação ao valor de referência, neste caso em relação ao gerenciador de memória Default que vem com o Delphi; quanto menor o percentual, melhor o desempenho obtido.

O Benchmark foi feito 3 vezes e os valores exibidos são a soma dos benchmarks / 3.

Para o benchmark, o servidor usado foi:

- Windows 2016 Server Datacenter
- 16 Processadores Virtuais
- 32 GB RAM
- 2.2 GHz

A versão do Delphi usada foi a Delphi 10.4.1, e as versões mais recentes do FastMM5 e FastMM4-AVX dos servidores do github.

Encontre abaixo o resultado do benchmark.

Clientes	Plataforma	Padrão (ms)	FMM5 (ms)	FMM5 (%)	FMM4-AVX (ms)	FMM4-AVX (%)
1	Win32	5364	5182	-3.39%	5838	8.84%
1	Win64	5507	5206	-0.61%	5135	1,54%
10	Win32	4922	1744	-64.57%	2088	-57.58%

10	Win64	4958	1770	-64.30%	1953	-60.61%
100	Win32	3359	1682	-49.93%	2244	-33.19%
100	Win64	3979	1536	-61.40%	1859	-53.28%
500	Win32	2364	1890	-20.05%	2344	-0,85%
500	Win64	2901	1666	-42.57%	1859	-35.92%
1000	Win32	3296	1968	-40.29%	2531	-23.21%
1000	Win64	4469	1989	-55.49%	2047	-54.20%

Comentários sobre Benchmarks

Encontre abaixo alguns comentários sobre os resultados obtidos após o benchmark dos 3 diferentes gerenciadores de memória:

- Ao utilizar em uma aplicação single-thread, não há grandes diferenças de desempenho entre FastMM4, FastMM5 e FastMM4-AVX.
- **FastMM5 e FastMM4-AVX** funcionam muito **melhor** em aplicações **multithread**.
- As **diferenças** entre FastMM5 e FastMM4-AVX são **pequenas**, pelo menos ao realizar estes benchmarks.
- Os benchmarks do **Windows 32** têm desempenho **melhor** que os testes do **Windows 64**. Utilizar FastMM5 ou FastMM4-AVX em aplicações Windows 64 melhora o desempenho mais do que no Windows 32.

A decisão final de escolher um gerenciador de memória ou outro depende do projeto. Acho que não há um único gerenciador de memória que funcione como o melhor em todas as condições, portanto, antes de escolher um ou outro, teste, teste e teste novamente para ver qual tem melhor desempenho para suas necessidades

OpenSSL

O OpenSSL é uma biblioteca de software para aplicações que protegem comunicações em redes de computadores contra interceptação ou que precisam identificar a parte na outra extremidade. É amplamente usado por servidores de Internet, incluindo a maioria dos sites HTTPS.

Esta biblioteca é necessária para componentes baseados na Indy Library quando uma conexão segura é necessária. Se a sua aplicação requer OpenSSL, você deve ter os arquivos necessários no seu sistema de arquivos antes de implantar sua aplicação:

Atualmente, o sgcWebSockets suporta as versões **1.0.2**, **1.1** e **3.0 a 3.3** do **openssl**.

Platform	API 1.0	API 1.1	API 3.*	Linkagem Estática/ Dinâmica
Windows (32-bit e 64-bit)	libeay32.dll e ssleay32.dll	libcrypto-1_1.dll e libssl-1_1.dll	libcrypto-3.dll e libssl-3.dll	Dynamic
OSX	libcrypto.dylib, libssl.dylib	libcrypto.1.1.dylib, libssl.1.1.dylib	libcrypto.3.dylib, libssl.3.dylib	Dynamic
iOS Device (32-bit and 64-bit)	libcrypto.a e libssl.a	libcrypto.a e libssl.a	libcrypto.a e libssl.a	Static
iOS Simulator	libcrypto.dylib, libssl.dylib	libcrypto.1.1.dylib, libssl.1.1.dylib	libcrypto.3.dylib, libssl.3.dylib	Dynamic
Android Device	libcrypto.so, libssl.so	libcrypto.so, libssl.so	libcrypto.so, libssl.so	Dynamic

Veja abaixo como **configurar o OpenSSL** para cada plataforma:

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

Configurações do openssl

Os componentes do sgcWebSockets baseados em Indy permitem que você configure algumas propriedades do OpenSSL. Acesse as seguintes propriedades:

- **Componentes Servidor:** SSLOptions.OpenSSL_Options.
- **Componentes cliente:** TLSOptions.OpenSSL_Options.

API Version

A **biblioteca Indy padrão** permite carregar apenas bibliotecas **OpenSSL 1.0.2**; essas bibliotecas foram descontinuadas e as versões mais recentes do OpenSSL usam a API 1.1.1.

O **sgcWebSockets Enterprise** permite que você carregue as bibliotecas **openssl 1.1.1**; você pode configurar nesta propriedade qual versão da openssl API será carregada. Apenas uma versão da API pode ser carregada por processo (portanto, você não pode misturar bibliotecas openssl 1.0.2 e 1.1.1 na mesma aplicação).

LibPath

Esta propriedade permite que você defina a localização das bibliotecas openssl. Isto é útil para projetos Android ou OSX, onde a localização das bibliotecas openssl deve ser definida.

Aceita os seguintes valores:

- **oslpNone**: este valor não define nenhum valor de caminho de bibliotecas (é o valor por padrão).
- **oslpDefaultFolder**: este valor define a pasta padrão das bibliotecas openssl. Este caminho é diferente para cada personalidade (windows, osx...).

Load Additional OpenSSL Functions

Utilize um callback para carregar funções adicionais do OpenSSL não definidas por padrão. Você pode ler mais em [OpenSSL Load Additional Functions](#).

Ciphers

Se você quiser fornecer suporte a TLS 1.2 e 1.3 no seu servidor usando a melhor segurança e desempenho, use a seguinte configuração:

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

E defina a seguinte lista de cifras.

```
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
```

Certificados Autoassinados

Você pode utilizar certificados autoassinados para fins de teste. Você só precisa executar o seguinte comando para criar um certificado autoassinado:

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

Isso criará 2 arquivos: cert.pem (certificado) e key.pem (chave privada). Você pode combinar ambos os arquivos em um único. Basta criar um novo arquivo e copiar o conteúdo de ambos os arquivos para ele. Assim, você terá uma estrutura como esta:

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

Erros Comuns

SSL_GET_RECORD: wrong version number

Este erro significa que o servidor e o cliente estão utilizando versões diferentes do protocolo SSL/TLS. Para corrigi-lo, tente definir a versão correta no componente servidor e/ou cliente.

```
Server.SSLOptions.Version  
Client.TLSOptions.Version
```

SSL3_GET_RECORD: decryption failed or bad record mac

Geralmente este erro é gerado quando:

1. Verifique se você está usando a versão mais recente do OpenSSL. Se for muito antiga, atualize para a versão suportada mais recente.
2. Se este erro aparece aleatoriamente, geralmente é porque mais de uma thread está acessando a conexão OpenSSL. Você pode tentar definir `NotifyEvents = neNoSync`, o que significa que os eventos: `OnConnect`, `OnDisconnect`, `OnMessage...` serão disparados no contexto da thread de conexão; isto evita alguns problemas de sincronização e proporciona melhor desempenho. Como desvantagem, se, por exemplo, você estiver atualizando um controle visual em um formulário quando receber uma mensagem, deverá implementar seus próprios métodos de sincronização, porque os controles visuais não são thread-safe.

OpenSSL | Windows

Há uma versão para 32 bits e outra para 64 bits. Você deve copiar essas bibliotecas na mesma pasta onde está sua aplicação ou no path do seu sistema.

Se o seu Sistema Operacional for Windows 32 bits, basta copiar para a pasta System32.

Se o seu Sistema Operacional for Windows 64 bits, copie a versão de 64 bits para a pasta System32 e a versão de 32 bits para a pasta SysWow64.

API 1.0

Requer as seguintes bibliotecas:

- libeay32.dll
- ssleay32.dll

Se o seu Sistema Operacional for Windows 32 bits, basta copiar para a pasta System32.

Se o seu Sistema Operacional for Windows 64 bits, copie a versão de 64 bits para a pasta System32 e a versão de 32 bits para a pasta SysWow64.

Você pode baixar as bibliotecas mais recentes a partir da sua conta (as bibliotecas não têm dependências externas e são assinadas digitalmente).

API 1.1

Requer as seguintes bibliotecas:

Windows 32

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

Windows 64

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

Se o seu Sistema Operacional for Windows 32 bits, basta copiar para a pasta System32.

Se o seu Sistema Operacional for Windows 64 bits, copie a versão de 64 bits para a pasta System32 e a versão de 32 bits para a pasta SysWow64.

Você pode baixar as bibliotecas mais recentes a partir da sua conta (as bibliotecas não têm dependências externas e são assinadas digitalmente).

API 3.*

Requer as seguintes bibliotecas:

Windows 32

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

Windows 64

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

Se o seu Sistema Operacional for Windows 32 bits, basta copiar para a pasta System32.

Se o seu Sistema Operacional for Windows 64 bits, copie a versão de 64 bits para a pasta System32 e a versão de 32 bits para a pasta SysWow64.

Você pode baixar as bibliotecas mais recentes a partir da sua conta (as bibliotecas não têm dependências externas e são assinadas digitalmente).

Se você estiver utilizando um certificado p12, é necessário implantar a biblioteca legacy.dll. Leia mais sobre [Certificados p12 do OpenSSL](#).

OpenSSL | OSX

Versões mais recentes do OSX não incluem as bibliotecas openssl ou são muito antigas, portanto você deve implantá-las com sua aplicação. Implante essas bibliotecas usando os seguintes passos:

- Abra Project/Deployment em seu projeto.
- Adicione as bibliotecas necessárias.
- Defina RemotePath = 'Contents\Macos'.
- Configure o LibPath do openssl para a pasta padrão:
 - Client.TLSOptions.OpenSSL_Options.LibPath = oslpDefaultFolder.
 - Server.SSLOptions.OpenSSL_Options.LibPath = oslpDefaultFolder.

API 1.0

Requer as seguintes bibliotecas:

- libcrypto.dylib
- libssl.dylib

Você pode baixar as bibliotecas mais recentes da sua conta.

API 1.1

Requer as seguintes bibliotecas:

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

Há uma versão para 32 bits e outra para 64 bits. Você deve copiar essas bibliotecas na mesma pasta onde sua aplicação está localizada.

Você pode baixar as bibliotecas mais recentes da sua conta.

API 3.0

Requer as seguintes bibliotecas:

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

Somente a versão de 64 bits é fornecida. Você deve copiar essas bibliotecas na mesma pasta onde sua aplicação está localizada.

Você pode baixar as bibliotecas mais recentes da sua conta.

Se você incluir as bibliotecas OpenSSL em uma aplicação OSX, após a aplicação ter sido Notarizada, as bibliotecas serão assinadas. Você pode verificar isso usando o seguinte comando:

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

Confira o vídeo a seguir, que mostra como compilar uma aplicação MacOSX64 com as bibliotecas OpenSSL.

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

Erros

Os clientes não devem carregar a dylib libcrypto sem versão, pois ela não possui uma ABI estável.

No MacOS Monterey+, você pode obter este erro ao tentar carregar as bibliotecas openssl. O erro ocorre quando tenta carregar primeiro as bibliotecas openssl sem versão (libcrypto.dylib, por exemplo).

Para corrigir este erro, defina na propriedade **OpenSSL_Options.UnixSymLinks** o valor **oslsSymLinksDontLoad**. Isso evita o carregamento das bibliotecas openssl sem versão.

OpenSSL | Android

Versões mais recentes do Android não incluem bibliotecas openssl ou são muito antigas, então você deve implantá-las com sua aplicação. Implante essas bibliotecas usando as seguintes etapas:

- Abra Project/Deployment em seu projeto.
- Adicione as bibliotecas necessárias.
- Defina RemotePath = '.\assets\internal'.
- Configure o LibPath do openssl para a pasta padrão:
 - Client.TLSOptions.OpenSSL_Options.LibPath = oslpDefaultFolder.
 - Server.SSLOptions.OpenSSL_Options.LibPath = oslpDefaultFolder.

API 1.0

Requer as seguintes bibliotecas:

- libcrypto.so
- libssl.so

Você pode baixar as bibliotecas mais recentes da sua conta.

No **Android 64bits**, usar TLS 1.2 pode gerar o seguinte erro:

INT_RSA_VERIFY:bad signature

Este é um erro do OpenSSL que foi corrigido na API 1.1.

Você pode tentar utilizar TLS 1.0 ou TLS 1.1 (se o servidor ainda suportar esses métodos de criptografia para evitar este erro).

API 1.1

Requer as seguintes bibliotecas:

- libcrypto.so
- libssl.so

Você pode baixar as bibliotecas mais recentes da sua conta.

API 3.0

Requer as seguintes bibliotecas:

- libcrypto.so
- libssl.so

Você pode baixar as bibliotecas mais recentes da sua conta.

OpenSSL | iOS

Para instalar o OpenSSL em um dispositivo iOS de 64 bits, você deve copiar os arquivos da biblioteca SSL libcrypto.a e libssl.a para o seu sistema. Baixe o .zip do iOS OpenSSL, extraia-o e encontre os arquivos .a no diretório lib. Você deve copiar os arquivos da biblioteca SSL libcrypto.a e libssl.a para estes diretórios:

- C:\Program Files (x86)\Embarcadero\Studio\<IDE Version>\lib\iosDevice64\debug
- C:\Program Files (x86)\Embarcadero\Studio\<IDE Version>\lib\iosDevice64\release

Adicione a unit **sgcIdSSLOpenSSLHeaders_static** (ou IdSSLOpenSSLHeaders_static se sua edição do sgcWebSockets não for Enterprise) à sua cláusula uses.

Se você precisar **implantar** qualquer arquivo, você pode definir RemotePath = StartUp\Documents e, para carregar o arquivo, utilize (requer adicionar System.IOUtils à cláusula uses):

TPath.GetDocumentsPath + PathDelim + <your filename>

As bibliotecas openssl não devem ser implantadas utilizando o menu Project/Deployment no iOS.

API 1.1

Modifique **IdCompilerDefines.inc** e habilite SGC_OPENSSL_API_1_1 na seção IOS:

```
{ $IFDEF IOS }
{ $DEFINE HAS_getifaddrs }
{ $DEFINE USE_OPENSSL }
{ $IFDEF CPUARM }
// RLebeau: For iOS devices, OpenSSL cannot be used as an external library,
// it must be statically linked into the app. For the iOS simulator, this
// is not true. Users who want to use OpenSSL in iOS device apps will need
// para adicionar a biblioteca estática OpenSSL ao projeto e então incluir o
// IdSSLOpenSSLHeaders_static unit in their uses clause. It hooks up the
// funções vinculadas estaticamente para a unit IdSSLOpenSSLHeaders utilizar...
{ $DEFINE STATICLOAD_OPENSSL }
// sgc--> habilite para openssl API 1.1
{ $DEFINE SGC_OPENSSL_API_1_1 }
{ $ENDIF }
{ $ENDIF }
```

Você pode baixar as bibliotecas a partir da sua conta.

API 3.0

Modifique **IdCompilerDefines.inc** e habilite SGC_OPENSSL_API_1_1 e SGC_OPENSSL_API_3_0 na seção IOS:

```
{ $IFDEF IOS }
{ $DEFINE HAS_getifaddrs }
{ $DEFINE USE_OPENSSL }
{ $IFDEF CPUARM }
// RLebeau: For iOS devices, OpenSSL cannot be used as an external library,
// it must be statically linked into the app. For the iOS simulator, this
// is not true. Users who want to use OpenSSL in iOS device apps will need
// para adicionar a biblioteca estática OpenSSL ao projeto e então incluir o
// IdSSLOpenSSLHeaders_static unit in their uses clause. It hooks up the
// funções vinculadas estaticamente para a unit IdSSLOpenSSLHeaders utilizar...
{ $DEFINE STATICLOAD_OPENSSL }
```

```
// sgc--> habilite para openssl API 1.1
{$DEFINE SGC_OPENSSL_API_1_1}
// sgc--> enable for openssl API 3.0
{$DEFINE SGC_OPENSSL_API_3_0}
{$ENDIF}
{$ENDIF}
```

Você pode baixar as bibliotecas a partir da sua conta.

OpenSSL | Own CA Certificates

Post do Github

Para criar um certificado assinado pela sua própria CA e que possa ser confiável pelos Navegadores Web (como o Chrome) após adicionar o certificado da CA à máquina local.

1. Prepare os arquivos de configuração para criar certificados sem prompts

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. Gerar uma chave privada de CA e um Certificado (válido por 5 anos)

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

3. Gerar a chave secreta do servidor web e o CSR

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

4. Criar o certificado e assiná-lo pela própria autoridade certificadora (válido por 1 ano)

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

5. Os arquivos de saída serão:

- `CA.cnf` → arquivo de configuração da CA OpenSSL. Pode ser excluído após o processo de criação do certificado.
- `CA_cert.pem` → certificado da [Certificate Authority]. Este certificado deve ser adicionado ao armazenamento de autoridade local do navegador para tornar confiáveis todos os certificados criados utilizando esta CA.
- `CA_cert.srl` → Número de série aleatório. Pode ser excluído após o processo de criação do certificado.
- `CA_key.pem` → Deve ser utilizado ao criar um novo certificado [localhost]. Pode ser excluído após o processo de criação do certificado (se você não planeja reutilizá-lo, junto com `CA_cert.pem`).
- `localhost.cnf` → arquivo de configuração do certificado SSL do OpenSSL. Pode ser excluído após o processo de criação do certificado.
- `localhost.csr` → Certificate Signing Request. Pode ser excluído após o processo de criação do certificado.
- `localhost_cert.pem` → Certificado SSL. **Deve ser configurado na propriedade `SSLOptions.CertFile` do servidor.**
- `localhost_key.pem` → Chave secreta. **Deve ser instalada na propriedade `SSLOptions.KeyFile` do servidor.**

OpenSSL | P12 Certificates

O OpenSSL 3.0 moveu vários algoritmos obsoletos ou inseguros para um módulo de biblioteca interno chamado legacy provider. Ele não é carregado por padrão, então as aplicações (ou seus runtimes de linguagem) que usam o OpenSSL para operações criptográficas não podem usar tais algoritmos ao carregar certificados, criar message digests...

Os algoritmos no legacy provider incluem MD2, MD4, MDC2, RMD160, CAST5, BF (Blowfish), IDEA, SEED, RC2, RC4, RC5 e DES (mas não 3DES).

Por razões de segurança, é altamente recomendável descontinuar o uso desses algoritmos legados.

Se a sua aplicação utiliza certificados de cliente armazenados em um arquivo criptografado com uma cifra legada como RC2-40-CBC, é possível "modernizar" o arquivo de certificado recriptografando-o com o programa openssl.

Por exemplo, se você tiver um arquivo de certificado client.p12 (ou client.pfx) em seu computador local:

```
$ openssl pkcs12 -legacy -in client.p12 -nodes -out cert-decrypted.tmp
(digite as passphrases se solicitado)
```

```
$ openssl pkcs12 -in cert-decrypted.tmp -export -out client-new.p12
(enter passphrases if prompted)
```

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

O arquivo de certificado client-new.p12 exportado agora contém as mesmas chaves, mas criptografadas usando AES-256-CBC.

Confira abaixo a configuração para os pacotes sgcWebSockets e sgclndy:

sgcWebSockets

- Defina a propriedade **OpenSSL_Options.Legacy.Enabled** como True.
- Defina a localização da biblioteca Legacy.
 - **OpenSSL_Options.Legacy.LibPath**: aqui você pode configurar onde está localizada a biblioteca legacy
 - **oslpNone**: este é o padrão, a biblioteca legacy deve estar na mesma pasta do binário ou em um caminho conhecido.
 - **oslpDefaultFolder**: define automaticamente o caminho da biblioteca legacy onde as bibliotecas devem estar localizadas para todas as personalidades da IDE.
 - **oslpCustomFolder**: se esta for a opção selecionada, defina o caminho completo na propriedade LibPathCustom.
 - **OpenSSL_Options.Legacy.LibPathCustom**: quando LibPath = oslpCustomFolder, defina aqui o caminho completo onde está localizada a biblioteca legacy.

sgclndy

- Defina a propriedade **SSLOptions.Legacy** como True.
- Antes de iniciar o servidor ou o cliente, defina o caminho onde a biblioteca legacy.dll está localizada. Utilize a função **IdOpenSSLSetOSSLPPath** e passe o caminho como argumento.

OpenSSL | Verify Certificate

Ao utilizar o OpenSSL e definir a opção Verify Certificate, o seguinte erro pode aparecer:

Error connecting with SSL.error:80000002:system library::No such file or directory.
Se você tratar o evento OnVerifyPeer e o parâmetro Error tiver um valor de 20, o erro significa:

X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY

A principal razão para este erro é que um ou mais certificados apresentados pelo servidor remoto não estão presentes no certificate store da sua aplicação. Para resolver isto, você pode utilizar a propriedade RootCertFile e definir o caminho onde o arquivo CA está localizado. Se você não tiver nenhum, pode baixar da mozilla, por exemplo:

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

Após definir o RootCertFile, o erro anterior deve ter desaparecido.

OpenSSL | Load Additional Functions

Por padrão, o Indy define as funções OpenSSL mais comuns necessárias para criptografar comunicações, mas às vezes você precisa de mais funções para criptografia, assinatura, etc. Você pode usar o método `IdOpenSSLSetLoadFuncsCallback` para atribuir um callback que carrega funções OpenSSL adicionais dinamicamente.

IdOpenSSLSetLoadFuncsCallback

```
TIdLoadSSLFuncsCallback = procedure(TIdLibHandle hIdSSL, TIdLibHandle hIdCrypto, const TStringList* FailedLoadLibs)
```

Este é um tipo de procedimento que serve como callback, recebendo três parâmetros:

- **hIdSSL:** `TIdLibHandle` - Handle para a biblioteca SSL carregada.
- **hIdCrypto:** `TIdLibHandle` - Handle para a biblioteca Crypto carregada.
- **FailedLoadList:** `TStringList` - Uma lista de funções que não puderam ser carregadas.

A finalidade deste callback é permitir que o usuário realize processamento personalizado quando as funções OpenSSL estão sendo carregadas, como registrar em log falhas de carregamento de funções ou tratar erros.

IdOpenSSLSetUnLoadFuncsCallback

```
TIdLoadSSLFuncsCallback = procedure(TIdLibHandle hIdSSL, TIdLibHandle hIdCrypto, const TStringList* FailedLoadLibs)
```

Serve como callback para o descarregamento de funções SSL. Isso é útil para realizar limpeza quando as bibliotecas OpenSSL estão sendo descarregadas.

Como carregar uma função personalizada

Encontre abaixo um exemplo simples de como carregar a função `EVP_PKEY_CTX_set_rsa_padding` utilizando os callbacks.

```
typedef int (__cdecl *TEVP_PKEY_CTX_set_rsa_padding)(void* ctx, int pad);
TEVP_PKEY_CTX_set_rsa_padding EVP_PKEY_CTX_set_rsa_padding = nullptr;

// Mock function for LoadLibFunction (you need to implement this according to your needs)
void* LoadLibFunction(TIdLibHandle hLib, const char* funcName) {
    return GetProcAddress((HMODULE)hLib, funcName);
}

void __fastcall DoOpenSSLLoadFuncsCallback(TIdLibHandle hIdSSL, TIdLibHandle hIdCrypto, const TStringList* FailedLoadLibs) {
    EVP_PKEY_CTX_set_rsa_padding = (TEVP_PKEY_CTX_set_rsa_padding)LoadLibFunction(hIdCrypto, "EVP_PKEY_CTX_set_rsa_padding");
}

void __fastcall DoOpenSSLUnLoadFuncsCallback() {
    EVP_PKEY_CTX_set_rsa_padding = nullptr;
}

// Mock function to simulate IdOpenSSLSetLoadFuncsCallback
void IdOpenSSLSetLoadFuncsCallback(void (__fastcall *callback)(TIdLibHandle, TIdLibHandle, const TStringList*)) {
    callback(nullptr, nullptr, nullptr); // Example call
}

// Mock function to simulate IdOpenSSLSetUnLoadFuncsCallback
void IdOpenSSLSetUnLoadFuncsCallback(void (__fastcall *callback)()) {
    callback(); // Example call
}
```

```
int main() {  
    IdOpenSSLSetLoadFuncsCallback(DoOpenSSLLoadFuncsCallback);  
    IdOpenSSLSetUnLoadFuncsCallback(DoOpenSSLUnLoadFuncsCallback);  
    return 0;  
}
```


Indy

A biblioteca Indy é uma biblioteca de comunicações cliente/servidor de código aberto que suporta sockets TCP/UDP/RAW, bem como mais de 100 protocolos de nível mais alto, incluindo SMTP, POP3, IMAP, NNTP, HTTP, FTP e muitos mais. Indy é escrita em Delphi, mas também está disponível para C++Builder e FreePascal. O sgcWebSockets usa Indy como base para alguns componentes e as diferentes Edições do sgcWebSockets fazem um uso diferente da biblioteca Indy.

A sgcWebSockets suporta protocolos como HTTP/2 que exigem o uso de ALPN, pode usar TLS 1.3 usando openssl 1.1.1 ou openssl 3.0.0... todos esses recursos não são suportados pela biblioteca padrão do Indy, portanto a sgcWebSockets Enterprise edition inclui uma biblioteca Indy personalizada que suporta esses recursos. Para evitar desinstalar a biblioteca padrão do Indy do IDE, os arquivos Indy necessários são renomeados adicionando o prefixo "sgc"; assim, por exemplo: a unit "IdGlobal" é renomeada para "sgcIdGlobal". Dessa forma, ambas as versões podem coexistir sem problemas.

Encontre abaixo qual versão do Indy é usada por cada Edição do sgcWebSockets:

sgcWebSockets Edition	Versão do Indy
STANDARD	Standard
PROFESSIONAL	Standard
ENTERPRISE	Custom

Os clientes com licenças "Registered", que são licenças antigas anteriores à divisão do pacote sgcWebSockets, encontrarão as seguintes versões do pacote sgcWebSockets:

sgcWebSockets Edition	Versão do Indy
sgcWebSockets	Standard
sgcWebSockets min	Standard
sgcWebSockets min Indy*	Custom

*O sgcWebSockets_min_indy é o mesmo que a edição sgcWebSockets Enterprise.

O uso da versão personalizada do Indy é definido no arquivo "sgcVer.inc" localizado na pasta de origem. Há um define de compilador chamado "SGC_CUSTOM_INDY" que habilita ou desabilita o uso desta versão do indy. Se você tem uma Enterprise Edition e quer desabilitar o uso do indy personalizado, basta excluir o seguinte define de compilador:

```
{ $DEFINE SGC_CUSTOM_INDY }
```

Claro, se você habilitar SGC_CUSTOM_INDY mas não tiver na pasta de código-fonte as units da versão custom do indy necessárias, este define do compilador não funcionará.

pacote sgclndy

O uso da versão personalizada do Indy não está limitado aos componentes sgcWebSockets. Alguns clientes querem fazer uso dos novos recursos desta versão personalizada do Indy em componentes Indy padrão, como SMTP por exemplo, para que possam utilizar TLS 1.3 ao enviar e-mails, utilizando servidores FTP, etc. A edição eSeGe-Ce ALL-ACCESS fornece um pacote Indy completo adicional com todos esses recursos. Este pacote, chamado "sgclndy package", inclui a biblioteca Indy completa com suporte para openssl 1.1.1 e openssl 3.0.0. Portanto, você primeiro deve desinstalar sua biblioteca Indy atual instalada em seu IDE e então instalar esta versão; o processo para instalar o pacote sgclndy é exatamente o mesmo de qualquer biblioteca Indy (aqui as units não são renomeadas).

Quando você quiser utilizar as bibliotecas openssl, basta definir a variável global `OPENSSL_API_VERSION` com a versão da API OpenSSL desejada antes de carregar as bibliotecas OpenSSL. Esta variável global está na unit `IdSSLOpenSSLHeaders`.

Exemplo: para usar as bibliotecas openssl 1.1.1

```
OPENSSL_API_VERSION := openssl_1_1;
```

Como usar um único pacote sgclndy

Ao utilizar o eSeGeCe ALL-ACCESS e o pacote sgclndy na mesma aplicação, as fontes podem ser duplicadas porque a versão sgcWebSockets Enterprise utiliza uma versão personalizada do Indy com as units do Indy renomeadas. Isso significa que, por exemplo, units como sgclGlobal.pas e IdGlobal.pas serão compiladas na mesma aplicação (a primeira é utilizada ao usar qualquer componente do pacote sgcWebSockets Enterprise e a segunda ao usar qualquer componente do pacote sgclndy, como ftp, smtp...).

Para evitar esse comportamento, o pacote sgcWebSockets pode ser configurado para usar a versão instalada do sgclndy e ainda fazer uso de todos os componentes. Para isso, siga as instruções abaixo:

1. Abra o arquivo sgcVer.inc, localizado na pasta Source do pacote sgcWebSockets.
2. Desabilite a diretiva de compilador: `SGC_CUSTOM_INDY`. Esta opção informa ao compilador que os arquivos que começam com sgcl*.pas existem e devem ser utilizados ao compilar o sgcWebSockets Enterprise Package.
3. Habilite o seguinte compilador: `SGC_INDY_LIB`. Esta opção informa ao compilador que o pacote sgclndy está instalado e deve ser usado ao compilar o pacote.

```
{ $DEFINE SGC_INDY_LIB }
```

Utilizando a configuração anterior, o pacote sgcWebSockets Enterprise utilizará o pacote sgclndy que está instalado e todos os recursos que fazem uso deste pacote (como http/2, IOCP, openssl 3.0...) serão habilitados.

WebSocket Events

As conexões WebSocket têm os seguintes eventos:

OnConnect

O evento gerado quando uma nova conexão é estabelecida.

OnDisconnect

O evento gerado quando uma conexão é fechada.

OnError

O evento gerado quando uma conexão tem algum erro.

OnMessage

O evento gerado quando uma nova mensagem de texto é recebida.

OnBinary

O evento gerado quando uma nova mensagem binária é recebida.

Por padrão, o `sgcWebSockets` utiliza um mecanismo **assíncrono** para gerar esses eventos; quando qualquer um desses eventos é gerado internamente, ele enfileira esta mensagem e o sistema operacional a despacha quando permitido. Esse comportamento pode ser modificado utilizando uma propriedade chamada **NotifyEvents**; por padrão, **neAsynchronous** está selecionado; se **neNoSync** estiver marcado, então os eventos serão gerados sem sincronizar com a thread principal (se você precisar atualizar qualquer controle VCL ou acessar recursos compartilhados, então precisará implementar seu próprio método de sincronização).

neNoSync é recomendado quando:

1. Você precisa tratar muitas mensagens em um período de tempo muito curto.
2. Seu projeto é compilado para linha de comando (se você não definir `neNoSync`, não receberá nenhum evento).
3. Seu projeto é uma biblioteca.

Caso contrário, você pode utilizar o valor padrão da propriedade **neAsynchronous**.

WebSocket Parameters Connection

Suportado por

[TsgcWebSocketClient](#)

Java script

Às vezes é útil passar parâmetros do cliente para o servidor quando uma nova conexão WebSocket é estabelecida. Se você precisar passar alguns parâmetros ao servidor, pode utilizar a seguinte propriedade:

Opções / Parâmetros

Por padrão, é definido como '/'. Se você precisar passar um parâmetro como id=1, pode definir esta propriedade como '/?id=1'

No lado do Servidor, você pode tratar os parâmetros do cliente utilizando o seguinte parâmetro:

```
void WSServerConnect(TsgcWSConnection *Connection);
{
    if (Connection->URL == "/?id=1")
    {
        HandleThisParameter;
    }
}
```

Utilizando Javascript, você pode passar parâmetros utilizando a url de conexão, exemplo:

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

Usando dentro de uma DLL

Se você precisar trabalhar com Dynamic Link Libraries (DLL) e `sgcWebSockets` (ou aplicações de console), a propriedade **NotifyEvents** precisa ser definida como **neNoSync**.

Windows Service

Ao executar a biblioteca `sgcWebSockets` em um serviço do Windows, a propriedade **NotifyEvents** deve ser definida como **neNoSync**.

```
void __fastcall TService1::ServiceStart(TService *Sender, bool &Started)
{
    FServer = new TsgcWebsocketHTTPServer(nullptr);
    FServer->Port = 80;
    FServer->Active = true;
}
//-----
// OnServiceExecute
void __fastcall TService1::ServiceExecute(TService *Sender)
{
    while (!Terminated)
    {
        ServiceThread->ProcessRequests(true);
    }
}
//-----
// OnServiceStop
void __fastcall TService1::ServiceStop(TService *Sender, bool &Stopped)
{
    if (FServer != nullptr)
    {
        delete FServer;
        FServer = nullptr;
    }
}
```

WebBrowser Test

O TsgcWebSocketServer implementa uma página Web integrada onde você pode testar a conexão do WebSocket Server com o seu Navegador favorito.

Para acessar esta página de teste, você precisa digitar a seguinte URL:

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

Exemplo: se você configurou seu Servidor WebSocket no IP 127.0.0.1 e utiliza a porta 80, então você precisa digitar:

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

Nesta página, você pode testar os seguintes métodos WebSocket:

Open
Close
Status
Send

Para desabilitar as páginas de Teste HTML do WebBrowser, basta definir em `TsgcWebSocketServer.Options.HTMLFiles = false;`

Subprotocolos Personalizados

Um cliente pode solicitar que o servidor utilize um subprotocolo específico incluindo o nome do subprotocolo em seu handshake. Se for especificado, o servidor precisa incluir um dos valores de subprotocolo selecionados em sua resposta para que a conexão seja estabelecida.

Para criar seu próprio subprotocolo, você deve herdar de `TsgcWSProtocol_Client_Base` e `TsgcWSProtocol_Server_Base` para criar seus subprotocolos personalizados.



Autenticação

Suportado por

TsgcWebSocketServer
TsgcWebSocketHTTPServer
TsgcWebSocketClient

Java script (*apenas a Autenticação por URL é suportada)

A especificação WebSocket não define nenhum método de autenticação, e as implementações em navegadores web não permitem enviar cabeçalhos personalizados em novas conexões WebSocket.

Para habilitar este recurso, você precisa acessar a seguinte propriedade:

Authentication/ Enabled

o sgcWebSockets implementa 3 tipos diferentes de autenticação WebSocket:

Session: o cliente precisa fazer um HTTP GET passando um nome de usuário e senha e, se autenticado, o servidor responde com um Session ID. Com este Session ID, o cliente abre uma conexão WebSocket passando-o como parâmetro. Você pode usar uma requisição HTTP normal para obter um session id passando usuário e senha como parâmetros

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

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

Isso retorna um token que é utilizado para conectar-se ao servidor utilizando conexões WebSocket:

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

URL: o cliente abre uma conexão WebSocket passando o nome de usuário e a senha como parâmetros.

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

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

Basic: implementa a Basic Access Authentication, aplica-se apenas a Websockets VCL (Servidor e Cliente) e Requisições HTTP (os Navegadores Web clientes não implementam este tipo de autenticação). Quando um cliente tenta conectar, ele envia um header usando a especificação AUTH BASIC.

Você pode definir uma lista de usuários Autenticados, usando a propriedade **Authentication/ AuthUsers**. Você precisa definir cada item seguindo este esquema: user=password. Exemplo:

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

Existe um evento chamado **OnAuthentication** onde você pode tratar a autenticação caso o usuário não esteja na lista AuthUsers, o cliente não envia uma requisição de autorização... Você pode verificar os parâmetros User e Password e, se estiverem corretos, definir a variável Authenticated como True. exemplo:

```
void WSServerAuthentication(TsgcWSConnection *Connection, string aUser, string aPassword, bool &Authenticated)
{
    if ((aUser == "John") && (aPassword == "1234"))
```

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

Conexões Seguras

Suportado por

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

O suporte a SSL é baseado na implementação do Indy, portanto você precisa implantar as bibliotecas openssl para utilizar este recurso.

O TsgcWebSocketClient suporta o Microsoft SChannel, então não há necessidade de implantar as bibliotecas openssl para windows 32 e 64 bits se a opção SChannel for selecionada no WebSocket Client.

Server Side

Para habilitar este recurso, você precisa habilitar a seguinte propriedade:

SSL/ Enable

Há outras propriedades que você precisa definir:

SSLOptions/ CertFile/ KeyFile/ RootCertFile: você precisa de um certificado no formato .PEM para criptografar as comunicações websocket.

SSLOptions/ Password: isto é opcional e só é necessário se o certificado tiver uma senha.

SSLOptions/ Port: porta usada em conexões SSL.

Lado do Cliente

Para habilitar este recurso, você precisa habilitar a seguinte propriedade:

TLS/ Enable

OpenSSL

Por padrão, os componentes de cliente e servidor baseados em Indy utilizam as bibliotecas openssl ao conectar a servidores websocket seguros.

O Indy suporta apenas a API OpenSSL 1.0.2, então a API 1.1 não é suportada. Se você compilar o sgcWebSockets com nossa biblioteca Indy personalizada, você pode fazer uso da API 1.1 e selecionar a versão TLS 1.3. Basta selecionar nas propriedades OpenSSL_Options qual API OpenSSL você gostaria de usar:

- **osIAPI_1_0:** esta é a API Indy padrão, você pode usar o pacote Indy padrão com as bibliotecas openssl 1.0.2.
- **osIAPI_1_1:** selecione apenas se você estiver compilando o sgcWebSockets com nossa biblioteca Indy personalizada (Enterprise Edition). Usará as bibliotecas openssl 1.1.1.
- **osIAPI_3_0:** selecione apenas se você estiver compilando o sgcWebSockets com nossa biblioteca Indy personalizada (Enterprise Edition). Utilizará as bibliotecas openssl 3.0.0.
 - **ECDHE:** permite que você habilite o ECDHE para TLS 1.2 (conexões mais seguras).

Eventos

Há 2 eventos que podem ser usados para personalizar suas configurações SSL:

OnSSLGetHandler

Este evento é gerado antes do handler SSL ser criado; você pode criar aqui seu próprio SSL Handler (precisa ser herdado de `TIdServerIOHandlerSSLBase` ou `TIdIOHandlerSSLBase`) e definir as propriedades necessárias

```
properties needed
void OnServerSSLGetHandler(TObject *Sender, TwSSLHandler aType, ref TIdServerIOHandlerSSLBase
*aSSLHandler)
{
    TCustomSSLHandler aSSLHandler = new TCustomSSLHandler();
    ...
}
```

OnSSLAfterCreateHandler

Se nenhum objeto SSL personalizado tiver sido criado, um é criado por padrão usando o handler OpenSSL. Você pode acessar as propriedades do SSL Handler e modificá-las se necessário.

```
void OnSSLAfterCreateHandler(TObject *Sender, TwSSLHandler aType, TIdServerIOHandlerSSLBase
*aSSLHandler)
{
    dynamic_cast <tidserveriohandlesslopenssl>*(aSSLHandler)->SSLOptions->Method = sslvTLSv1_2;
}
```

Microsoft SChannel

A partir do `sgcWebSockets` 4.2.6 você pode usar SChannel em vez do openssl (apenas para windows a partir do Windows 7+). Isso significa que não há necessidade de implantar as bibliotecas openssl. O TLS 1.0 é suportado a partir do windows 7, mas se você precisar de implementações mais modernas como TLS 1.2 no Windows 7, deve habilitar TLS 1.1 e TLS 1.2 no Registro do Windows.

Requer a Edição Profissional do Delphi 2010 (ou a Edição Enterprise para Delphi 7, 2007 e 2009).

HeartBeat

Suportado por

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

Em componentes de Servidor, envia automaticamente um ping a todas as conexões WebSocket ativas a cada x segundos.

Nos componentes Client, envia automaticamente um ping ao servidor a cada x segundos.

HeartBeat possui as seguintes propriedades:

- **Enabled:** se true, envia um ping
- **Interval:** é o valor em segundos em que um ping será enviado. Exemplo: se o valor for 10, um ping será enviado a cada 10 segundos
- **Timeout:** é o tempo que ele aguardará por uma resposta do servidor. Exemplo: se o valor for 30, ele aguardará 30 segundos para receber uma resposta antes de fechar a conexão.
- **HeartBeatType:** permite personalizar como o HeartBeat funciona
 - **hbtAlways:** envia um ping a cada x segundos definidos no Interval.
 - **hbtOnlyIfNoMsgRcvInterval:** envia um ping a cada x segundos somente se nenhuma mensagem tiver sido recebida durante os últimos x segundos definidos na propriedade Interval.

Personalizar HeartBeat

Os componentes cliente e servidor permitem a personalização do HeartBeat para enviar pings personalizados e verificar se a conexão ainda está ativa. O evento **OnBeforeHeartBeat** foi criado exatamente para isso; ele permite que você envie uma mensagem personalizada e/ou não envie o ping padrão.

Exemplo: enviar um texto de mensagem como ping a cada 30 segundos.

```
void OnBeforeHeartBeat(TObject *Sender; const TsgcWSConnection *Connection; ref bool Handled)
{
    Connection->WriteData("ping");
    Handled = true;
}
```

WatchDog

Suportado por

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

Server

Em componentes de Servidor, reinicia automaticamente o servidor após um desligamento inesperado. Para verificar se o servidor está ativo a cada 60 segundos, basta definir as seguintes propriedades:

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

WatchDog.Monitor permite verificar se novos clientes conseguem conectar ao servidor. Isso é feito por um cliente interno que tenta abrir uma conexão WebSocket ao servidor; se falhar, ele reinicia o servidor. Para monitorar se os clientes conseguem conectar ao servidor com um time-out de 10 segundos, defina as seguintes propriedades:

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

Client

Em componentes Client, reconecta automaticamente ao servidor após uma desconexão inesperada. Para reconectar após uma desconexão a cada 10 segundos, basta definir as seguintes propriedades:

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

Logs

Suportado por

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

Este é um recurso útil que permite depurar conexões WebSocket. Para habilitá-lo, você precisa acessar a seguinte propriedade:

LogFile/ Enabled

Uma vez habilitado, toda vez que uma nova conexão é estabelecida, ela será registrada em um arquivo de texto. No componente de servidor, se o arquivo não existir, ele será criado, mas você não pode acessá-lo até que o servidor seja fechado. Se você quiser abrir o arquivo de log enquanto o servidor está ativo, o arquivo de log precisa ser criado antes de iniciar o servidor.

Exemplo:

```
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.
```

WebSocket Messages

Os frames WebSocket podem ser mascarados, o que significa que a mensagem registrada não pode ser lida. Quando a propriedade **LogFile.UnMaskFrames** = True (por padrão é true)

- As mensagens **enviadas** pelo **WebSocket Client** são salvas como **não mascaradas**.
- As mensagens **recebidas** pelo **Servidor WebSocket** são salvas **mascaradas** e **desmascaradas** (o motivo é que, quando o socket lê o buffer, ele ainda não sabe o protocolo da mensagem, então salva ambas).

HTTP

Suportado por

[TsgcWebSocketHTTPServer](#)

TsgcWebSocketHTTPServer é um componente que permite que você trate conexões WebSocket e HTTP utilizando a mesma porta. É muito útil quando você precisa configurar um servidor onde somente a porta HTTP está habilitada (normalmente a porta 80). Este componente suporta todos os recursos do [TsgcWebSocketServer](#) e permite que você sirva páginas HTML.

Você pode **servir páginas HTML estaticamente**, utilizando a propriedade **DocumentRoot**, exemplo: se você salvar test.html no diretório "C:\inetpub\wwwroot", e definir **DocumentRoot** como "C:\inetpub\wwwroot". Se um cliente tentar acessar test.html, ele será servido automaticamente, exemplo:

`http://localhost/test.html`

Ou você pode **servir HTML ou outros recursos dinamicamente** por código. Para fazer isso, há um evento chamado **OnCommandGet** que é disparado toda vez que um cliente solicita uma nova página HTML, imagem, arquivo javascript... Basicamente, você precisa verificar qual documento o cliente está solicitando (usando `ARequestInfo.Document`) e enviar uma resposta ao cliente (usando `AResponseInfo.ContentText`, onde você envia o conteúdo da resposta, `AResponse.ContentType`, que é o tipo da resposta, e um `AResponseInfo.ResponseNo` com um número de código de resposta, geralmente 200), exemplo:

```
void WSServerCommandGet(TidContext *AContext, TidHTTPRequestInfo *ARequestInfo,
    TidHTTPResponseInfo *AResponseInfo)
{
    if (ARequestInfo->Document == "/myfile.js")
    {
        AResponseInfo->ContentText = "<script type='text/javascript'>alert('Hello!');</script>";
        AResponseInfo->ContentType = "text/javascript";
        AResponseInfo->ResponseNo = 200;
    }
}
```


Broadcast e Canais

Suportado por

[TsgcWebSocketServer](#)

[TsgcWebSocketHTTPServer](#)

[TsgcWebSocketServer_HTTPAPI](#)

O método **Broadcast** por padrão envia uma mensagem para **todos os clientes conectados**, mas você pode utilizar o argumento **channels** para filtrar e **enviar a mensagem somente para os clientes assinantes** de um canal.

Exemplo: seu servidor tem 2 tipos de clientes conectados, dispositivos desktop e móveis, então você pode criar 2 canais "desktop" e "mobile".

Se você puder identificar no evento OnConnect do servidor se um cliente é móvel, você pode fazer algo como o seguinte:

```
void OnServerConnect(TsgcWSConnection *Connection)
{
    if (desktop == true)
    {
        dynamic_cast(Connection)->DoSubscribe("desktop");
    }
}
```

Primeiro converta Connection para TsgcWSConnectionServer para acessar os métodos de assinatura e, se ele corresponder ao seu filtro, será assinado no canal desktop. A assinatura de um canal pode ser feita em qualquer evento. Por exemplo, você pode pedir ao cliente que informe se ele é móvel ou não e enviar uma mensagem do cliente para o servidor com informações sobre o cliente. Então você pode fazer broadcast somente para as conexões desktop:

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

Se você tiver 100 conexões e 30 forem móveis, a mensagem será enviada somente para as outras 70.

Bindings

Suportado por

[TsgcWebSocketServer](#)

[TsgcWebSocketHTTPServer](#)

Normalmente, os servidores têm mais de um IP. Se você habilita um Servidor WebSocket e define a porta de escuta como 80, quando o servidor inicia, ele tenta escutar na porta 80 de TODOS os IPs. Então, se você tiver 3 IPs, ele vinculará a porta 80 em cada um deles.

Bindings permitem definir exatamente quais IP e Porta são usados pelo Servidor. Por exemplo, se você precisa escutar na porta 80 para o IP 127.0.0.1 (endereço interno) e 80.254.21.11 (endereço público), você pode fazer isso antes de o servidor ser ativado:

```
bind = WSServer->Bindings->Add();
{
    bind->Port = 80;
    bind->IP = "127.0.0.1";
}
bind = WSServer->Bindings->Add();
{
    bind->Port = 80;
    bind->IP = "80.254.21.11";
}
```

Enviar Arquivos Grandes por POST

Suportado por

[TsgcWebSocketHTTPServer](#)

[TsgcWebSocketServer_HTTPAPI](#)

Quando um cliente HTTP envia um stream **multipart/form-data**, o stream é salvo pelo servidor na memória. Quando os arquivos são grandes, o servidor pode gerar uma exceção de **falta de memória**. Para evitar essas exceções, o servidor possui uma propriedade chamada **HTTPUploadFiles** na qual você pode configurar como os streams POST são tratados: na memória ou como file streams. Se os streams forem tratados como file streams, os streams recebidos são armazenados diretamente no disco rígido, evitando assim os problemas de memória.

Para configurar seu servidor para salvar streams multipart/form-data como file streams, siga os próximos passos:

1. Defina a propriedade **HTTPUploadFiles.StreamType = pstFileStream**. Com essa configuração, o servidor armazenará esses streams no disco rígido.
2. Você pode configurar qual é o **tamanho mínimo em bytes** a partir do qual os arquivos serão armazenados como file stream. Por padrão o valor é zero, o que significa que todos os streams serão armazenados como file stream.
3. A pasta onde os streams são armazenados é definida por **SaveDirectory**; se não for definida, eles serão armazenados na mesma pasta onde está a aplicação.
4. Quando um cliente envia um multipart/form-data, o conteúdo é codificado dentro de boundaries; se a propriedade **RemoveBoundaries** estiver habilitada, o conteúdo dos boundaries será extraído automaticamente após o stream completo ser recebido.

Código de Exemplo

Primeiro crie uma nova instância de servidor e defina que os Streams sejam salvos como File Streams.

```
TsgcWebSocketHTTPServer *oServer = new TsgcWebSocketHTTPServer();
oServer->Port = 5555;
oServer->HTTPUploadFiles->StreamType = pstFileStream;
oServer->Active = true;
```

Em seguida crie um novo arquivo html com a seguinte configuração

```
<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>
```

Por fim abra o arquivo html com um navegador web e envie um arquivo para o servidor. O servidor criará um novo file stream com a extensão ".sgc_ps" e, quando o stream for totalmente recebido, extrairá o arquivo dos boundaries.

Eventos

Existem 2 eventos que podem ser utilizados para personalizar o fluxo de upload de arquivos (requer que a propriedade **HTTPUploadFiles.RemoveBoundaries** esteja habilitada)

OnHTTPUploadBeforeSaveFile

Este evento é disparado ANTES de o arquivo ser salvo e permite personalizar o nome do arquivo recebido.

```
void OnHTTPUploadBeforeSaveFileEvent(TObject *Sender, String &aFileName, String &aFilePath)
{
    if (aFileName == "test.jpg")
    {
        aFileName = "custom_test.jpg";
    }
}
```

OnHTTPUploadAfterSaveFile

Este evento é disparado DEPOIS de o arquivo ser salvo e permite que você saiba o nome do arquivo salvo.

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

OnHTTPUploadReadInput

Este evento é disparado quando o decodificador lê um valor de entrada recebido diferente da entrada de arquivo (exemplo: se o formulário tiver algumas variáveis como nome, data...).

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

Compressão

Suportado por

[TsgcWebSocketServer](#)

[TsgcWebSocketHTTPServer](#)

[TsgcWebSocketClient](#)

Navegadores web como o Chrome

Este é um recurso que funciona muito bem quando você precisa enviar muitos dados, normalmente usando uma mensagem binária, porque comprime a mensagem WebSocket usando o protocolo "PerMessage_Deflate", que é suportado por alguns navegadores como o Chrome.

Para habilitar este recurso, você precisa ativar a seguinte propriedade:

Extensions/ PerMessage_Deflate / Enabled

Quando um cliente tenta conectar a um servidor WebSocket e esta propriedade está habilitada, ele envia um cabeçalho com esta propriedade habilitada. Se o servidor tiver ativado este recurso, ele envia uma resposta ao cliente com este protocolo ativado e todas as mensagens serão comprimidas. Se o servidor não tiver este recurso, então todas as mensagens serão enviadas sem compressão.

Nos Navegadores Web, você não precisa fazer nada, se esta extensão for suportada, ela será usada automaticamente; se não for, então as mensagens serão enviadas sem compressão.

Se as mensagens WebSocket forem pequenas, é melhor não habilitar esta propriedade, pois ela consome ciclos de CPU para comprimir/descomprimir as mensagens. Mas se você estiver transferindo uma grande quantidade de dados, notará um aumento na velocidade de troca de mensagens.

Flash

Suportado por

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

WebSockets são suportados nativamente por uma ampla gama de navegadores web (verifique <http://caniuse.com/websockets>), mas existem algumas versões antigas que não implementam WebSockets (como o Internet Explorer 6, 7, 8 ou 9). Você pode habilitar o **Flash Fallback** para todos esses navegadores que não implementam WebSockets.

Quase todos os outros navegadores ou navegadores mais antigos suportam Flash instalando o Adobe Flash Player. Para suportar a conexão Flash, você precisa **abrir a porta 843** no seu servidor, porque o Flash utiliza esta porta por motivos de segurança para verificar o acesso entre domínios. Se a porta 843 não estiver acessível, ele aguarda 3 segundos e tenta conectar à porta padrão do servidor.

O Flash só é aplicado se o navegador não suportar WebSockets nativamente. Portanto, se você habilitar o Flash Fallback no lado do servidor, e o navegador web suportar WebSockets nativamente, ele ainda usará WebSockets como transporte.

Para habilitar o Flash Fallback, você precisa acessar a propriedade **FallBack / Flash** no servidor e **habilitá-la**. Há mais 2 propriedades:

1. Domain: se você precisar restringir as conexões flash a um ou múltiplos domínios (por padrão, todos os domínios são permitidos). Exemplo: Isto permitirá acesso ao domínio `swf.example.com`

`swf.example.com`

2. Ports: se você precisar restringir as conexões flash a uma única/múltiplas portas (por padrão, todas as portas são permitidas). Exemplo: Isto permitirá o acesso às portas 123, 456, 457 e 458

`123,456-458`

Conexões Flash suportam apenas mensagens de Texto; mensagens binárias não são suportadas.

Custom Objects

Suportado por

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

Toda vez que uma nova conexão WebSocket é estabelecida, o sgcWebSockets cria uma classe [TsgcWSConnection](#) onde você pode acessar algumas propriedades como o identificador, bytes recebidos/enviados, IP do cliente... e há uma propriedade chamada **Data** onde você pode armazenar objetos em memória, como acesso a banco de dados, objetos de sessão...

```
// You can create a new class called MyClass and create some properties, example:
class TMyClass
{
private:
    bool FRegistered;
    String FUser;
public:
    __property bool Registered = {read=FRegistered, write=FRegistered};
    __property String User = {read=FUser, write=FUser};
};

// Then, when a new client connects, OnConnect Event, create a new TMyClass and Assign to Data:
void WSServerConnect(TsgcWSConnection *Connection)
{
    Connection->Data = new TMyClass();
}

// Every time a new message is received by the server, you can access your custom object using
// Connection.Data property.
void WSServerMessage(TsgcWSConnection *Connection, string Text)
{
    if (dynamic_cast<TMyClass*>(Connection->Data)->Registered == true)
    {
        DoSomeStuff();
    }
}

// When a connection is closed, you may free your object:
void TfrmServerChat::WSServerDisconnect(TsgcWSConnection *Connection, int Code)
{
    TMyClass *oMyClass = dynamic_cast<TMyClass*>(Connection->Data);
    if (oMyClass != 0)
    {
        delete oMyClass;
        Connection->Data = NULL;
    }
}
```

Groups

Suportado por

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

O **sgcWebSockets** fornece um método poderoso para **transmitir mensagens para subconjuntos específicos de clientes conectados**. Um grupo pode ter qualquer número de clientes, e um cliente pode ser membro de qualquer número de grupos. Você não precisa criar grupos explicitamente. Na prática, um grupo é criado automaticamente na primeira vez que você especifica seu nome em uma chamada a Groups.Add.

Quando você adiciona um usuário a um grupo utilizando o método **Groups.Add**, o usuário recebe mensagens direcionadas àquele grupo durante a conexão atual.

Adicionar e remover usuários

Para adicionar ou remover usuários de um grupo, você chama os métodos Add ou Remove e passa o Nome do Grupo e a classe TsgcWSCConnection. Você não precisa remover manualmente um usuário de um grupo quando a conexão termina.

O exemplo a seguir mostra o método Groups.Add.

```
void OnConnect(TsgcWSCConnection *Connection)
{
    TsgcWebSocketServer1->Groups->Add("Room1", Connection);
}
```

Enviando Mensagens para um Grupo

Você pode enviar uma mensagem a todos os membros de um grupo conforme mostrado no exemplo a seguir.

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

Ou você pode enviar uma mensagem a todos os grupos que começam com "Room" (então se existirem Room1, Room2, Room3... esses usuários receberão uma mensagem).

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

Eventos

Há 2 eventos que podem ser utilizados para tratar os Grupos e os Clientes toda vez que um novo cliente é adicionado a um grupo ou quando um é removido:

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

Exemplo: envie uma mensagem ao grupo quando um membro sair do grupo.

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

IOCP

Suportado por

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

*Requer versão personalizada do Indy.

O IOCP para Windows é uma API que permite tratar milhares de conexões utilizando um pool limitado de threads em vez de utilizar uma thread por conexão, como o Indy faz por padrão.

Para habilitar IOCP para servidores Indy, vá até a propriedade **IOHandlerOptions** e selecione **iohIOCP** como IOHandler Type.

```
Server->IOHandlerOptions->IOHandlerType = iohIOCP;  
Server->IOHandlerOptions->IOCP->IOCPThreads = 0;  
Server->IOHandlerOptions->IOCP->WorkOpThreads = 0;
```

IOCPThreads são as threads utilizadas para requisições assíncronas IOCP (operações sobrepostas/overlapped); por padrão o valor é zero, o que significa que o número de threads é calculado utilizando o número de processadores (exceto para Delphi 7 e 2007, onde o número de threads é definido como 32 porque a função cpucount não é suportada).

WorkOpThreads só deve ser habilitado se você quiser que as conexões sejam sempre processadas na mesma thread. Ao usar IOCP, as requisições são processadas por um pool de threads, e cada requisição (para a mesma conexão) pode ser processada em threads diferentes. Se você quiser tratar cada conexão na mesma thread, defina em WorkOpThreads o número de threads usadas para tratar essas requisições. Isso impacta o desempenho do servidor e só é recomendável definir um valor maior que zero se você precisar deste recurso.

Habilitar o IOCP para servidores windows é recomendado quando você precisa tratar milhares de conexões. Se o seu servidor só estiver tratando no máximo 100 conexões simultâneas, você pode permanecer com o modelo de threads padrão do Indy.

Evento OnDisconnect não disparado

O IOCP funciona de forma diferente do IOHandler padrão do Indy. Com o IOHandler padrão do Indy, cada conexão é executada em uma thread e essas threads estão sempre em execução, verificando se a conexão está ativa, de modo que, se houver uma desconexão, ela é notificada em um curto período de tempo.

O IOCP funciona de forma diferente, há um pool de threads que trata todas as conexões, em vez de 1 thread = 1 conexão, como o Indy faz por padrão. Para o IOCP, a única maneira de detectar se uma conexão ainda está ativa é tentar escrever no socket. Se houver algum erro, significa que a conexão está fechada. Há 2 opções para detectar desconexões:

1. Se você usar **TsgcWebSocketClient**, pode habilitá-lo na propriedade Options, **CleanDisconnect := True** (por padrão está desabilitado). Se estiver habilitado, antes de o cliente desconectar, ele envia uma mensagem informando o servidor sobre a desconexão, então o servidor receberá essa mensagem e o evento OnDisconnect será gerado.
2. Você pode habilitar o **heartbeat** no lado do **servidor**, por exemplo a cada 60 segundos, de modo que ele tentará enviar um ping a todos os clientes conectados e, se houver algum cliente desconectado, OnDisconnect será chamado.

EPOLL

Suportado por

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

***Requer a Edição Enterprise do sgcWebSockets.**

EPOLL para Linux é uma API que permite tratar milhares de conexões usando um pool limitado de threads em vez de usar uma thread por conexão, como o Indy faz por padrão.

Para habilitar EPOLL para servidores Indy, vá até a propriedade **IOHandlerOptions** e selecione **iohIEPOLL** como IOHandler Type.

```
Server->IOHandlerOptions->IOHandlerType = iohEPOLL;  
Server->IOHandlerOptions->EPOLL->EPOLLThreads = 0;  
Server->IOHandlerOptions->EPOLL->WorkOpThreads = 0;
```

EPOLLThreads são as threads utilizadas para requisições assíncronas EPOLL (operações sobrepostas/overlap-ped); por padrão o valor é zero, o que significa que o número de threads é calculado utilizando o número de processadores (exceto para Delphi 7 e 2007, onde o número de threads é definido como 32 porque a função `cpu-count` não é suportada). Você pode ajustar o número de threads manualmente.

WorkOpThreads só deve ser habilitado se você quiser que as conexões sejam sempre processadas na mesma thread. Ao usar EPOLL, as requisições são processadas por um pool de threads, e cada requisição (para a mesma conexão) pode ser processada em threads diferentes. Se você quiser tratar cada conexão na mesma thread, defina em `WorkOpThreads` o número de threads usadas para tratar essas requisições. Isso impacta o desempenho do servidor e só é recomendado definir um valor maior que zero se você realmente precisar deste recurso.

Habilitar o EPOLL para servidores Linux é recomendado quando você precisa tratar milhares de conexões. Se o seu servidor está tratando no máximo 100 conexões concorrentes, você pode permanecer com o modelo de threads padrão do Indy.

Evento OnDisconnect não disparado

O EPOLL funciona de forma diferente do IOHandler indy padrão. Com o IOHandler indy padrão, cada conexão é executada em uma thread, e essas threads ficam em execução o tempo todo verificando se a conexão está ativa, de modo que, se houver uma desconexão, ela é notificada em um curto período de tempo.

O EPOLL funciona de forma diferente, há um pool de threads que trata de todas as conexões, em vez de 1 thread = 1 conexão como o indy faz por padrão. Para EPOLL, a única maneira de detectar se uma conexão ainda está ativa é tentar escrever no socket; se houver algum erro, significa que a conexão está fechada. Há 2 opções para detectar desconexões:

1. Se você usar **TsgcWebsocketClient**, pode habilitá-lo na propriedade Options, **CleanDisconnect := True** (por padrão está desabilitado). Se estiver habilitado, antes de o cliente desconectar, ele envia uma mensagem informando o servidor sobre a desconexão, então o servidor receberá essa mensagem e o evento `OnDisconnect` será gerado.
2. Você pode habilitar o **heartbeat** no lado do **servidor**, por exemplo a cada 60 segundos, de modo que ele tentará enviar um ping a todos os clientes conectados e, se houver algum cliente desconectado, `OnDisconnect` será chamado.

Linux Connections Limit

Se você quiser aumentar o número de conexões abertas concorrentes, utilize o seguinte comando

```
ulimit -n 10000
```

O comando anterior define o número máximo de descritores de arquivos abertos como 10000

ALPN

Suportado por

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

***Requer versão personalizada do Indy.**

Application-Layer Protocol Negotiation (ALPN) é uma extensão do Transport Layer Security (TLS) para negociação de protocolo de camada de aplicação. O ALPN permite que a camada de aplicação negocie qual protocolo deve ser executado sobre uma conexão segura de uma maneira que evita round trips adicionais e que é independente dos protocolos de camada de aplicação. É necessário por conexões HTTP/2 seguras, que melhoram a compressão de páginas web e reduzem sua latência em comparação com o HTTP/1.x.

Client

Você pode configurar em `TLSSOptions.ALPNProtocols` quais protocolos são suportados pelo cliente. Quando o cliente conecta ao servidor, esses protocolos são enviados no 'Client Hello' inicial do handshake TLS, listando os protocolos que o cliente suporta, e o servidor seleciona qual protocolo será utilizado, se algum.

Você pode obter qual protocolo foi selecionado pelo servidor acessando a propriedade `ALPNProtocol` do `TsgcWSConnectionClient`.

Server

Quando há uma nova conexão TLS, o evento `OnSSLALPNSelect` é chamado, aqui você pode acessar uma lista de protocolos que são suportados pelo cliente e o servidor pode selecionar qual deles é suportado.

Se não houver suporte para nenhum protocolo, `aProtocol` pode ser deixado vazio.

```
// Client
void OnClientConnect(TsgcWSConnection *Connection)
{
    string vProtocol = "";
    vProtocol = dynamic_cast<TsgcWSConnectionClient*>(Connection)->ALPNProtocol;
}

// Server
void OnSSLALPNSelect(TObject *Sender, TStringList *aProtocols, String &aProtocol)
{
    for (int i = 0; i < aProtocols->Count; i++)
    {
        if (aProtocols->Strings[i] == "h2")
        {
            aProtocol = "h2";
            break;
        }
    }
}
```

Encaminhar Requisições HTTP

Suportado por

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

Você pode configurar o servidor para encaminhar algumas requisições HTTP para outro servidor. Isto é muito útil quando você tem mais de um servidor e apenas um servidor está escutando em um endereço público.

Exemplo: você pode configurar seu servidor para encaminhar a outro servidor todas as requisições para `/internal`, enquanto todas as outras requisições são tratadas pelo servidor `sgcWebSockets`.

Use o evento **OnBeforeForwardHTTP** para verificar se a URL solicitada deve ser encaminhada e, se for, definir a URL a encaminhar.

Exemplo: se você quiser encaminhar todas as requisições do documento `"/internal"` para o servidor `"localhost:8080"`, faça o seguinte:

```
void OnBeforeForwardHTTP(TsgcWSConnection *Connection, TIDHTTPRequestInfo *ARequestInfo,
    TsgcWSServerForwardHTTP *aForward)
{
    if (ARequestInfo->Document == "/internal")
    {
        aForward->Enabled = true;
        aForward->URL = "http://localhost:8080";
    }
}
```

Outras Opções

Quando você quiser encaminhar uma requisição HTTP, tem as seguintes opções adicionais:

1. Por padrão, a requisição é encaminhada usando o documento original. **Exemplo:** se você encaminhar a requisição `http://localhost:8080/internal` para o servidor interno `http://localhost:5555`, a URL encaminhada será `http://localhost:5555/internal`. Mas você pode modificar o `Document`, usando a propriedade **Document** do objeto `Forward` (por padrão, usará o mesmo da requisição original).

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

2. Se você encaminhar uma conexão HTTP segura (HTTPSs), pode personalizar as opções de SSL/TLS na propriedade **TLSoptions** do objeto `Forward`. **Exemplo:** definir a versão TLS

```
aForward.TLSoptions.Version = tls1_2
```

3. As seguintes propriedades podem ser usadas para personalizar a requisição HTTP:

- **QueryParams:** os parâmetros após o documento, exemplo: `'id=1&user=2'`.
- **Host:** especifica o host e o número da porta do servidor para o qual a requisição está sendo enviada. Exemplo: `www.esegece.com:443`
- **Origin:** a origem (esquema, hostname e porta) que causou a requisição. Exemplo: `https://www.esegece.com/document`.
- **LogFilename:** o nome do arquivo onde a requisição/resposta será armazenada.
- **NoCache:** se a requisição não deve utilizar o cache do navegador web; por padrão está habilitado.
- **CustomHeaders:** uma Lista de cabeçalhos personalizados a serem adicionados à requisição. Exemplo: `CustomHeaders.Add('X-ReverseProxy-Host: http://127.0.0.1:8888/test');`

Quality Of Service

Suportado por

TsgcWSPServer_sgc
TsgcWSPClient_sgc
TsgcWSPClient_MQTT
Java script

SGC Default Protocol e MQTT implementam QoS (Quality of Service) para entrega de mensagens. Há 3 tipos diferentes:

Level 0: "No máximo uma vez", onde as mensagens são entregues de acordo com o melhor esforço (best efforts) da rede TCP/IP subjacente. Perda ou duplicação de mensagens pode ocorrer. Este nível pode ser utilizado, por exemplo, com dados de sensores ambientais onde não importa se uma leitura individual é perdida já que a próxima será publicada logo depois.

Level 1: "At least once", onde a chegada das mensagens é garantida, mas duplicatas podem ocorrer.

Level 2: "Exatamente uma vez", em que se garante que as mensagens chegam exatamente uma vez. Este nível pode ser usado, por exemplo, com sistemas de cobrança onde mensagens duplicadas ou perdidas poderiam levar à aplicação de cobranças incorretas.

Nível 0

A mensagem é entregue de acordo com os melhores esforços da rede TCP/IP subjacente. Uma resposta não é esperada e nenhuma semântica de reenvio é definida no protocolo. A mensagem chega ao servidor uma vez ou não chega.

A tabela abaixo mostra o fluxo do protocolo no nível QoS 0.

Client	Message and direction	Servidor
QoS = 0	PUBLISH ----->	Action: Publicar uma mensagem para assinantes

Level 1

O recebimento de uma mensagem pelo servidor é confirmado por uma mensagem ACKNOWLEDGEMENT. Se houver uma falha identificada do link de comunicação ou do dispositivo emissor, ou se a mensagem de confirmação não for recebida após um período de tempo especificado, o remetente reenvia a mensagem. A mensagem chega ao servidor pelo menos uma vez.

Uma mensagem com QoS nível 1 tem um Message ID na mensagem.

A tabela abaixo mostra o fluxo do protocolo QoS de nível 1.

Client	Message and direction	Servidor
QoS = 1 Message ID = x Action: Armazenar mensagem	PUBLISH ----->	Actions: • Armazenar mensagem • Publicar uma mensagem para os assinantes • Excluir mensagem

Action: Descartar mensagem	ACKNOWLEDGEMENT 	

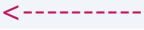
Se o cliente não receber uma mensagem ACKNOWLEDGMENT (seja dentro de um período de tempo definido na aplicação, ou se uma falha for detectada e a sessão de comunicação for reiniciada), o cliente pode reenviar a mensagem PUBLISH.

Level 2

Fluxos de protocolo adicionais acima do QoS nível 1 garantem que mensagens duplicadas não sejam entregues à aplicação receptora. Este é o nível mais alto de entrega, para uso quando mensagens duplicadas não são aceitáveis. Há um aumento no tráfego de rede, mas isso geralmente é aceitável devido à importância do conteúdo da mensagem.

Uma mensagem com QoS nível 2 tem um Message ID na mensagem.

A tabela abaixo mostra o fluxo do protocolo QoS level 2. Há duas semânticas disponíveis para como um fluxo PUBLISH deve ser tratado pelo destinatário.

Client	Message and direction	Servidor
QoS = 2 Message ID = x Action: Armazenar mensagem	PUBLISH 	Action: Armazenar mensagem
	PUBREC 	Message ID = x
Message ID = x	PUBREL 	Actions: • Publicar uma mensagem aos assinantes • Excluir mensagem
Action: Descartar mensagem	ACKNOWLEDGEMENT 	Message ID = x

Se uma falha for detectada, ou após um período de tempo definido, o fluxo do protocolo é repetido a partir da última mensagem de protocolo não confirmada. Os fluxos de protocolo adicionais garantem que a mensagem seja entregue aos assinantes exatamente uma vez.

Queues

Suportado por

[TsgcWSPServer_sgc](#)

[TsgcWSPClient_sgc](#)

Java script

O [Protocolo Padrão SGC](#) implementa Filas para adicionar persistência às mensagens publicadas (está disponível apenas para **mensagens Publicadas**)

Level 0: As mensagens não são enfileiradas no Servidor

Level 1: apenas a última mensagem é enfileirada no Servidor, e é enviada toda vez que um cliente assina um novo canal ou se conecta ao servidor.

Nível 2: Todas as mensagens são enfileiradas no Servidor e são enviadas toda vez que um cliente assina um novo canal ou se conecta ao servidor.

Nível 0

A mensagem não é enfileirada pelo servidor

A tabela abaixo mostra o fluxo do protocolo Queue level 0.

Client	Message and direction	Servidor
Queue = 0	PUBLISH ----->	Action: Publicar uma mensagem para assinantes

Level 1

Uma mensagem com Queue level 1 é armazenada no servidor e, se houver outras mensagens armazenadas para este canal, elas são excluídas.

A tabela abaixo mostra o fluxo de protocolo da Queue de nível 1.

Client	Message and direction	Servidor
Queue = 1	PUBLISH ----->	Ações: • Exclui todas as mensagens deste canal • Armazena a última mensagem por Canal
Action: Processar mensagem	NOTIFY <-----	Action: Cada vez que um novo cliente assina este canal, a última mensagem é enviada.

Isto é útil onde os publicadores enviam mensagens com base em "relatório por exceção", em que pode decorrer algum tempo entre as mensagens. Isso permite que novos assinantes recebam instantaneamente os dados com o valor retido, ou Last Known Good (último valor válido conhecido).

Level 2

Todas as mensagens com nível de Fila 2 são armazenadas no servidor.

A tabela abaixo mostra o fluxo do protocolo Queue level 2.

Client	Message and direction	Servidor
Queue = 2	PUBLISH ----->	Action: Armazenar mensagem
Action: Proces-sar mensagem	NOTIFY <-----	Action: Toda vez que um novo cliente assina este canal, TODAS as Mensagens são enviadas.

Transactions

Suportado por

[TsgcWSPServer_sgc](#)

[TsgcWSPClient_sgc](#)

Java script

O Protocolo SGC do sgcWebSockets suporta mensageria transacional, quando um cliente confirma uma transação, todas as mensagens enviadas pelo cliente são processadas no lado do servidor. Há 3 métodos chamados pelo cliente:

StartTransaction

Cria uma Nova Transação no lado do servidor e todas as mensagens enviadas do cliente para o servidor após este método são enfileiradas no lado do servidor até que o cliente chame Commit ou Rollback.

Client	Message and direction	Servidor
Channel = X	STARTTRANSACTION ----->	Action: Cria uma nova Queue para armazenar todas as Mensagens do canal especificado
Channel = X	PUBLISH ----->	Action: A mensagem é armazenada no lado do servidor.
Action: O cliente recebe a confirmação da mensagem enviada	ACKNOWLEDGEMENT < -----	Action: O servidor retorna um Acknowledgement ao client porque a mensagem foi armazenada.
....		

Commit

Quando um cliente chama Commit, todas as mensagens enfileiradas pelo servidor são processadas.

Client	Message and direction	Servidor
Channel = X	COMMIT ----->	Action: Processa todas as mensagens enfileiradas pela Transaction

RollBack

Quando um cliente chama RollBack, todas as mensagens enfileiradas pelo servidor são excluídas e não são processadas no lado do servidor.

Client	Message and direction	Servidor

Channel = X	ROLLBACK ----->	Ação: Exclui todas as mensagens enfileiradas por Transaction

Conexões TCP

Suportado por

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

Por padrão, o `sgcWebSocket` utiliza o `WebSocket` como protocolo, mas você pode utilizar o protocolo TCP simples nos componentes de cliente e servidor.

Cliente Componente

Desabilita o protocolo `WebSocket`.

```
Client->Specifications->RFC6455 = false;
```

Server Component

Trate o evento `OnUnknownProtocol` e defina `Transport` como `trpTCP` e aceite (`Accept`) a conexão.

```
void OnUnknownProtocol(TsgcWSConnection *Connection, ref bool Accept)
{
    Connection->Transport = trpTCP;
    Accept = true;
}
```

Então, quando um cliente se conecta ao servidor, esta conexão será definida como TCP e usará o protocolo TCP puro em vez de `WebSockets`. As conexões TCP puro não distinguem entre mensagens de texto e binárias, portanto todas as mensagens recebidas são tratadas pelo evento `OnBinary`.

Fim da mensagem

Se as mensagens forem grandes, elas podem às vezes ser recebidas fragmentadas. Existe um método para detectar o fim de uma mensagem especificando quais bytes procurar. Exemplo: no protocolo STOMP, todas as mensagens terminam com os bytes 0 e 10.

```
void OnWSClientConnect(TsgcWSConnection *Connection)
{
    Connection->TCPEndOfFrameScanBuffer = eofScanAllBytes;
    Connection->AddTCPEndOfFrame(0);
    Connection->AddTCPEndOfFrame(10);
}
```

SubProtocol

Suportado por

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

O WebSocket fornece uma negociação simples de subprotocolo. Basicamente, ele adiciona um cabeçalho com os nomes de protocolo suportados pela requisição. Esses protocolos são recebidos e, se o receptor suportar um deles, envia uma resposta com o subprotocolo suportado.

O sgcWebSockets suporta vários SubProtocolos: [MQTT](#), [WAMP](#)... e mais. Você pode implementar seus próprios subprotocolos utilizando um método muito fácil, basta chamar RegisterProtocol e enviar o Nome do SubProtocolo como argumento.

Exemplo: você precisa conectar a um servidor que implementa o subprotocolo "Test 1.0"

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

Para **utilizar mais de 1 protocolo em uma única conexão**, você pode utilizar os componentes **Broker Protocol** (Servidor e Cliente) para tratá-lo. Basta colocar um Broker entre o Cliente/Servidor e os protocolos. **Exemplo:** protocolos User SGC e Files utilizando uma única conexão.

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

Quando um protocolo de broker é anexado entre o Servidor/Cliente e o protocolo, os eventos **OnConnect** e **OnDisconnect** são disparados no componente Broker (em vez de nos componentes Servidor ou Cliente).

Throttle

Suportado por

[TsgcWebSocketServer](#)

[TsgcWebSocketHTTPServer](#)

[TsgcWebSocketClient](#)

A limitação de banda (Bandwidth Throttling) é suportada por componentes servidor e cliente; se habilitada, pode limitar o número de bits por segundo enviados/recebidos pelo socket. O Indy utiliza um método bloqueante, portanto, se um cliente estiver limitando sua leitura, os dados não lidos ficarão dentro do socket do cliente e o servidor ficará bloqueado para escrever novos dados ao cliente. Quanto mais lentamente o cliente lê os dados, mais lentamente o servidor pode escrever novos dados.

Server-sent Events (Notificações push)

Suportado por

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)
Java script

SSE não fazem parte dos WebSockets, definem uma API para abrir uma conexão HTTP para receber notificações push de um servidor.

Os SSEs são enviados sobre HTTP tradicional. Isso significa que não requerem um protocolo ou implementação de servidor especial para funcionar. Além disso, os Server-Sent Events têm uma variedade de recursos que os WebSockets, por design, não têm, como reconexão automática, IDs de evento e a capacidade de enviar eventos arbitrários.

Eventos

- **Open:** quando uma nova conexão SSE é aberta.
- **Message:** quando o cliente recebe uma nova mensagem.
- **Error:** quando há qualquer erro de conexão, como uma desconexão.

JavaScript API

Para se inscrever em um stream de eventos, crie um objeto EventSource e passe a ele a URL do seu 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 as atualizações são enviadas (push) do servidor, o handler onmessage dispara e novos dados ficam disponíveis em sua propriedade e.data. Se a conexão for fechada, o navegador reconectará automaticamente à fonte após ~3 segundos (este é um intervalo de retry padrão, você pode alterá-lo no lado do servidor).

Campos

Os seguintes nomes de campo são definidos pela especificação:

event

O tipo do evento. Se isto for especificado, um evento será despachado no navegador para o listener do nome de evento especificado; o site utilizaria addEventListener() para escutar eventos nomeados. O manipulador onmessage é chamado se nenhum nome de evento for especificado para uma mensagem.

data

O campo de dados para a mensagem. Quando o EventSource recebe várias linhas consecutivas que começam com data:, ele as concatena, inserindo um caractere de nova linha entre cada uma. As novas linhas finais são removidas.

id

O ID do evento para definir o valor do último event ID do objeto EventSource.

retry

O tempo de reconexão a ser usado ao tentar enviar o evento. Este deve ser um inteiro, especificando o tempo de reconexão em milissegundos. Se um valor não inteiro for especificado, o campo será ignorado.

Todos os outros nomes de campo são ignorados.

Para strings de múltiplas linhas, use #10 como line feed.

Exemplos de uso:

Se você precisar enviar uma mensagem a um cliente, basta utilizar o método 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

Suportado por

[TsgcWebSocketServer](#)

[TsgcWebSocketHTTPServer](#)

[TsgcWebSocketLoadBalancerServer](#)

O Balanceamento de Carga permite distribuir o trabalho entre vários servidores back-end; cada vez que um novo cliente solicita uma conexão, ele se conecta a um servidor balanceador de carga (que está conectado aos servidores back-end) e retorna uma string de conexão com informações sobre o host, porta... que é usada pelo cliente para se conectar a um servidor. Se você tiver, por exemplo, 4 servidores, com este método todos os servidores terão, mais ou menos, o mesmo número de conexões, e a carga de trabalho será similar.

Se um cliente quiser enviar uma mensagem a todos os clientes de todos os servidores, basta usar o método Broadcast, e a mensagem será difundida a todos os servidores conectados ao Load Balancer Server.

Para habilitar este recurso:

1. Solte um componente [TsgcWebSocketLoadBalancerServer](#), defina uma porta de escuta e defina active como True.
2. Os componentes de Servidor e Cliente têm uma propriedade chamada LoadBalancer, onde você precisa definir o host e a porta do Servidor Balanceador de Carga e definir Enabled como True.

O componente permite o balanceamento de carga dos protocolos **WebSocket** e **HTTP**.

Files

Suportado por

[TsgcWSPServer_sgc](#)

[TsgcWSPClient_sgc](#)

Este protocolo permite enviar arquivos do cliente para o servidor e do servidor para o cliente de forma fácil. Você pode enviar desde arquivos realmente pequenos até arquivos grandes utilizando baixo consumo de memória. Você pode definir:

1. Tamanho do pacote em bytes.
2. Use canais personalizados para enviar arquivos apenas a clientes assinados.
3. O progresso do envio e recebimento de arquivos.
4. Autorização dos arquivos recebidos.
5. Confirmação de pacotes enviados.

Proxy

Suportado por

[TsgcWebSocketClient](#)

Os componentes Client WebSocket suportam conexões WebSocket através de proxies HTTP. Para habilitar a conexão por proxy, você precisa ativar as seguintes propriedades:

Proxy / Enabled

Uma vez definido como True, você pode configurar:

Host: Endereço do servidor proxy

Port: Porta do servidor proxy

UserName/Password: Autenticação para conectar ao proxy, apenas se necessário.

ProxyType: os seguintes proxies são suportados:

- HTTP
- Socks4
- Socks4A
- Socks5

Você pode configurar proxies SOCKS acessando a propriedade SOCKS e definindo Enabled como True.

Mensagens Fragmentadas

Suportado por

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

Por padrão, quando um stream é enviado utilizando a biblioteca sgcWebSockets, ela envia todos os dados em um único pacote ou faz buffer de todos os pacotes e, quando o último pacote é recebido, o evento de mensagem OnBinary é chamado.

Esse comportamento pode ser personalizado pela propriedade **Options.FragmentedMessages**, que aceita os seguintes valores:

1. frgOnlyBuffer: este é o valor padrão. Significa que as mensagens de pacote serão armazenadas em buffer e apenas quando o stream inteiro for recebido o evento de mensagem OnBinary será chamado.
2. frgOnlyFragmented: isto significa que o evento OnFragmented só será chamado para cada pacote recebido.
3. frgAll: isto significa que o evento OnFragmented será chamado para cada pacote recebido e quando o stream completo for recebido.

O evento **OnFragmented** é útil quando você precisa enviar grandes streams e o receptor deve mostrar o progresso da transferência.

Exemplo: o cliente deve enviar um stream de tamanho 1.000.000 bytes ao servidor e o servidor deseja mostrar o progresso a cada 1000 bytes recebidos.

O cliente enviará um stream utilizando o método writedata com um tamanho de pacote de 1000

```
Client->WriteData(stream, 1000);
```

O servidor definirá Options.FragmentedMessages := frgAll e tratará o evento OnFragmented para receber o progresso dos streams

```
void OnFragmented(TsgcWSConnection *Connection, const TMemoryStream *Data, const TOpCode OpCode, const boolean C
{
    ShowProgress(Data->Size);
    if (Continuation == false)
    {
        SaveStream(Data);
    }
}
```

TsgcWebSocketClient

Componente cliente WebSocket — conecta-se a qualquer servidor WebSocket RFC 6455 e envia/recebe mensagens de texto e binárias.

Introdução

TsgcWebSocketClient implementa o componente cliente WebSocket e pode conectar a um servidor WebSocket. Siga os passos abaixo para configurar este componente:

1. Coloque um componente **TsgcWebSocketClient** no form.
2. Defina **Host** e **Port** (o padrão é 80) para conectar a um servidor WebSocket disponível. Você pode definir a propriedade **URL** e Host, Port, Parameters... serão atualizados a partir da URL. **Exemplo:** wss://127.0.0.1:8080/ws/ resultará em:

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

3. Você pode selecionar se requer **TLS** (conexão segura) ou não; por padrão, não está ativado.
4. Você pode conectar-se por meio de um HTTP Proxy Server; é necessário definir as propriedades de proxy:
Host: nome de host do servidor proxy.
Port: número da porta do servidor proxy.
Username: nome de usuário para autenticação, deixe em branco para anônimo.
Password: senha para autenticação, deixe em branco para anônimo.
5. Se o servidor suportar **compressão**, você pode habilitar a compressão para comprimir as mensagens que são enviadas.
6. Defina as **Specifications** permitidas; por padrão, todas as especificações estão habilitadas.
RFC6455: é a especificação WebSocket padrão e recomendada.
Hixie76: sempre é false
7. Defina Active = true para iniciar uma nova conexão WebSocket.

Referência

- Propriedades — 23 propriedades disponíveis no componente.
- Métodos — 22 métodos públicos.
- Eventos — 16 eventos que você pode tratar.

Guides

Most common uses

- **Connection**
 - [Como Conectar a um Servidor WebSocket](#)
 - [Abrir uma Conexão de Cliente](#)
 - [Fechar uma Conexão de Cliente](#)
 - [Manter a Conexão ativa](#)
 - [Desconexões Inesperadas](#)
 - [Conectar a um Servidor TCP](#)
 - [Redirecionamentos WebSocket](#)
- **Servidores Seguros**
 - [Conectar a Servidor Seguro](#)

- [Certificados OpenSSL](#)
- [Certificados SChannel](#)
- [SChannel Get Connection Info](#)
- **Enviar Mensagens**
 - [Enviar Mensagem de Texto](#)
 - [Enviar Mensagem Binária](#)
- **Receber Mensagens**
 - [Receber Mensagens de Texto](#)
 - [Receber Mensagens Binárias](#)
- **Autenticação**
 - [Client Authentication](#)
- **Other**
 - [Client Exceptions](#)
 - [Client WebSocket HandShake](#)
 - [Client Register Protocol](#)
 - [Client Proxies](#)

TsgcWebSocketClient | Connect WebSocket Server

URL Property

A maneira mais fácil de conectar-se a um servidor WebSocket é utilizar a propriedade **URL** e definir **Active = true**.

Exemplo: conectar a www.esegece.com utilizando conexão segura.

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

Host, Port e Parameters

Você pode conectar a um servidor WebSocket utilizando as propriedades Host e port.

Exemplo: conectar a www.esegece.com utilizando conexões seguras

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

TsgcWebSocketClient | Client Open Connection

Depois que seu cliente está configurado para se conectar a um servidor, há 3 opções diferentes para abrir uma nova conexão.

Propriedade Active

A maneira mais fácil de abrir uma nova conexão é definir a propriedade Active como true. Isso tentará conectar ao servidor usando a configuração do componente.

Se você definir a propriedade Active como false, ela fechará a conexão se estiver ativa.

Este método é executado na mesma thread que o chamador. Portanto, se você o chamar a partir da Main Thread, o método será executado na Main Thread da aplicação.

Open Connection

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

Quando você chama Active = true, **você ainda não pode enviar nenhum dado ao servidor** porque o cliente ainda pode estar conectando. Você deve primeiro aguardar até que o evento OnConnect seja disparado e, então, pode começar a enviar mensagens ao servidor.

Close Connection

```
oClient->Active = false;
```

Quando você chama Active = false, **você não pode ter certeza de que a conexão já foi fechada** logo após esse código, portanto você deve aguardar até que o evento OnDisconnect seja disparado.

Métodos Start/Stop

Quando você chama Start() ou Stop() para conectar/desconectar do servidor, a chamada é executada em uma thread secundária, de modo que não bloqueia a thread onde ela é chamada. Utilize este método se você quiser conectar a um servidor e deixar o seu código abaixo continuar.

Open Connection

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

Quando você chama Start(), **você ainda não pode enviar nenhum dado ao servidor** porque o cliente pode ainda estar conectando. Você deve primeiro aguardar até que o evento OnConnect seja disparado, e então pode começar a enviar mensagens ao servidor.

Close Connection

```
oClient->Stop();
```


Quando você chama `Stop()`, **você não pode ter certeza de que a conexão já está fechada** logo após este código, portanto você deve aguardar até que o evento `OnDisconnect` seja disparado.

Métodos Connect/Disconnect

Quando você chama `Connect()` ou `Disconnect()` para abrir/fechar uma conexão com o servidor, a chamada é executada na mesma thread onde é chamada, mas aguarda até que o processo seja concluído. Você deve definir um `Timeout` para definir o tempo máximo a aguardar até que o processo seja concluído (por padrão 10 segundos).

Exemplo: conectar ao servidor e aguardar até 5 segundos

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

Se o método `Connect()` retornar um resultado bem-sucedido, você já pode enviar uma mensagem ao servidor porque a conexão está ativa.

Exemplo: desconectar do servidor e aguardar até 10 segundos

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

Se o método `Disconnect()` retornar um resultado bem-sucedido, isso significa que a conexão já está fechada.

O evento **OnBeforeConnect** pode ser utilizado para personalizar as propriedades de conexão do servidor antes de o cliente tentar conectar a ele.

TsgcWebSocketClient | Client Close Connection

A conexão pode ser fechada usando a propriedade `Active`, os métodos `Stop` ou `Disconnect`, leia mais em [Client Open Connection](#).

CleanDisconnect

Quando uma conexão é fechada, você pode notificar o outro peer de que a conexão está sendo fechada enviando uma mensagem de fechamento. Para habilitar esse recurso, defina a propriedade `Options.CleanDisconnect` como `true`.

Se esta propriedade estiver habilitada, antes de a conexão ser fechada, uma mensagem `Close` será enviada ao servidor para notificar que o cliente está fechando a conexão.

Disconnect

[TsgcWSConnection](#) possui um método chamado `Disconnect()`, que permite que você desconecte a conexão no nível do socket. Se você chamar este método, o socket será desconectado diretamente sem aguardar qualquer resposta do servidor. Você pode enviar um `Close Code` com este método.

Close

[TsgcWSConnection](#) possui um método chamado `Close()`, que permite enviar uma mensagem ao servidor solicitando o fechamento da conexão. Se o servidor receber esta mensagem, ele deve fechar a conexão e o cliente receberá uma notificação de que a conexão foi fechada. Você pode enviar um `Close Code` com este método.

TsgcWebSocketClient | Manter a conexão do cliente aberta

Uma vez que seu cliente tenha conectado a um servidor, às vezes a conexão pode ser fechada devido a sinal fraco, erros de conexão, etc. Há 2 propriedades que ajudam a manter a conexão ativa.

HeartBeat

A propriedade **HeartBeat** permite que você **envie um Ping** a cada **X segundos** para **manter a conexão ativa**. Alguns servidores fecham conexões TCP se não houver dados trocados entre os peers. O HeartBeat resolve esse problema enviando um ping em um intervalo específico. Geralmente isto é suficiente para manter uma conexão ativa, mas você pode definir um intervalo de Timeout se quiser fechar a conexão quando uma resposta do servidor não for recebida após X segundos.

Exemplo: envia um ping a cada 30 segundos

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

Há um evento chamado **OnBeforeHeartBeat** que permite personalizar o comportamento do HeartBeat. Por padrão, se o HeartBeat estiver habilitado, o cliente enviará um ping WebSocket a cada X segundos, conforme definido pela propriedade HeartBeat.Interval.

OnBeforeHeartBeat tem um parâmetro chamado Handled, por padrão é false, o que significa que o fluxo é controlado pelo componente TsgcWebSocketClient. Se você definir o valor como True, então o ping não será enviado, e você pode enviar sua mensagem personalizada utilizando a classe Connection.

WatchDog

Se o WatchDog estiver habilitado, quando o cliente detectar uma desconexão, o WatchDog tenta reconectar a cada X segundos até que a conexão esteja ativa novamente.

Exemplo: reconecta a cada 10 segundos após uma desconexão com tentativas ilimitadas.

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

Você pode utilizar o evento **OnBeforeWatchDog** para alterar o servidor ao qual o cliente tentará conectar. **Exemplo:** após 3 tentativas, se o cliente não conseguir conectar a um servidor, ele tentará conectar a um servidor secundário.

A propriedade **Handled**, se definida como True, significa que o cliente não tentará reconectar.

TsgcWebSocketClient | Dropped Disconnections

Depois que a conexão foi estabelecida, se nenhum peer enviar dados, então nenhum pacote é enviado pela rede. O TCP é um protocolo ocioso, então ele presume que a conexão ainda está ativa.

Motivos de desconexão

- **A aplicação fecha:** quando um processo é finalizado, normalmente envia um pacote FIN que confirma ao outro peer que a conexão foi fechada. Mas se um processo falha, não há garantia de que este pacote será enviado ao outro peer.
- **Device Closes:** se o dispositivo fechar, muito provavelmente não haverá notificação sobre isso.
- **Cabo de rede desconectado:** se o cabo de rede for desconectado, é o mesmo que um roteador fechando; não há dados sendo transferidos, então a conexão não é fechada.
- **Perda de sinal do roteador:** se a aplicação perder o sinal do roteador, a conexão ainda permanecerá ativa.

Detectar desconexões Half-Open

Você pode tentar detectar desconexões utilizando os seguintes métodos

Segunda Conexão

Você pode tentar abrir uma segunda conexão e tentar conectar, mas isto tem algumas desvantagens: você consome mais recursos, cria novas threads, etc. Além disso, se o outro peer tiver reiniciado, a segunda conexão funcionará, mas a primeira não.

Pingar outro peer

Se você tentar enviar um ping ou qualquer mensagem com uma conexão half-open, você verá que não recebe nenhum erro.

Habilitar o KeepAlive no nível do Socket TCP

Um pacote TCP keep-alive é simplesmente um ACK com o número de sequência definido como um a menos que o número de sequência atual da conexão. Um host que recebe um desses ACKs responde com um ACK para o número de sequência atual. Os keep-alives podem ser usados para verificar se o computador na extremidade remota de uma conexão ainda está disponível. Os TCP keep-alives podem ser enviados uma vez a cada TCPKeepAlive.Time (padrão de 7.200.000 milissegundos ou duas horas) se nenhum outro dado ou keep-alives de nível mais alto tiverem sido transportados pela conexão TCP. Se não houver resposta a um keep-alive, ele é repetido uma vez a cada TCPKeepAlive.Interval segundos. O KeepAliveInterval é por padrão 1000 milissegundos.

Você pode habilitar o KeepAlive por conexão e permitir que o protocolo TCP verifique se a conexão está ativa ou não. Este é o método preferido se você quiser detectar desconexões abruptas (por exemplo: quando você desconecta um cabo de rede).

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient(NULL);
oClient->TCPKeepAlive->Enabled = true;
oClient->TCPKeepAlive->Time = 5000;
oClient->TCPKeepAlive->Interval = 1000;
```

TsgcWebSocketClient | Connect TCP Server

O TsgcWebSocketClient pode conectar a servidores WebSocket, mas também pode conectar a Servidores TCP simples.

URL Property

A maneira mais fácil de conectar a um servidor TCP é utilizar a propriedade **URL** e definir **Active = true**.

Exemplo: conectar a 127.0.0.1 porta 5555

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

Host, Port e Parameters

Você pode conectar a um servidor TCP utilizando as propriedades Host e port.

Exemplo: conectar 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 | Conexões

TIME_WAIT

Quando um cliente inicia uma desconexão do servidor, há uma troca entre cliente e servidor para comunicar o estado da desconexão. Quando o processo termina, a conexão do socket do cliente entra no estado `TIME_WAIT` por uma duração variável. Este é um comportamento normal; em sistemas operacionais Windows, esse tempo é, por padrão, de cerca de 4 minutos.

Você pode reduzir ou eliminar esse comportamento, mas faça-o com cautela, usando as alternativas a seguir.

REGEDIT

Você pode reduzir o valor de `TIME_WAIT` usando o Windows Regedit

1. Abra o Regedit e acesse as subchaves de registro `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TCPIP\Parameters`.
2. Crie um novo valor `REG_DWORD` chamado **TcpTimedWaitDelay**
3. Defina o valor em segundos. Exemplo: se você definir um valor de 5, significa que o `TIME_WAIT` aguardará por um máximo de 5 segundos.
4. Salve e reinicie o sistema.

LINGER

Outra opção para evitar o estado `TIME_WAIT` é usar a opção de socket `SO_LINGER`. Se habilitada, em vez de fechar a conexão graciosamente, o cliente reseta a conexão para que o estado `TIME_WAIT` seja evitado.

Você pode habilitar esta opção utilizando a propriedade **LingerState**, que por padrão tem um valor de -1. Se você definir um valor de zero, a conexão será resetada ao desconectar do socket sem um timeout.

Esta opção é provavelmente a menos recomendada e só deve ser utilizada como último recurso.

TsgcWebSocketClient | Redirecionamentos WebSocket

Quando o cliente se conecta a um servidor WebSocket, o servidor pode retornar um HTTP Response Code 30x. Se o código de resposta for 301, significa que o local foi movido permanentemente e a nova URL é fornecida no cabeçalho HTTP Location.

O cliente WebSocket trata redirecionamentos automaticamente, portanto, se detectar que a resposta do servidor contém um redirecionamento, ele desconectará a conexão atual e tentará conectar-se à nova URL de Location.

Exemplo

1. O cliente primeiro tenta se conectar à url `ws://127.0.0.1:5000`
2. O servidor retorna um Response Code de 301 e contém um Header Location com o valor `ws://80.50.1.2:3000`
3. O cliente lê a Resposta do servidor, detecta que é um redirecionamento e lê o Location
 1. Primeiro desconecta a conexão atual.
 2. Atualiza a propriedade URL com o valor do Cabeçalho Location (`ws://80.50.1.2:3000`)
 3. Conecta ao novo servidor.

TsgcWebSocketClient | Connect Secure Server

O TsgcWebSocketClient pode conectar a servidores WebSocket utilizando conexões seguras e não seguras.

Você pode configurar uma conexão segura, utilizando a propriedade URL ou as propriedades Host / Port, consulte [Connect to WebSocket Server](#).

TLSOptions

Na propriedade **TLSOptions** estão as propriedades para **personalizar uma conexão segura**. A propriedade mais importante é a **version**, que especifica a **versão do protocolo TLS**. Normalmente, definir a **propriedade TLS como true** e **TLSOptions.Version como tlsUndefined** é suficiente para a grande maioria dos Servidores WebSocket.

TLSOptions.Version permite que você defina a versão TLS utilizada para conectar ao servidor ou deixe o cliente negociar a versão TLS entre todas as disponíveis (este é o padrão quando o valor é **tlsUndefined**).

Se você obtiver um **erro ao tentar conectar a um servidor** relacionado ao protocolo TLS, **muito provavelmente** o servidor **requer uma versão TLS mais recente** do que a que você definiu.

Se **TLSOptions.IOHandler** estiver definido como **iohOpenSSL**, você precisa **implantar as bibliotecas OpenSSL** (que são as bibliotecas que tratam de todo o assunto TLS), confira o artigo a seguir sobre [OpenSSL](#).

Se **TLSOptions.IOHandler** estiver definido como **iohSChannel**, então **não há necessidade de implantar** nenhuma biblioteca (apenas o Windows é suportado).

TsgcWebSocketClient | Certificates OpenSSL

Quando o servidor exige que o cliente se conecte usando um Certificado SSL, use a propriedade `TLSEOptions` de `TsgcWebSocketClient` para definir os arquivos do certificado. O certificado deve estar no formato PEM, então, se o certificado tiver um formato diferente, ele deve primeiro ser convertido para PEM.

A conexão através das bibliotecas OpenSSL requer que `TLSEOptions.IOHandler = iohOpenSSL`.

Configure as seguintes propriedades:

- **CertFile:** é o caminho para o certificado no formato PEM.
- **KeyFile:** é o caminho para a chave privada do certificado.
- **RootCertFile:** é o caminho para a raiz do certificado.
- **Password:** se o certificado estiver protegido por senha, defina o segredo aqui.

TsgcWebSocketClient | Certificates SChannel

Quando o servidor exige que o cliente conecte usando um Certificado SSL, use a propriedade `TLSOptions` do `TsgcWebSocketClient` para definir os arquivos do certificado.

A conexão através do SChannel requer que `TLSOptions.IOHandler = iohSChannel`.

O SChannel suporta 2 tipos de autenticação por certificado:

1. Utilizando um **certificado PFX**
2. Definindo o **Hash Certificate** de um certificado já instalado no sistema windows.

PFX Certificate

O certificado PFX é um arquivo que contém o certificado e a chave privada; às vezes você tem um certificado no formato PEM, portanto, antes de utilizá-lo, é necessário convertê-lo para PFX.

Utilize o seguinte comando openssl para converter um certificado PEM em PFX

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

Uma vez que o certificado esteja no formato PFX, você só precisa implantar o certificado e definir a propriedade `TLSOptions.CertFile` com o caminho dele.

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

Hash Certificate

Se o certificado já estiver instalado no repositório de certificados do Windows, você só precisa conhecer o thumbprint do certificado e defini-lo na propriedade `TLSOptions.SChannel_Options`.

Encontrar o hash de um certificado é tão fácil no **powershell** quanto executar um comando **dir** no contêiner de certificados.

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

O hash é o valor hexadecimal do **Thumbprint**.

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

Uma vez que você tenha o valor do Thumbprint, deve definir o hash e a localização do certificado na propriedade `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 | SChannel Get Connection Info

Uma vez que o cliente tenha conectado ao servidor seguro, você pode solicitar informações sobre qual versão do TLS está sendo usada (TLS 1.2, TLS 1.3, etc.), a cifra usada, a força e mais.

Chame a função **GetInfo** do SChannel Handler para acessar essa informação. Você pode acessar o SSL Handler utilizando o método **OnSSLAfterCreateHandler**, que é chamado depois que o SChannel Handler é criado. Depois que o cliente conecta ao servidor, se o SSL Handler estiver atribuído, chame a função **GetInfo** e, se bem-sucedida, retornará os dados da conexão.

```
oClient = new TsgcWebSocketClient();
oClient->URL = "wss://www.esegece.com:2053";
oClient->TLSOptions->Version = tls1_2;
oClient->TLSOptions->IOHandler = iohSChannel;
oClient->OnSSLAfterCreateHandler = OnSSLAfterCreateHandlerEvent;
oClient->OnConnect = OnConnectEvent;
oClient->Active = true;
void OnSSLAfterCreateHandlerEvent(TObject *Sender, TwssSSLHandler aType,
    TIdSSLIOHandlerSocketBase *aSSLHandler)
{
    if (aSSLHandler->ClassType() == __classid(TsgcIdSSLIOHandlerSocketSChannel))
    {
        SSL = dynamic_cast<TsgcIdSSLIOHandlerSocketSChannel*>(aSSLHandler);
    }
}
void OnConnectEvent(TsgcWSConnection *Connection)
{
    if (SSL != NULL)
    {
        TsgcSChannelConnectionInfo oInfo = SSL->GetInfo();
        if (oInfo->Protocol != tls1_2)
        {
            throw Exception("Client cannot connect using TLS 1.2");
        }
    }
}
```

TsgcWebSocketClient | Cliente Envia Mensagem de Texto

Uma vez que o cliente se conectou ao servidor, ele pode enviar mensagens de texto. Para enviar uma mensagem de texto, basta chamar o método `WriteData()`.

Enviar uma Mensagem de Texto

Chame o método **`WriteData()`** para enviar uma mensagem de texto. Este método é executado na **mesma thread** a partir da qual é chamado.

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

Se **`QueueOptions.Text`** tem um **valor diferente de `qmNone`**, em vez de ser processada na mesma thread que é chamada, ela será processada em uma thread secundária. Por padrão, esta opção está desabilitada.

Enviar uma Mensagem de Texto e Aguardar a Resposta

Chame o método `WriteAndWaitData()` para enviar uma mensagem de texto e aguardar uma resposta do servidor. A função retorna a mensagem de texto recebida.

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

TsgcWebSocketClient | Cliente enviar mensagem binária

Depois que o cliente se conectou ao servidor, ele pode enviar mensagens binárias. Para enviar uma mensagem binária, basta chamar o método `WriteData()`.

Enviar uma Mensagem Binária

Chame o método **`WriteData()`** para enviar uma mensagem binária. Este método é executado na **mesma thread** a partir da qual é chamado.

```
TMemoryStream oStream = new TMemoryStream();  
...  
TsgcWebSocketClient1->WriteData(oStream);  
delete oStream;
```

Se **`QueueOptions.Binary`** tiver um **valor diferente de `qmNone`**, em vez de ser processada na mesma thread que é chamada, ela será processada em uma thread secundária. Por padrão, esta opção está desabilitada.

Enviar uma Mensagem Binária e Aguardar a Resposta

Chame o método `WriteAndWaitData()` para enviar uma mensagem binária e aguardar uma resposta do servidor. A função retorna a mensagem binária recebida.

```
TsgcWebSocketClient1->WriteAndWaitData(oStream);
```

TsgcWebSocketClient | Cliente Envia uma Mensagem de Texto e Binária

O protocolo WebSocket só permite dois tipos de mensagens: Texto ou Binária. No entanto, você não pode enviar dados binários junto com texto na mesma mensagem.

Uma forma de resolver isso é adicionar um cabeçalho à mensagem binária antes de ela ser enviada e decodificar essa mensagem binária quando ela for recebida.

Existem 2 funções em `sgcWebSocket_Helpers` que podem ser utilizadas para definir uma descrição curta de um pacote binário. Isto basicamente adiciona um cabeçalho ao stream que é utilizado para identificar o pacote binário.

Antes de enviar uma mensagem binária, chame o método para codificar o stream.

```
sgcWSStreamWrite("00001", oStream);  
TsgcWebSocketClient1->WriteData(oStream);
```

Quando uma mensagem binária é recebida, chame o método para decodificar o stream.

```
sgcWSStreamRead(oStream, vID);
```

A única limitação é que o texto usado para identificar a mensagem binária tem um comprimento máximo de 10 caracteres (isso pode ser modificado se você tiver acesso ao código-fonte).

TsgcWebSocketClient | Receber Mensagens de Texto

Quando o cliente recebe uma mensagem de texto, o evento **OnMessage** é disparado. Leia o parâmetro `Text` para recuperar a string da mensagem recebida.

```
void OnMessage(TsgcWSConnection *Connection, const string Text)
{
    ShowMessage("Message Received from Server: " + Text);
}
```

Por padrão, o cliente usa o método **neAsynchronous** para despachar o evento `OnMessage`, o que significa que **este evento é executado no contexto da Thread Principal**, portanto é thread-safe atualizar qualquer controle de um formulário, por exemplo.

Se o seu cliente recebe muitas mensagens ou você precisa controlar a sincronização com outras threads, defina a propriedade `NotifyEvents` como **neNoSync**; isso significa que o evento `OnMessage` será **executado no contexto da thread da conexão**, então, se você precisar atualizar qualquer controle de um formulário ou acessar objetos compartilhados, deverá implementar seus próprios métodos de sincronização.

TsgcWebSocketClient | Receive Binary Messages

Quando o cliente recebe uma Mensagem Binária, o evento **OnBinary** é disparado. Leia o parâmetro Data para recuperar a mensagem binária recebida.

```
void OnBinary(TsgcWSConnection *Connection, const TMemoryStream *Data)
{
    oBitmap = new TBitmap();
    oBitmap->LoadFromStream(Data);
    Image1->Picture->Assign(oBitmap);
    Log(
        "#image uncompressed size: " + IntToStr(Data->Size) +
        ". Total received: " + IntToStr(Connection->RecBytes));
    delete oBitmap;
}
```

Por padrão, o cliente utiliza o método **neAsynchronous** para despachar o evento OnMessage, isso significa que **este evento é executado no contexto da Thread Principal**, portanto é thread-safe atualizar qualquer controle de um formulário, por exemplo.

Se o seu cliente recebe muitas mensagens ou você precisa controlar a sincronização com outras threads, defina a propriedade NotifyEvents como **neNoSync**, isso significa que o evento OnMessage será **executado no contexto da thread de conexão**, portanto, se você precisar atualizar qualquer controle de um formulário ou acessar objetos compartilhados, deve implementar seus próprios métodos de sincronização.

TsgcWebSocketClient | Client Authentication

O cliente TsgcWebSocket suporta 4 tipos de Autenticações:

- **Basic:** envia um Header HTTP durante o HandShake WebSocket com User e Password codificados como Basic Authentication.
- **Token:** envia um Token como um Cabeçalho HTTP durante o HandShake do WebSocket. Defina o token necessário em Authentication.Token.AuthToken conforme exigido pelo servidor.
- **Session:** primeiro o cliente solicita uma sessão HTTP do servidor e, se o servidor retornar uma sessão, ela é passada no GET HTTP Header do HandShake WebSocket. (* método de Autenticação próprio da biblioteca sgcWebSockets).
- **URL:** o cliente solicita autorização usando o cabeçalho HTTP GET do HandShake WebSocket. (* método de Autenticação próprio da biblioteca sgcWebSockets).

Authentication Basic

Este é um método de autorização simples no qual usuário e senha são codificados e passados como um Header HTTP. Basta definir o User e a Password e habilitar apenas o tipo de Authorization Basic para utilizar este método.

```
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 de autenticação

Permite que você autorize usando JWT. Isso exige que você obtenha um token usando uma ferramenta externa (por exemplo: uma conexão HTTP, OAuth2, etc.).

Se você anexar um componente OAuth2, pode obter este token automaticamente. Leia mais sobre [OAuth2](#).

Você deve definir seu AuthToken e habilitar a Token Authentication.

```
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;
```

Sessão de Autenticação

Primeiro o cliente conecta-se ao servidor utilizando uma conexão HTTP solicitando uma nova Session. Em caso de sucesso, o servidor retorna um SessionId e o cliente envia este SessionId no GET HTTP Header do HandShake WebSocket.

Requer a definição de UserName e Password e a habilitação da Session Authentication.

```
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;
```

Authentication URL

Este método de autenticação passa o nome de usuário e a senha no cabeçalho HTTP GET do 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 | Exceções do cliente

Às vezes há erros nas comunicações: o servidor pode desconectar uma conexão porque não está autorizada ou uma mensagem não tem o formato correto. Há 2 eventos onde os erros são capturados.

OnError

Este evento é disparado toda vez que há um erro no protocolo WebSocket, como tipo de mensagem inválido, string utf8 inválida...

```
void OnError(TsgcWSConnection *Connection, const string Error)
{
    WriteLn("#error: " + Error);
}
```

OnException

Este evento é disparado toda vez que há uma exceção, como escrever em um socket que não está ativo ou acessar um objeto que não existe.

```
void OnException(TsgcWSConnection *Connection, Exception *E)
{
    WriteLn("#exception: " + E->Message);
}
```

Por padrão, quando uma **conexão é fechada pelo servidor**, uma **exceção será gerada**. Se você não quiser que essas exceções sejam geradas, desabilite **Options.RaiseDisconnectExceptions**.

TsgcWebSocketClient | WebSocket HandShake

O protocolo WebSocket usa um HandShake HTTP para fazer o upgrade do protocolo HTTP para o protocolo WebSocket. Este handshake é tratado internamente pelo componente TsgcWebSocket Client, mas você pode adicionar seus próprios headers HTTP personalizados se o servidor exigir informações adicionais de header HTTP.

Exemplo: se você precisar adicionar este Cabeçalho HTTP "Client: sgcWebSockets"

```
void OnHandshake(TsgcWSConnection *Connection, ref TStringList *Headers)
{
    Headers->Add("Client: sgcWebSockets");
}
```

TsgcWebSocketClient | Client Register Protocol

Por padrão, o TsgcWebSocketClient não usa nenhum SubProtocol. Os subprotocolos WebSocket são construídos sobre o protocolo WebSocket e definem um protocolo de mensagens personalizado. Exemplos de subprotocolos WebSocket incluem MQTT, STOMP etc.

O nome do SubProtocolo WebSocket é enviado como um Cabeçalho HTTP no HandShake do WebSocket. Este cabeçalho é processado pelo servidor e, se o servidor suportar este subprotocolo, ele aceitará a conexão. Se não for suportado, a conexão será fechada automaticamente.

Exemplo: conecta a um servidor websocket com SubProtocol de nome 'myprotocol'

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

TsgcWebSocketClient | Client Proxies

O cliente TsgcWebSocket suporta conexões através de proxies. Para configurar uma conexão proxy, basta preencher as propriedades **Proxy** do cliente 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 é um componente de cliente TCP que permite conectar a servidores TCP simples.

Introdução

TsgcTCPClient é um componente cliente TCP que permite conectar a servidores TCP simples. Ele fornece suporte a conexões TLS/SSL, servidores proxy, monitoramento de heartbeat, reconexão automática (WatchDog) e TCP keep-alive.

1. Solte um componente **TsgcTCPClient** no formulário.
2. Defina **Host** e **Port** (o padrão é 80) para conectar-se a um TCP Server disponível.
3. Defina **TLS** como True se você precisar de uma conexão segura (o padrão é False).
4. Opcionalmente, configure o **Proxy** se estiver se conectando através de um servidor proxy.
5. Trate os eventos: **OnConnect**, **OnDisconnect**, **OnMessage**, **OnBinary**, **OnError**, **OnException**.
6. Defina **Active** como True para abrir a conexão.

Referência

- Propriedades — 23 propriedades disponíveis no componente.
- Métodos — 7 métodos públicos.
- Eventos — 12 eventos que você pode tratar.

TsgcTCPClient

TsgcTCPClient é um componente de cliente TCP que permite conectar a servidores TCP simples.

Introdução

TsgcTCPClient é um componente cliente TCP que permite conectar a servidores TCP simples. Ele fornece suporte a conexões TLS/SSL, servidores proxy, monitoramento de heartbeat, reconexão automática (WatchDog) e TCP keep-alive.

1. Solte um componente **TsgcTCPClient** no formulário.
2. Defina **Host** e **Port** (o padrão é 80) para conectar-se a um TCP Server disponível.
3. Defina **TLS** como True se você precisar de uma conexão segura (o padrão é False).
4. Opcionalmente, configure o **Proxy** se estiver se conectando através de um servidor proxy.
5. Trate os eventos: **OnConnect**, **OnDisconnect**, **OnMessage**, **OnBinary**, **OnError**, **OnException**.
6. Defina **Active** como True para abrir a conexão.

Referência

- Propriedades — 23 propriedades disponíveis no componente.
- Métodos — 7 métodos públicos.
- Eventos — 12 eventos que você pode tratar.

TsgcWebSocketServer

TsgcWebSocketServer implementa um Componente WebSocket Servidor e pode tratar múltiplas conexões de clientes em threads.

Introdução

TsgcWebSocketServer implementa o componente servidor WebSocket e pode tratar múltiplas conexões de cliente em threads. Siga os passos abaixo para configurar este componente:

1. Coloque um componente TsgcWebSocketServer no formulário
2. Defina a Port (o padrão é 80). Se você estiver atrás de um firewall, provavelmente precisará configurá-lo.
3. Defina as Specifications permitidas; por padrão, todas as especificações são permitidas.

RFC6455: é a especificação WebSocket padrão e recomendada.

Hixie76: é um rascunho e só é recomendado estabelecer conexões Hixie76 se você quiser fornecer suporte a navegadores antigos como o Safari 4.2

```
void __fastcall TForm1::OnTCPConnectEvent(TsgcWSConnection *aConnection, bool &Accept)
{
    aConnection->Transport = trpTCP;
    Accept = true;
}
```

5. Crie um procedimento e defina a propriedade Active = True.

Referência

- Propriedades — 27 propriedades disponíveis no componente.
- Métodos — 7 métodos públicos.
- Eventos — 22 eventos que você pode tratar.

Guides

- **Start**
 - [Iniciar servidor](#)
 - [Bindings do servidor](#)
 - [Inicialização - Desligamento do servidor](#)
 - [Manter servidor ativo](#)
- **Conexões**
 - [Server Keep Connections Alive](#)
 - [Server Plain TCP](#)
 - [Server Close Connection](#)
 - [Client Connections](#)
- **Autenticação**

- [Server Authentication](#)
- **Send Messages**
 - [Server Send Text Message](#)
 - [Server Send Binary Message](#)
- **Receber Mensagens**
 - [Servidor Receber Mensagem de Texto](#)
 - [Servidor Receber Mensagem Binária](#)
- **SSL**
 - [Server SSL](#)
 - [Server SSL SChannel](#)

TsgcWebSocketServer | Iniciar Servidor

A primeira coisa que você deve definir quando quer iniciar um servidor é a porta de escuta. Por padrão, ela é definida como porta 80, mas você pode alterá-la para qualquer porta. Uma vez que a porta esteja definida, há 2 métodos para iniciar um servidor.

Propriedade Active

Se você definir a propriedade Active como true, o servidor começará a escutar todas as conexões de entrada na porta configurada.

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

Se você definir a propriedade Active como false, o servidor irá parar e fechar todas as conexões ativas.

```
oServer->Active = false;
```

Métodos Start / Stop

Embora definir a propriedade Active inicie/pare o servidor na mesma thread, os métodos Start e Stop são executados em uma thread secundária.

```
oServer = new TsgcWebSocketServer();  
oServer->Port = 80;  
oServer->Start();
```

Se você chamar o método Stop(), o servidor parará e fechará todas as conexões ativas.

```
oServer->Stop();
```

Você pode utilizar o método **ReStart**, para Parar e Iniciar o servidor em uma thread secundária.

Se você alterar a Port após fechar um servidor, para começar a escutar em uma porta diferente, chame o método **Bindings.Clear()** após fechar o servidor para excluir todos os bindings anteriores. Caso contrário, o servidor tentará vincular-se aos bindings anteriores.

TsgcWebSocketServer | Server Bindings

Por padrão, se você preencher apenas a **propriedade Port**, o servidor **vincula a porta de escuta em TODOS os IPs**, então se por exemplo você tiver 3 IPs: 127.0.0.1, 80.54.11.22 e 12.55.41.17, seu servidor vinculará esta porta em todos os 3 IPs.

Geralmente é recomendado fazer o bind apenas aos IPs necessários. É aqui que você pode utilizar a propriedade Bindings.

Em vez de utilizar a propriedade Port, basta utilizar a propriedade Bindings e preencher o IP e a Port necessários.

Exemplo: vincula a Porta 5555 ao IP 127.0.0.1 e ao IP 80.58.25.40

```
oServer = new TsgcWebSocketServer();
binding = oServer->Bindings->Add();
binding->IP = "127.0.0.1";
binding->Port = 5555;
binding = oServer->Bindings->Add();
binding->IP = "80.58.25.40";
binding->Port = 5555;
oServer->Active = true;
```

Se você alterar a Port após fechar um servidor, para começar a escutar em uma porta diferente, chame o método **Bindings.Clear()** após fechar o servidor para excluir todos os bindings anteriores. Caso contrário, o servidor tentará vincular-se aos bindings anteriores.

TsgcWebSocketServer | Inicialização e Encerramento do Servidor

Depois de definir todas as configurações necessárias do seu servidor, há 2 eventos úteis para saber quando o servidor foi iniciado e quando foi parado.

OnStartup

Este evento é disparado quando o servidor foi iniciado e pode processar novas conexões.

```
void OnStartup(TObject *Sender)
{
    WriteLn("#server started");
}
```

OnShutdown

Este evento é disparado após o servidor ter parado e nenhuma outra conexão ser aceita.

```
void OnShutdown(TObject *Sender)
{
    WriteLn("#server stopped");
}
```

TsgcWebSocketServer | Server Keep Active

Uma vez que o servidor é iniciado, às vezes ele pode parar por qualquer motivo. Se você quiser reiniciar o servidor após um shutdown inesperado, você pode utilizar a propriedade WatchDog.

WatchDog

Se o WatchDog estiver habilitado, quando o servidor detectar um desligamento, o WatchDog tenta reiniciar a cada X segundos até que o servidor esteja ativo novamente.

Exemplo: reiniciar a cada 10 segundos após uma parada inesperada, com tentativas ilimitadas.

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


TsgcWebSocketServer | Server SSL

O servidor pode ser configurado para utilizar **Certificados SSL**. Para obter um servidor de produção com um certificado de servidor, você deve **adquirir** um certificado de um **provedor conhecido**: Namecheap, GoDaddy, Thawte, etc. Para **fins de teste**, você pode utilizar um **certificado autoassinado** (verifique o exemplo Demos/Chat, que utiliza um certificado autoassinado).

O certificado deve estar no **formato PEM**, o PEM (de Privacy Enhanced Mail) é definido nas RFCs 1421 a 1424; é um formato de container que pode incluir apenas o certificado público (como em instalações do Apache e arquivos de certificado CA /etc/ssl/certs), ou pode incluir uma cadeia de certificados completa, incluindo chave pública, chave privada e certificados raiz. Para criar um único certificado pem, basta abrir seu arquivo de chave privada, copiar o conteúdo e colá-lo no arquivo de certificado.

Exemplo:

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-----
```

Para habilitar SSL, basta **habilitar a propriedade SSL** e configurar os caminhos para **CertFile**, **KeyFile** e **RootFile**. Se o certificado contém toda a cadeia de certificados (chave pública, chave privada, etc.), basta definir todos os caminhos para o mesmo certificado.

Outra propriedade que você deve definir é **SSLOptions.Port**, esta é a porta usada para conexões seguras.

Configuração SSL simples

Exemplo: configure SSL no IP 127.0.0.1 e Port 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 and Non-SSL

Você pode configurar o servidor para escutar em mais de um IP e porta; confira o [artigo sobre Binding](#), que explica como funciona. O servidor pode ser configurado para permitir conexões SSL e conexões não-SSL ao mesmo tempo (é claro, escutando em portas diferentes). Você só precisa vincular a duas portas diferentes e configurar uma porta para conexões SSL e outra porta para conexões não-SSL.

Exemplo: configurar o servidor no IP 127.0.0.1, porta 80 (não criptografada) e 443 (SSL)

```
oServer = new TsgcWebSocketServer();
bind = oServer->Bindings->Add();
bind->IP = "127.0.0.1";
bind->Port = 80;
bind = oServer->Bindings->Add();
bind->IP = "127.0.0.1";
bind->Port = 443;
oServer->Port = 80;
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->Active = true;
```

Server SSL | SChannel for Indy Servers

Os componentes de servidor baseados em Indy (**TsgcWebSocketServer**, **TsgcWebsocketHTTPServer**) podem usar **Windows SChannel** (Secure Channel) como provedor TLS em vez do OpenSSL. O SChannel é a implementação TLS nativa do Windows, portanto não requer nenhuma DLL externa.

Como funciona

Quando um componente servidor tem SSL habilitado e o IOHandler está definido como **iohSChannel**, o servidor cria uma instância **TsgcIdServerIOHandlerSSLSChannel** que trata de todas as operações TLS utilizando a API SChannel do Windows. Para cada conexão de cliente recebida, o servidor realiza o handshake TLS através do provedor SChannel, negociando a versão do protocolo, a suíte de cifras e vinculando o certificado configurado. O SChannel lê os certificados do **Windows Certificate Store** ou de um **arquivo PFX** (.pfx / .p12). Nenhum arquivo PEM é necessário e nenhuma biblioteca OpenSSL precisa ser implantada.

Configuração

Para habilitar o SChannel em um servidor Indy, configure as seguintes propriedades:

1. Defina a propriedade **SSL** como **True**.
2. Defina **SSLOptions.IOHandler** como **iohSChannel**.
3. Defina **SSLOptions.Version** com a versão TLS desejada (tls1_2, tls1_3, ...).
4. Defina **SSLOptions.Port** com a porta usada para conexões seguras.
5. Configure o certificado usando um dos dois métodos descritos abaixo.

Propriedades de SChannel_Options

A subpropriedade **SSLOptions.SChannel_Options** expõe as configurações específicas do SChannel:

Property	Descrição
CertHash	O thumbprint (hash hexadecimal) de um certificado instalado no Windows Certificate Store.
CertStoreName	Qual repositório de certificados pesquisar: scsnMY (Personal), scsnRoot , scsnTrust , scsnCA .
CertStorePath	Local do armazenamento: scspStoreLocalMachine ou scspStoreCurrentUser .
CipherList	Lista opcional separada por dois pontos de algoritmos de cifra (por exemplo, CALG_AES_256:CALG_AES_128). Vazio significa usar os padrões do sistema.
UseLegacyCredentials	Quando True, utiliza a estrutura legada SCHANNEL_CRED em vez de SCH_CREDENTIALS. Habilite isto para versões mais antigas do Windows que não suportam a API mais recente.

Além disso, as propriedades gerais de **SSLOptions CertFile** e **Password** são usadas ao carregar um certificado de um arquivo PFX.

Certificado do Windows Store

Se o certificado já estiver instalado no Windows Certificate Store, forneça o **thumbprint** do certificado e indique onde ele está localizado.

Para encontrar o thumbprint, abra o **PowerShell** e execute:

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

A coluna **Thumbprint** mostra o hash hexadecimal de que você precisa.

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

```
oServer = new TsgcWebSocketServer();
oServer->SSL = true;
oServer->SSLOptions->IOHandler = iohSChannel;
oServer->SSLOptions->Version = tls1_2;
oServer->SSLOptions->Port = 443;
oServer->Port = 443;
oServer->SSLOptions->SChannel_Options->CertHash = "C12A8FC8AE668F866B48F23E753C93D357E9BE10";
oServer->SSLOptions->SChannel_Options->CertStoreName = scsnMY;
oServer->SSLOptions->SChannel_Options->CertStorePath = scspStoreLocalMachine;
oServer->Active = true;
```

Certificado a partir de arquivo PFX

Se você tem um arquivo de certificado PFX (.pfx ou .p12), defina as propriedades **CertFile** e **Password** em **SSLOptions**. O **SChannel** importará o certificado na inicialização.

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

Se você possui um certificado PEM e uma chave privada, converta-os primeiro para o formato PFX utilizando **OpenSSL**:

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

Versão TLS

Utilize a propriedade **SSLOptions.Version** para controlar qual versão do TLS o servidor aceita:

Value	Descrição
tls1_0	TLS 1.0 (não recomendado)
tls1_1	TLS 1.1
tls1_2	TLS 1.2 (recomendado)
tls1_3	TLS 1.3
tlsUndefined	Aceitar TLS 1.0, 1.1 e 1.2

Cipher List

Por padrão, o **SChannel** utiliza a configuração de cifras do sistema. Você pode restringir as cifras permitidas definindo **SChannel_Options.CipherList** como uma lista de nomes de algoritmos separados por dois-pontos, por exemplo:

```
CALG_AES_256:CALG_AES_128
```

Deixe esta propriedade vazia para utilizar os padrões do Windows.

Legacy Credentials

O Windows Server 2019 e anteriores podem não suportar a nova API **SCH_CREDENTIALS**. Se o servidor falhar ao iniciar em uma versão mais antiga do Windows, defina **SChannel_Options.UseLegacyCredentials** como **True** para utilizar a estrutura legada **SCHANNEL_CRED** em vez dela.

O componente detecta a versão do Windows automaticamente na maioria dos casos, mas você pode forçar o modo legacy se necessário.

Vantagens em relação ao OpenSSL

- **Sem DLLs externas:** o SChannel está integrado ao Windows, portanto você não precisa implantar libeay32.dll / ssleay32.dll ou libcrypto / libssl.
- **Integração com o Windows Certificate Store:** utilize certificados já instalados e gerenciados pelo sistema operacional.
- **Atualizações automáticas:** melhorias de TLS e patches de segurança são aplicados por meio do Windows Update.

Notas

- O SChannel está disponível **somente no Windows**. Para servidores multiplataforma, utilize o OpenSSL (iohOpenSSL).
- O servidor deve ter a **chave privada** associada ao certificado. Ao usar o método Windows Store, o certificado deve ter sido importado com sua chave privada.
- Para servidores de produção que utilizam o método Certificate Store, o armazenamento **Local Machine** (scspStoreLocalMachine) é recomendado para que o certificado esteja disponível independentemente de qual conta de usuário executa o serviço.
- Somente arquivos de certificado **PFX** (.pfx / .p12) são suportados. Se você tem arquivos PEM, converta-os para o formato PFX primeiro.

TsgcWebSocketServer | Servidor verificar certificado

Por padrão, o servidor não verifica os certificados de peer. Para configurar o servidor para verificar os certificados do cliente, implemente os seguintes passos:

1. Defina a propriedade `SSLOptions.VerifyCertificate = true`

Trate o evento `OnSSLVerifyPeer` e implemente o código a seguir para ser notificado sempre que um cliente se conectar com um certificado.

```
void OnSSLSSLVerifyPeer(System::TObject* Sender, TIdX509* Certificate, bool &Accept)
{
    // ... validate the certificate
    if (Certificate_OK)
        Accept = true;
    else
        Accept = false;
}
```

Observe que o evento `OnSSLVerifyPeer` é chamado apenas se o cliente fornecer um certificado; se um cliente não fornecer um certificado, o evento não é disparado.

Você pode configurar o **servidor para permitir apenas conexões SSL que utilizem um certificado**. Para isso, defina a seguinte propriedade:

- `SSLOptions.VerifyCertificate_Options.FailIfNoCertificate = true`

Se o cliente não fornecer um certificado, a conexão será fechada no SSL Handshake.

TsgcWebSocketServer | Server Keep Connections Alive

Uma vez que um cliente se conectou ao servidor, às vezes a conexão pode ser fechada devido a sinal fraco, erros de conexão, etc. Use o HeartBeat para manter a conexão ativa.

HeartBeat

A propriedade **HeartBeat** permite que você **envie um Ping** a cada **X segundos** para **manter a conexão ativa**. Alguns clientes fecham conexões TCP se não houver dados trocados entre os peers. O HeartBeat resolve esse problema enviando um ping em um intervalo específico. Geralmente isso é suficiente para manter uma conexão ativa, mas você pode definir um intervalo de TimeOut se quiser fechar a conexão quando uma resposta do cliente não for recebida após X segundos.

Example: envia um ping a todos os clientes conectados a cada 30 segundos

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

TsgcWebSocketServer | Servidor TCP Simples

O servidor WebSocket aceita WebSocket, HTTP, SSE e outros protocolos, mas também pode trabalhar com conexões TCP simples. Leia mais sobre [conexões TCP](#).

Há 2 eventos que podem ser usados para tratar conexões TCP.

OnTCPConnect

Este evento é chamado depois que um cliente se conecta ao servidor e antes de qualquer handshake entre cliente e servidor. O evento OnConnect só é disparado depois que o cliente envia uma mensagem (para permitir que o servidor detecte qual protocolo está sendo usado).

Este evento permite que você saiba que um novo cliente está tentando conectar ao servidor, e o servidor pode aceitar ou rejeitar a conexão. Por padrão, o servidor sempre aceita a conexão.

OnUnknownProtocol

Este evento é chamado quando o servidor recebe a primeira mensagem de um cliente, mas não consegue detectar se ela corresponde a algum dos protocolos conhecidos. Neste evento, o servidor pode aceitar ou rejeitar o protocolo.

OnConnect

Este evento é disparado após uma conexão bem-sucedida e completa. Se a conexão for TCP simples, ele é disparado depois que o protocolo é aceito no evento OnUnknownProtocol.

TsgcWebSocketServer | Servidor Fechar Conexão

Uma única Connection pode ser fechada utilizando os métodos Close ou Disconnect.

Disconnect

O [TsgcWSConnection](#) tem um método chamado Disconnect(), que permite desconectar a conexão no nível de socket. Se você chamar este método, o socket será desconectado diretamente sem aguardar qualquer resposta do cliente. Você pode enviar um Close Code com este método.

Close

[TsgcWSConnection](#) possui um método chamado Close(), que permite enviar uma mensagem ao cliente solicitando o fechamento da conexão. Se o cliente receber esta mensagem, ele deve fechar a conexão e o servidor receberá uma notificação de que a conexão foi fechada. Você pode enviar um Close Code com este método.

DisconnectAll

Desconecta todas as conexões ativas. Este método é chamado automaticamente antes de o servidor parar de escutar, mas você pode chamar este método a qualquer momento.

TsgcWebSocketServer | Client Connections

Para acessar as conexões de cliente ativas, você pode usar a propriedade `Connections` para iterar pela lista e acessar a classe de conexão do cliente. A propriedade `Connections` acessa uma lista com thread, portanto primeiro trava a lista e, quando terminar, destrava a lista.

```
void DoClientIPAddresses() {
    TList *oList;
    TsgcWSConnectionServer *oConnection;
    oList = TsgcWebSocketHTTPServer1->LockList();
    try {
        for (int i = 0; i < oList->Count; ++i) {
            oConnection = dynamic_cast<TsgcWSConnectionServer*>(static_cast<TIdContext*>(oList->Items[i])->Data);
            ShowMessage(oConnection->IP + ":" + IntToStr(oConnection->Port));
        }
    } __finally {
        TsgcWebSocketHTTPServer1->UnlockList();
    }
}
```

TsgcWebSocketServer | Server Authentication

O servidor TsgcWebSocket suporta 3 tipos de Autenticações:

- **Basic:** lê um cabeçalho HTTP durante o HandShake WebSocket com User e Password codificados como Basic Authorization.
- **Session:** primeiro o cliente solicita uma sessão HTTP ao servidor e, se o servidor retornar uma sessão, ela é passada no GET HTTP Header do HandShake WebSocket. (* método de autorização próprio da biblioteca sgcWebSockets).
- **URL:** lê a autorização da requisição utilizando o cabeçalho HTTP GET do HandShake WebSocket. (* método de autorização próprio da biblioteca sgcWebSockets).

Você pode definir uma lista de usuários autenticados utilizando a propriedade **AuthUsers**; basta definir seus usuários com o seguinte formato: user=password

OnAuthentication

Sempre que o servidor recebe uma Requisição de Autenticação de um cliente, este evento é chamado para indicar se o usuário está autenticado ou não.

Utilize o parâmetro Authenticated para aceitar ou não a conexão.

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

OnUnknownAuthentication

Referencia o TsgcWebSocketServer que entrega o subprotocolo de sinalização WebRTC sobre conexões WebSocket.

```
void OnUnknownAuthentication(TsgcWSConnection *Connection, string AuthType, string AuthData,
    ref string aUser, ref string aPassword, ref bool Authenticated)
{
    if (AuthType == "Bearer")
    {
        if (AuthData == "jwt_token")
        {
            Authenticated = true;
        }
        else
        {
            Authenticated = false;
        }
    }
    else
    {
        Authenticated = false;
    }
}
```


TsgcWebSocketServer | Server Send Text Message

Uma vez que o cliente tenha se conectado ao servidor, o servidor pode enviar mensagens de texto. Para enviar uma Mensagem de Texto, chame o método `WriteData()` para enviar uma mensagem a um único cliente, ou use `Broadcast` para enviar uma mensagem a todos os clientes.

Enviar uma Mensagem de Texto

Chame o método **`WriteData()`** para enviar uma mensagem de texto.

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

Se **`QueueOptions.Text`** tem um **valor diferente de `qmNone`**, em vez de ser processada na mesma thread que é chamada, ela será processada em uma thread secundária. Por padrão, esta opção está desabilitada.

QueueOptions não funciona se a propriedade `IOHandlerOptions.IOHandlerType = iohIOCP` (devido à arquitetura IOCP, este recurso não é suportado).

Você também pode chamar o método `WriteData()` de **`TsgcWSConnection`**; **exemplo:** enviar uma mensagem a um cliente quando ele se conecta ao servidor.

```
void OnConnect(TsgcWSConnection *Connection)
{
    Connection->WriteData("Hello From Server");
}
```

Envia uma mensagem a TODOS os clientes conectados

Chame o método **`Broadcast()`** para enviar uma mensagem de texto a todos os clientes conectados.

```
TsgcWebSocketServer1->Broadcast("Hello From Server");
```

TsgcWebSocketServer | Server Send Binary Message

Depois que o cliente conecta ao servidor, o servidor pode enviar mensagens binárias. Para enviar uma Mensagem Binária, chame o método `WriteData()` para enviar uma mensagem a um único cliente, ou utilize `Broadcast` para enviar uma mensagem a todos os clientes.

Enviar uma Mensagem Binária

Chame o método **`WriteData()`** para enviar uma mensagem binária.

```
TsgcWebSocketServer1->WriteData("guid", new TMemoryStream());
```

Se **`QueueOptions.Binary`** tem um **valor diferente de `qmNone`**, em vez de ser processado na mesma thread que é chamada, ele será processado em uma thread secundária. Por padrão, esta opção está desabilitada.

QueueOptions não funciona se a propriedade `IOHandlerOptions.IOHandlerType = iohIOCP` (devido à arquitetura IOCP, este recurso não é suportado).

Você também pode chamar o método `WriteData()` de **`TsgcWSConnection`**; **exemplo:** enviar uma mensagem a um cliente quando ele se conecta ao servidor.

```
void OnConnect(TsgcWSConnection *Connection)
{
    Connection->WriteData(new TMemoryStream());
}
```

Envia uma mensagem a TODOS os clientes conectados

Chame o método **`Broadcast()`** para enviar uma mensagem binária para todos os clientes conectados.

```
TsgcWebSocketServer1->Broadcast(new TMemoryStream());
```

TsgcWebSocketServer | Servidor receber mensagem de texto

Quando o servidor recebe uma Mensagem de Texto, o evento **OnMessage** é disparado. Leia o parâmetro Text para recuperar a string da mensagem recebida.

```
void OnMessage(TsgcWSConnection *Connection, const string Text)
{
    ShowMessage("Message Received from Client: " + Text);
}
```

Por padrão, o servidor utiliza o método **neAsynchronous** para despachar o evento OnMessage, o que significa que **este evento é executado no contexto da Thread Principal**, portanto é thread-safe atualizar qualquer controle de um form, por exemplo.

Se seu servidor recebe muitas mensagens ou você precisa controlar a sincronização com outras threads, defina a propriedade NotifyEvents como **neNoSync**, isto significa que o evento OnMessage será **executado no contexto da thread da conexão**, então se você precisar atualizar qualquer controle de um formulário ou acessar objetos compartilhados, você deve implementar seus próprios métodos de sincronização.

TsgcWebSocketServer | Servidor Recebe Mensagem Binária

Quando o servidor recebe uma mensagem binária, o evento **OnBinary** é disparado. Leia o parâmetro *Data* para recuperar a mensagem binária recebida.

```
void OnBinary(TsgcWSConnection *Connection, const TMemoryStream *Data)
{
    oBitmap = new TBitmap();
    oBitmap->LoadFromStream(Data);
    Image1->Picture->Assign(oBitmap);
    Log(
        "#image uncompressed size: " + IntToStr(Data->Size) +
        ". Total received: " + IntToStr(Connection->RecBytes));
    oBitmap.Free();
}
```

Por padrão, o servidor utiliza o método **neAsynchronous** para despachar o evento *OnMessage*, o que significa que **este evento é executado no contexto da Thread Principal**, portanto é thread-safe atualizar qualquer controle de um form, por exemplo.

Se seu servidor recebe muitas mensagens ou você precisa controlar a sincronização com outras threads, defina a propriedade *NotifyEvents* como **neNoSync**, isto significa que o evento *OnMessage* será **executado no contexto da thread da conexão**, então se você precisar atualizar qualquer controle de um formulário ou acessar objetos compartilhados, você deve implementar seus próprios métodos de sincronização.

TsgcWebSocketServer | Server Read Headers from Client

Quando um **cliente se conecta** ao servidor WebSocket, ele envia uma lista de **headers** com informações sobre a conexão do cliente. Para ler esses headers do cliente, você pode usar o evento **OnHandshake** do componente servidor, que é chamado quando o servidor recebe os headers do cliente e antes de enviar uma resposta. Os cabeçalhos do cliente são armazenados na propriedade **HeadersRequest** de **TsgcWSConnectionServer**.

```
void OnServerHandshake(TsgcWSConnection *Connection, TStringList *Headers)
{
    ShowMessage(dynamic_cast<TsgcWSConnectionServer*>(Connection)->HeadersRequest->Text);
}
```

TsgcWebSocketHTTPServer

AResponseInfo.

Introdução

TsgcWebSocketHTTPServer implementa um Componente WebSocket Servidor e pode tratar múltiplas conexões de clientes em threads, assim como [TsgcWebSocketServer](#), e permite servir páginas HTML usando um Servidor HTTP integrado, compartilhando a mesma porta para conexões WebSocket e requisições HTTP.

Siga os passos abaixo para configurar este componente:

1. Coloque um componente TsgcWebSocketHTTPServer no formulário
2. Defina a Port (o padrão é 80). Se você estiver atrás de um firewall, provavelmente precisará configurá-lo.
3. Defina as Specifications permitidas; por padrão, todas as especificações são permitidas.

RFC6455: é a especificação WebSocket padrão e recomendada.

Hixie76: é um rascunho e só é recomendado estabelecer conexões Hixie76 se você quiser fornecer suporte a navegadores antigos como o Safari 4.2

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

OnHTTPUploadBeforeCreatePostStream: este evento é chamado depois que os cabeçalhos foram lidos e antes de o post stream ser criado.

* Em alguns casos, você pode obter um alto consumo de cpu devido a conexões não solicitadas; nesses casos, basta retornar um erro 500 se for uma requisição HTTP ou fechar a conexão para requisições de Protocolo Desconhecido (Unknown Protocol).

5. Crie um procedimento e defina a propriedade Active = true.

Referência

- Propriedades — 33 propriedades disponíveis no componente.
- Methods — 9 métodos públicos.
- Eventos — 36 eventos que você pode tratar.

Guides

- HTTP
 - [HTTP Server Requests](#)
 - [HTTP Dispatch Files](#)
 - [HTTP/2 Server](#)
 - [HTTP/2 Server Push](#)
 - [HTTP/2 Alternate Service](#)
 - [HTTP/2 Server Threads](#)
 - [HTTP Post Big Files](#)
 - [HTTP 404 Error without Response Body](#)

- **SSL**
 - [Server SSL SChannel](#)
- **Outros**
 - [Sessões do servidor HTTP](#)

TsgcWebSocketHTTPServer | Requisições do servidor HTTP

Use `OnCommandGet` para tratar as requisições de clientes HTTP. Use os seguintes parâmetros:

- **RequestInfo:** contém informações da requisição HTTP.
- **ResponseInfo:** é a resposta HTTP à Requisição HTTP.
 - **ContentText:** é a resposta em formato de texto.
 - **ContentType:** é o tipo do Content-Type.
 - **ResponseNo:** número da resposta HTTP, exemplo: 200.

```
void OnCommandGet(TIdContext *AContext, TIdHTTPRequestInfo *ARequestInfo,
  TIdHTTPResponseInfo *AResponseInfo)
{
  if (ARequestInfo->Document == "/")
  {
    AResponseInfo->ContentText = "<html><head><title>Test Page</title></head><body></body></html>";
    AResponseInfo->ContentType = "text/html";
    AResponseInfo->ResponseNo = 200;
  }
}
```

OnBeforeCommand

Utilize este evento para personalizar a resposta HTTP. Por exemplo, se você quiser que alguns endpoints utilizem um esquema de autorização enquanto outros possam ser acessados sem autorização, utilize o parâmetro `options` para permitir ou desabilitá-lo. Abaixo está um exemplo em que a Authorization Basic está habilitada, mas quando um usuário requisita o endpoint `/public`, a autorização não é necessária.

```
void __fastcall TForm1::OnBeforeCommand(TsgcWSConnection* aConnection, TIdHTTPRequestInfo* ARequestInfo, TIdHTTPF
{
  if (ARequestInfo->Document == "/public")
    aOptions << hcoAuthorizedBasic;
}
```

TsgcWebSocketHTTPServer | HTTP Dispatch Files

Quando um cliente solicita um arquivo, o evento **OnCommandGet** é disparado, mas você pode utilizar a propriedade **DocumentRoot** para despachar arquivos automaticamente.

Exemplo: se você definir **DocumentRoot** como **c:/www/files**. Toda vez que um novo arquivo for solicitado, o servidor procurará nesta pasta. Se o arquivo existir, ele será despachado automaticamente.

TsgcWebSocketHTTPServer | Servidor HTTP/2

O servidor HTTP do sgcWebSockets permite que você trate requisições HTTP/1.1 e HTTP/2.0. Você pode habilitar o protocolo HTTP/2 usando HTTP2Options do servidor.

Defina **HTTP2Options.Enabled = true** para permitir que o servidor aceite requisições do protocolo HTTP/2. As requisições podem ser processadas pelo usuário exatamente da mesma forma que com o protocolo HTTP/1.1, [leia mais](#).

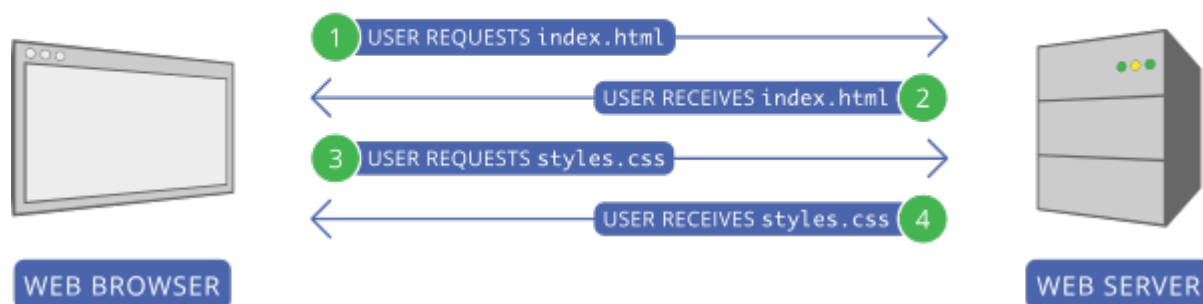
Quando o protocolo HTTP/2 está habilitado, o servidor ainda suportará requisições HTTP/1.1 .

Por padrão, os eventos OnConnect e OnDisconnect não serão chamados quando houver uma nova conexão HTTP/2, mas isso pode ser modificado acessando as propriedades HTTP2Options.Events, onde você pode personalizar se deseja ser notificado toda vez que houver uma nova conexão e/ou desconexão HTTP/2.

TsgcWebSocketHTTPServer | HTTP/2 Server Push

O HTTP geralmente funciona com um padrão de Requisição/Resposta, onde o cliente envia uma REQUEST por um recurso ao SERVER, e o SERVER envia uma RESPONSE com o recurso solicitado ou um erro. Normalmente o cliente, como um navegador, faz uma série de requisições por recursos que são fornecidos pelo servidor.

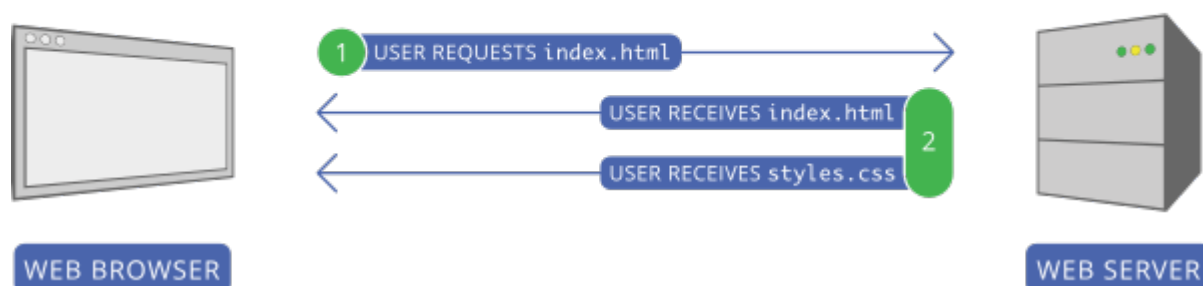
TYPICAL WEB SERVER COMMUNICATION



O principal problema com essa abordagem é que o cliente deve primeiro enviar uma requisição para obter o recurso (por exemplo, index.html), aguardar até que o servidor envie a resposta, ler o conteúdo e então fazer todas as outras requisições (por exemplo, styles.css).

O server push do HTTP/2 tenta resolver esse problema. Quando o cliente requisita um arquivo, se o servidor determinar que esse arquivo requer arquivos adicionais, esses arquivos serão ENVIADOS (PUSHED) automaticamente ao cliente.

WEB SERVER COMMUNICATION WITH HTTP/2 SERVER PUSH



Na captura de tela acima, o cliente primeiro requisita index.html. O servidor lê esta requisição e envia 2 arquivos como resposta: index.html e styles.css, evitando assim uma segunda requisição para obter styles.css.

Configurar Server Push

Seguindo as capturas de tela acima, você pode configurar seu servidor para que, toda vez que houver uma nova requisição para o arquivo /index.html, o servidor envie index.html e styles.css.

Use o método **PushPromiseAddPreLoadLinks** para associar cada requisição a uma lista de push promise.

```
TsgcWebSocketHTTPServer *server = new TsgcWebSocketHTTPServer(this);
TStringList *oLinks = new TStringList();
try
{
```

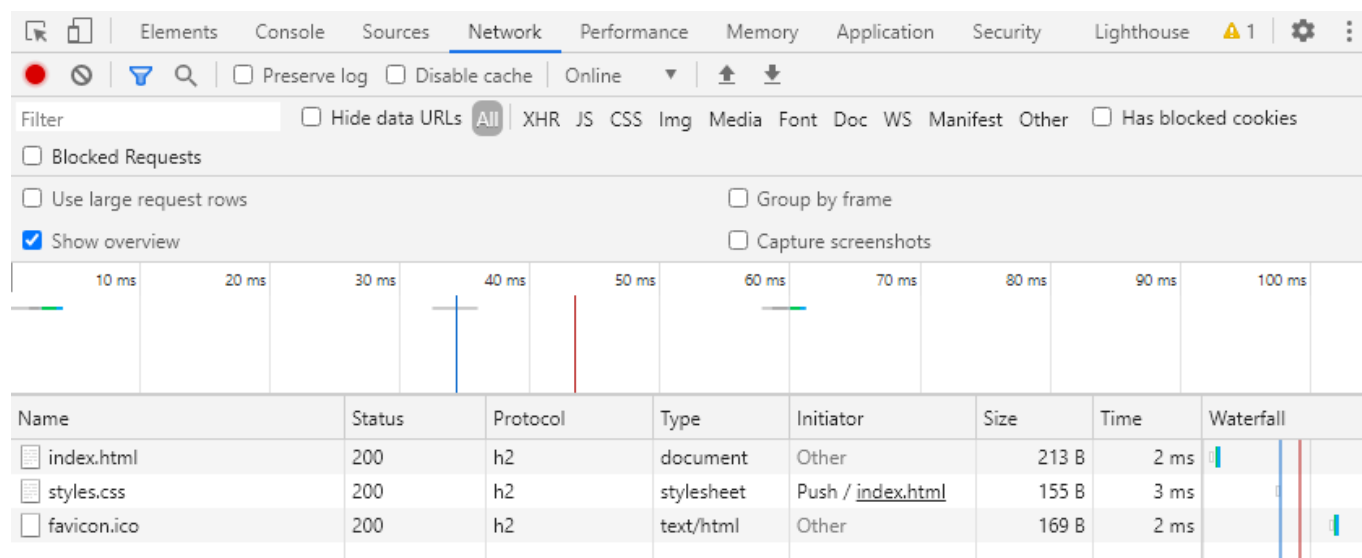
```

oLinks->Add("/styles.css");
server->PushPromiseAddPreLoadLinks("/index.html", oLinks);
}
__finally
{
    oLinks->Free();
}

void OnCommandGet(TidContext *AContext, TidHttpRequestInfo *ARequestInfo, TidHttpResponseInfo *AResponseInfo)
{
    if (ARequestInfo->Document == "/index.html")
    {
        AResponseInfo->ContentText = "";
        AResponseInfo->ContentType = "text/html";
        AResponseInfo->ResponseNo = 200;
    }
    else if (ARequestInfo->Document == "/styles.css")
    {
        AResponseInfo->ContentText = "";
        AResponseInfo->ContentType = "text/css";
        AResponseInfo->ResponseNo = 200;
    }
}

```

Utilizando a ferramenta de desenvolvedor do Chrome, você pode ver como o arquivo styles.css é enviado ao cliente.



TsgcWebSocketHTTPServer | HTTP/2 Alternate Service

O cabeçalho HTTP **Alt-Svc** é usado para **informar** aos clientes que o **mesmo recurso pode ser alcançado a partir de outro serviço ou protocolo**; isto é útil se você quiser informar aos clientes HTTP que seu servidor suporta HTTP/2, por exemplo.

Exemplo: se o seu servidor estiver em execução em um IP local 127.0.0.1 e estiver escutando em 2 portas: 80 (não criptografada) e 443 (criptografada). Você pode informar aos clientes que o HTTP/2 é suportado na porta 443 utilizando o seguinte header HTTP

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

Quando o HTTP/2 está habilitado, este cabeçalho é adicionado automaticamente se a conexão não estiver rodando no protocolo HTTP/2.

Você pode habilitar ou desabilitar este recurso utilizando a propriedade **HTTP2Options.AltSvc**.

TsgcWebSocketHTTPServer | HTTP/2 Server Threads

Veja abaixo as diferenças entre HTTP 1.1 e HTTP 2.0:

HTTP 1.1

No comportamento tradicional do HTTP, ao fazer múltiplas requisições pela mesma conexão, o cliente precisa esperar pela resposta de cada requisição antes de enviar a próxima. Essa abordagem sequencial aumenta significativamente o tempo de carregamento dos recursos de um site. Para resolver esse problema, o HTTP/1.1 introduziu um recurso chamado **pipelining**, permitindo que um cliente envie múltiplas requisições sem esperar pelas respostas do servidor. O servidor, por sua vez, responde ao cliente na mesma ordem em que recebeu as requisições.

Embora o **pipelining** parecesse ser uma solução, ele enfrentou desafios:

- **Ignorância do Servidor ou Corrupção de Resposta:** Alguns servidores ignoravam requisições em **pipeline** ou corrompiam as respostas, levando a uma comunicação não confiável.
- **Head-of-Line Blocking:** A primeira requisição no **pipeline** poderia bloquear as requisições subsequentes, causando um atraso no processamento de outras requisições. Esse fenômeno, conhecido como **head-of-line blocking**, resultava em tempos de carregamento de página mais lentos.

Em um esforço para otimizar o carregamento de páginas a partir de servidores que suportam HTTP/1.1, os navegadores web implementaram uma solução alternativa. Eles abrem de seis a oito conexões paralelas com o servidor, permitindo a transmissão simultânea de múltiplas requisições. Esse paralelismo visa mitigar os problemas associados ao **pipelining** e melhorar os tempos gerais de carregamento da página.

A escolha de seis a oito conexões paralelas pelos navegadores web baseia-se em considerações de otimização. As razões específicas por trás da seleção desse número podem envolver um equilíbrio entre utilização de recursos, eficiência de rede e a prevenção de potenciais gargalos.

HTTP 2.0

Em resposta às restrições encontradas no **pipelining**, o HTTP/2 introduziu um recurso chamado **multiplexing**. O **Multiplexing** permite uma **comunicação mais eficiente** entre o cliente e o servidor ao habilitar a **transmissão concorrente de múltiplas requisições** e respostas **sobre uma única conexão**.

O HTTP/2 utiliza um mecanismo de **framing binário**, o que significa que as mensagens HTTP são divididas em unidades menores e independentes chamadas **frames**. Esses **frames** podem ser intercalados e enviados pela conexão de forma independente uns dos outros. Na extremidade receptora, os **frames** são remontados para reconstruir a mensagem HTTP original.

Este mecanismo de **framing binário** é fundamental para alcançar a **multiplexação** no HTTP/2. Ele permite que o navegador envie várias requisições pela mesma conexão sem encontrar problemas de bloqueio. Como resultado, navegadores como o Chrome utilizam o mesmo ID de conexão para requisições HTTP/2, permitindo uma comunicação eficiente e ininterrupta entre o cliente e o servidor.

Em essência, o recurso de **multiplexação** do HTTP/2, habilitado pelo mecanismo de **binary framing**, aprimora a eficiência e a velocidade da troca de dados entre clientes e servidores, facilitando a transmissão simultânea de múltiplas requisições e respostas sobre uma única conexão.

TsgcWebSocketHTTPServer

Para melhorar o desempenho do protocolo HTTP/2, as requisições são despachadas por padrão em um pool de threads (por padrão, 32) cada vez que uma nova requisição HTTP/2 é recebida pelo servidor. Isso evita esperas quando uma única conexão envia muitas requisições concorrentes que exigiriam processamento sequencial (no contexto da thread de conexão) na ausência deste pool de threads.

O comportamento do pool de threads pode ser configurado com as propriedades a seguir.

- **HTTP2Options.PoolOfThreads.Enabled:** (por padrão false) habilite isto para despachar requisições HTTP/2 no pool de threads em vez da thread de conexão.
- **HTTP2Options.Threads:** (por padrão 32) o número de threads usadas para tratar as requisições HTTP/2. Defina um número de acordo com o número de processadores do seu servidor.

Para **ajustar finamente as requisições**, selecionando quais requisições devem ser processadas no pool de threads (porque consomem muito tempo) enquanto outras podem ser processadas na thread de conexão, você pode usar o evento **OnHttp2BeforeAsyncRequest**. Este evento é gerado antes de enfileirar a requisição no pool de threads. Utilize o parâmetro **Async** para definir se a requisição é processada em thread ou não.

```
void __fastcall TForm1::OnHTTP2BeforeAsyncRequest(TObject *Sender, TsgcWSConnection *Connection, const TIdHTTPRec
{
    if (ARequestInfo.Document == "/fast-request")
        Async = false;
}
```

TsgcWebSocketHTTPServer | Erro 404 sem corpo de resposta

Por padrão, a biblioteca Indy adiciona um corpo de conteúdo às respostas HTTP se não houver `ContentText` ou `ContentStream` atribuído. Se você quiser retornar um corpo de resposta vazio (para um erro 404 ou similar), pode utilizar a seguinte abordagem.

Crie um novo `TStringStream` sem conteúdo e atribua-o à propriedade `ContentStream` da Resposta HTTP. Dessa forma, a resposta HTTP será enviada sem as tags HTML padrão.

Exemplo

```
private void OnCommandGet(TIdContext *AContext, TIdHTTPRequestInfo *ARequestInfo,
    TIdHTTPResponseInfo *AResponseInfo)
{
    AResponseInfo->ContentStream = new TStringStream("");
    AResponseInfo->ContentType = "text/html";
    AResponseInfo->ResponseNo = 404;
}
```

TsgcWebSocketHTTPServer | Sessions

O HTTP é um protocolo stateless (pelo menos até o HTTP 1.1), portanto o cliente solicita um arquivo, o servidor envia uma resposta, e a conexão é fechada (você pode habilitar keep-alive para que a conexão não seja fechada imediatamente, mas isso está além do escopo deste artigo). As sessões permitem que você armazene informações sobre o cliente, que podem ser utilizadas durante um login de cliente, por exemplo. Você pode utilizar qualquer session ID único, pesquisar a lista de sessões para ver se uma já existe e, caso não exista, criar uma nova sessão. Uma sessão pode ser destruída após um período de inatividade ou manualmente após o logout do cliente.

Configuração

Existem algumas propriedades no TsgcWebSocketHTTPServer que habilitam/desabilitam sessões no componente servidor. As mais importantes são:

Propriedade	Descrição
SessionState	Testa é a primeira propriedade que deve ser habilitada para usar Sessions. Sem esta propriedade habilitada, as sessions não funcionarão
SessionTimeout	Aqui você deve definir um valor maior que zero (em milissegundos) para o tempo máximo que uma sessão estará ativa.
AutoStartSession	As sessões podem ser criadas automaticamente (AutoStartSession = true) ou manualmente (AutoStartSession = false). Se as sessões forem criadas automaticamente, o servidor utilizará o telP como identificador único para verificar se há uma sessão ativa armazenada.

```
TsgcWebSocketHTTPServer1->SessionState = true;
TsgcWebSocketHTTPServer1->SessionTimeout = 600000;
AutoStartSession = false;
```

Criar Sessão

Para criar uma nova sessão, você deve criar um novo **session ID** que seja **único**. Você pode usar qualquer valor. **Exemplo:** se o cliente está se autenticando, você pode usar usuário + senha + remoteip como o session ID. Em seguida, pesquise a lista de sessões para verificar se ela já existe. Se não existir, crie uma nova.

Quando uma nova sessão é criada, o evento **OnSessionStart** é chamado e, quando a sessão é fechada, o evento **OnSessionEnd** é gerado.

```
void OnCommandGet(TIdContext *AContext, TIdHTTPRequestInfo *ARequestInfo,
    TIdHTTPResponseInfo *AResponseInfo)
{
    if (ARequestInfo->Document == "/")
    {
        AResponseInfo->ServeFile(AContext, "yourpathhere\\index.html");
    }
    else
    {
        // check if user is valid
        if (((ARequestInfo->AuthUsername == "user") && (ARequestInfo->AuthPassword == "pass")) == false)
        {
            AResponseInfo->AuthRealm = "Authenticate";
        }
        else
        {
            // create a new session id with authentication data
        }
    }
}
```

```

string vID = ARequestInfo->AuthUsername + "_" + ARequestInfo->AuthPassword + "_" + ARequestInfo->RemoteIP;

// search session
TIdHTTPSession oSession = TsgcWebSocetHTTPServer1->SessionList->GetSession(vID, ARequestInfo->RemoteIP);

// create new session if not exists
if (oSession == NULL)
{
    oSession = TsgcWebSocetHTTPServer1->SessionList->CreateSession(ARequestInfo->RemoteIP, vID);
}

AResponseInfo->ContentText = "<html><head></head><body>Authenticated</body></html>";
AResponseInfo->ResponseNo = 200;
}
}
}

```

TsgcWebSocketHTTPServer | Stream Video

Se você deseja transmitir um arquivo de vídeo utilizando o servidor, pode usar a função `IndyStreamFileVideo` para transmitir o arquivo utilizando chunked transfer encoding.

A função recebe os seguintes parâmetros:

- **AContext:** é obtido do evento `OnCommandGet`.
- **AResponseInfo:** é obtido a partir do evento `OnCommandGet`.
- **aFileName:** é o nome completo do arquivo do vídeo a transmitir.
- **ContentType:** por padrão é "video/mpeg".
- **aBufferSize:** por padrão é 1024 bytes.

Exemplo: faça streaming de um vídeo quando um usuário acessa o documento `/video.mp4` e este vídeo está na pasta `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 for Indy Servers

Os componentes de servidor baseados em Indy (**TsgcWebSocketServer**, **TsgcWebsocketHTTPServer**) podem usar **Windows SChannel** (Secure Channel) como provedor TLS em vez do OpenSSL. O SChannel é a implementação TLS nativa do Windows, portanto não requer nenhuma DLL externa.

Como funciona

Quando um componente servidor tem SSL habilitado e o IOHandler está definido como **iohSChannel**, o servidor cria uma instância **TsgcIdServerIOHandlerSSLSChannel** que trata de todas as operações TLS utilizando a API SChannel do Windows. Para cada conexão de cliente recebida, o servidor realiza o handshake TLS através do provedor SChannel, negociando a versão do protocolo, a suíte de cifras e vinculando o certificado configurado. O SChannel lê os certificados do **Windows Certificate Store** ou de um **arquivo PFX** (.pfx / .p12). Nenhum arquivo PEM é necessário e nenhuma biblioteca OpenSSL precisa ser implantada.

Configuração

Para habilitar o SChannel em um servidor Indy, configure as seguintes propriedades:

1. Defina a propriedade **SSL** como **True**.
2. Defina **SSLOptions.IOHandler** como **iohSChannel**.
3. Defina **SSLOptions.Version** com a versão TLS desejada (tls1_2, tls1_3, ...).
4. Defina **SSLOptions.Port** com a porta usada para conexões seguras.
5. Configure o certificado usando um dos dois métodos descritos abaixo.

Propriedades de SChannel_Options

A subpropriedade **SSLOptions.SChannel_Options** expõe as configurações específicas do SChannel:

Property	Descrição
CertHash	O thumbprint (hash hexadecimal) de um certificado instalado no Windows Certificate Store.
CertStoreName	Qual repositório de certificados pesquisar: scsnMY (Personal), scsnRoot , scsnTrust , scsnCA .
CertStorePath	Local do armazenamento: scspStoreLocalMachine ou scspStoreCurrentUser .
CipherList	Lista opcional separada por dois pontos de algoritmos de cifra (por exemplo, CALG_AES_256:CALG_AES_128). Vazio significa usar os padrões do sistema.
UseLegacyCredentials	Quando True, utiliza a estrutura legada SCHANNEL_CRED em vez de SCH_CREDENTIALS. Habilite isto para versões mais antigas do Windows que não suportam a API mais recente.

Além disso, as propriedades gerais de **SSLOptions CertFile** e **Password** são usadas ao carregar um certificado de um arquivo PFX.

Certificado do Windows Store

Se o certificado já estiver instalado no Windows Certificate Store, forneça o **thumbprint** do certificado e indique onde ele está localizado.

Para encontrar o thumbprint, abra o **PowerShell** e execute:

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


A coluna **Thumbprint** mostra o hash hexadecimal de que você precisa.

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

```
oServer = new TsgcWebSocketHTTPServer();
oServer->SSL = true;
oServer->SSLOptions->IOHandler = iohSChannel;
oServer->SSLOptions->Version = tls1_2;
oServer->SSLOptions->Port = 443;
oServer->Port = 443;
oServer->SSLOptions->SChannel_Options->CertHash = "C12A8FC8AE668F866B48F23E753C93D357E9BE10";
oServer->SSLOptions->SChannel_Options->CertStoreName = scsnMY;
oServer->SSLOptions->SChannel_Options->CertStorePath = scspStoreLocalMachine;
oServer->Active = true;
```

Certificado a partir de arquivo PFX

Se você tem um arquivo de certificado PFX (.pfx ou .p12), defina as propriedades **CertFile** e **Password** em **SSLOptions**. O **SChannel** importará o certificado na inicialização.

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

Se você possui um certificado PEM e uma chave privada, converta-os primeiro para o formato PFX utilizando **OpenSSL**:

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

Versão TLS

Utilize a propriedade **SSLOptions.Version** para controlar qual versão do TLS o servidor aceita:

Value	Descrição
tls1_0	TLS 1.0 (não recomendado)
tls1_1	TLS 1.1
tls1_2	TLS 1.2 (recomendado)
tls1_3	TLS 1.3
tlsUndefined	Aceitar TLS 1.0, 1.1 e 1.2

Cipher List

Por padrão, o **SChannel** utiliza a configuração de cifras do sistema. Você pode restringir as cifras permitidas definindo **SChannel_Options.CipherList** como uma lista de nomes de algoritmos separados por dois-pontos, por exemplo:

```
CALG_AES_256:CALG_AES_128
```

Deixe esta propriedade vazia para utilizar os padrões do Windows.

Legacy Credentials

O Windows Server 2019 e anteriores podem não suportar a nova API **SCH_CREDENTIALS**. Se o servidor falhar ao iniciar em uma versão mais antiga do Windows, defina **SChannel_Options.UseLegacyCredentials** como **True** para utilizar a estrutura legada **SCHANNEL_CRED** em vez dela.

O componente detecta a versão do Windows automaticamente na maioria dos casos, mas você pode forçar o modo legacy se necessário.

Vantagens em relação ao OpenSSL

- **Sem DLLs externas:** o SChannel está integrado ao Windows, portanto você não precisa implantar libeay32.dll / ssleay32.dll ou libcrypto / libssl.
- **Integração com o Windows Certificate Store:** utilize certificados já instalados e gerenciados pelo sistema operacional.
- **Atualizações automáticas:** melhorias de TLS e patches de segurança são aplicados por meio do Windows Update.

Notas

- O SChannel está disponível **somente no Windows**. Para servidores multiplataforma, utilize o OpenSSL (iohOpenSSL).
- O servidor deve ter a **chave privada** associada ao certificado. Ao usar o método Windows Store, o certificado deve ter sido importado com sua chave privada.
- Para servidores de produção que utilizam o método Certificate Store, o armazenamento **Local Machine** (scspStoreLocalMachine) é recomendado para que o certificado esteja disponível independentemente de qual conta de usuário executa o serviço.
- Somente arquivos de certificado **PFX** (.pfx / .p12) são suportados. Se você tem arquivos PEM, converta-os para o formato PFX primeiro.

TsgcWebSocketServer_HTTPAPI

A HTTP Server API permite que aplicações se comuniquem por HTTP sem utilizar o Microsoft Internet Information Server (IIS).

Introdução

A HTTP Server API permite que as aplicações se comuniquem por HTTP sem utilizar o Microsoft Internet Information Server (IIS). As aplicações podem se registrar para receber requisições HTTP de URLs específicas, receber requisições WebSocket e enviar respostas WebSocket. A HTTP Server API inclui suporte a SSL para que as aplicações possam trocar dados por conexões HTTP seguras sem IIS. Ela também foi projetada para trabalhar com I/O completion ports.

O servidor suporta os seguintes protocolos:

- WebSockets (Requer Windows 8 ou posterior)
- HTTP 1.1
- HTTP/2 (Requer Windows 2016+ ou Windows 10+).

Por padrão, este componente requer que sua aplicação seja executada em modo Administrador, para o registro de URL. Se a URL já tiver sido registrada usando uma ferramenta externa como o netsh, você pode executar sem direitos de administrador; desabilite a propriedade BindingOptions.ConfigureSSLCertificate para permitir iniciar a aplicação sem direitos de administrador. Defina o FastMM4/FastMM5 como a primeira unit do seu projeto.

Siga os passos abaixo para configurar este componente:

1. Coloque um componente TsgcWebSocketServer_HTTPAPI no formulário
2. Defina o endereço e a porta de escuta:

```
Server->Host = "127.0.0.1";  
Server->Port = 80;
```

3. Defina as Specifications permitidas; por padrão, todas as especificações são permitidas.

RFC6455: é a especificação WebSocket padrão e recomendada.

Hixie76: é um rascunho e só é recomendado estabelecer conexões Hixie76 se você quiser fornecer suporte a navegadores antigos como o Safari 4.2

5. Crie um procedimento e defina a propriedade Active = true

URL Reservation

O servidor HTTP.SYS utiliza a reserva de URL para atribuir quais endpoints de URL serão utilizados pelo servidor HTTP.SYS.

Reserva Básica de URL

Este é o modo mais simples de configurar o Servidor; basicamente você só define o Host e a Port que o servidor HTTP.SYS tratará.

Exemplo: se o seu servidor é executado no IP 127.0.0.1 e na Porta 80, basta definir as seguintes propriedades

```
Server->Host = "127.0.0.1";
Server->Port = 80;
```

Se o servidor for executado em mais de um IP e você quiser fazer bind em múltiplos IPs, utilize o método **NewBinding**. Primeiro limpe as propriedades Host e Bindings e então utilize o método NewBinding para definir todos os Bindings do Servidor.

```
Server->Host = "";
Server->Bindings->Clear;
Server->Bindings->NewBinding("127.0.0.1", 80, "");
Server->Bindings->NewBinding("80.50.55.11", 80, "");
```

Se o servidor exigir conexões SSL, faça o seguinte para definir o Host e a Port que serão usados para tratar as conexões SSL.

```
Server->Host = "127.0.0.1";
Server->Port = 443;
Server->SSL = true;
Server->SSLOptions->Hash = "CERTIFICATE_HASH";
```

Se o servidor exigir conexões SSL com múltiplos endereços IP, primeiro limpe as propriedades Host e Bindings e então registre os novos 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");
```

Referência

- Propriedades — 27 propriedades disponíveis no componente.
- Methods — 9 métodos públicos.
- Eventos — 23 eventos que você pode manipular.

Guides

- **Configuração**
 - [Reserva de URL](#)
- **Conexão**
 - [OnDisconnect não disparado](#)
- **SSL**
 - [HTTPAPI Server SSL](#)
 - [Certificados Autoassinados](#)
- **HTTP**
 - [Custom Headers](#)
 - [Send Text Response](#)
 - [Send File Response](#)
 - [Post Big Files](#)
- **HTTP/2**
 - [Disable HTTP/2](#)

HTTPAPI | Reserva de URL

A reserva de URL HTTP.SYS é um recurso do sistema operacional Windows que permite que um usuário reserve um Uniform Resource Locator (URL) específico para sua aplicação ou serviço. Quando uma URL é reservada usando HTTP.SYS, o sistema operacional intercepta quaisquer requisições HTTP de entrada para essa URL e as encaminha para a aplicação ou serviço especificado.

Para reservar uma URL utilizando o HTTP.SYS, uma aplicação ou serviço deve primeiro registrar a URL com o driver HTTP.SYS fazendo uma chamada à HTTP API. A aplicação ou serviço especifica a URL, o método HTTP (por exemplo, GET, POST) e quaisquer configurações adicionais, como requisitos de autenticação.

Uma vez que a URL está registrada, o HTTP.SYS interceptará quaisquer requisições HTTP recebidas para aquela URL e procurará a aplicação ou serviço registrado com base na URL e no método. Se uma aplicação ou serviço correspondente for encontrado, o driver HTTP.SYS passará a requisição para aquela aplicação ou serviço para processamento.

Comandos NETSH

Registrar uma URL

Neste exemplo, a URL `http://example.com:80/` está sendo registrada para o usuário `DOMAIN\user`. Você pode substituí-la pela URL e pelo usuário desejados.

```
netsh http add urlacl url=http://example.com:80/ user=DOMAIN\user
```

Excluir uma URL

Neste exemplo, a URL `http://example.com:80/` está sendo excluída. Você pode substituí-la pela URL que deseja excluir.

```
netsh http delete urlacl url=http://example.com:80/
```

Show All URLs

Este comando exibirá uma lista de todas as reservas de URL registradas no sistema.

```
netsh http show urlacl
```

TsgcWebSocketServer_HTTPAPI

O servidor HTTP.SYS registra as URLs automaticamente quando é iniciado. Isso é feito usando os seguintes parâmetros e métodos.

- **Host e Port:** se Host não estiver vazio e o Port for diferente de zero, o servidor tentará registrar a URL. Exemplo: a URL `https://127.0.0.1:5000` será registrada usando as seguintes propriedades
 - Host = '127.0.0.1';
 - Port = 5000
 - SSL = True
- **NewBinding:** use este método para registrar uma ou várias URLs.
 - Registre a url `https://127.0.0.1:5000` --> `NewBinding('127.0.0.1', 5000, '/', True)`
 - Registre a url `https://+:5000/ws/` --> `NewBinding('+', 5000, '/ws/')`

O registro da URL requer privilégios de administrador nos seguintes casos:

- Número da Porta abaixo de 1024
- O host é um curinga "+", em vez de um endereço ip.

Se você quiser registrar a porta 443 para todos os Endereços IP do servidor e escutar apenas no endpoint "/ws/", mas não quiser executar o servidor com direitos de administrador, faça os seguintes passos:

- Registre a URL utilizando netsh
 - `netsh http add urlacl url=https://+:443/ws/ user=DOMAIN\user`
- Configure o servidor com o seguinte binding
 - `NewBinding('+', 443, '/ws/', True);`
- Desabilite a propriedade `ConfigureSSLCertificate`
 - `TsgcWebSocketServer_HTTPAPI.BindingOptions.ConfigureSSLCertificate = false;`
- Configure o Certificado SSL
 - [HTTPAPI Server SSL](#)

HTTPAPI | FineTune

TsgcWSServer_HTTPAPI expõe uma propriedade `published` chamada `FineTune` do tipo `TsgcServerHTTPAPI_FineTune`. Ela agrupa os ajustes de baixo nível em modo kernel que influenciam como a Windows HTTP Server API (`http.sys`) enfileira, despacha e completa as requisições.

QueueLength

Encapsula a configuração de kernel `HttpServerQueueLengthProperty`. Controla o número máximo de requisições pendentes que o `http.sys` manterá em sua queue de modo kernel antes que o servidor as tenha retirado da fila. Quando a queue está cheia, o kernel responde a novas conexões diretamente com **503 Service Unavailable**, sem nunca chegar ao modo usuário.

Type: ULONG.

Padrão: 1000 (o padrão do kernel do Windows).

Range: até 65535 requisições.

O que melhora: aumentar este valor impede que o kernel rejeite tráfego legítimo em cargas de trabalho com picos (bursty) - frotas IoT reconectando após uma falha de rede, clientes de broadcast de dados de mercado reconectando após um reset do provedor ou workers de jobs agendados que conectam todos em um tick de cron compartilhado.

SkipIOCPOnSuccess

Quando definido como `True`, o servidor chama `SetFileCompletionNotificationModes` no handle da fila de requisições com as flags `FILE_SKIP_COMPLETION_PORT_ON_SUCCESS` e `FILE_SKIP_SET_EVENT_ON_HANDLE`. Operações de I/O sobrepostas na fila de requisições que completam de forma síncrona não postam mais um pacote de conclusão extra à I/O completion port.

Type: Boolean.

Default: False (nenhuma flag é definida, preservando o comportamento atual).

O que isso melhora: elimina um salto de modo kernel para modo usuário no caminho crítico de requisições quando a chamada retorna `NO_ERROR` de forma síncrona, o worker despacha a conclusão inline na thread chamadora em vez de aguardar um pacote IOCP. Este é o padrão usado pelo exemplo de referência HTTP Server High Performance da Microsoft. O flag deve ser combinado com `OperatingMode = ompHighPerf`.

A API é resolvida em tempo de execução via `GetProcAddress` a partir de `kernel32`. No Windows XP e anteriores (target de runtime do Delphi 7), o ponto de entrada não existe - o wrapper retorna `False` e o servidor é executado sem a flag definida.

OperatingMode

Seleciona uma de duas arquiteturas de accept/dispatch.

Type: `TsgcHTTPAPIOperatingMode = (ompClassic, ompHighPerf)`.

Padrão: `ompClassic`.

- **ompClassic** - uma única thread de acceptor chama `HttpReceiveHttpRequest` de forma síncrona e despacha manualmente cada requisição para o pool de workers IOCP por meio de `PostQueuedCompletionStatus`. Este é o comportamento histórico.
- **ompHighPerf** - implementa o padrão de referência HTTP Server High Performance da Microsoft. A thread acceptor é removida. Na inicialização, o servidor pré-posta `ThreadPoolSize x HighPerfAcceptsPerWorker` recebimentos assíncronos no IOCP vinculado à fila (padrão $32 \times 4 = 128$ recebimentos pendentes concorrentes). Cada worker atende sua conclusão inline e re-posta outro recebimento assíncrono, mantendo uma janela deslizante até o desligamento.

O que melhora: O `ompHighPerf` compensa quando o servidor enfrenta pipelines profundos de stream único (uploads/downloads de frames grandes) ou muitos clientes simultâneos (centenas a milhares). A janela de recebimento pré-postada absorve bursts sem alocação por conexão, e o dispatch inline remove o gargalo de hand-off do acceptor. Mantenha o `ompClassic` padrão para APIs de baixo tráfego e ambientes de desenvolvimento — em cargas leves, o overhead de manter 128 contextos pré-postados custa mais do que economiza. O modo só pode ser alterado antes de o servidor ser ativado.

HighPerfAcceptsPerWorker

Controla quantos recebimentos assíncronos cada worker IOCP pré-posta quando `operatingMode = ompHighPerf`. O valor é ignorado no modo `ompClassic`. O número total de recebimentos pendentes simultâneos que o servidor mantém é igual a `ThreadPoolSize x HighPerfAcceptsPerWorker`.

Type: Integer.

Default: 4.

O que isso melhora: uma janela mais profunda por worker permite que o servidor absorva rajadas maiores de requisições de entrada sem alocar novos contextos no caminho crítico. Aumente esse valor para implantações de alta concorrência (frotas de IoT, distribuição de dados de mercado, brokers de fan-out); o custo é a memória, cada recebimento pré-postado mantém um buffer de requisição (~16 KB) reservado até ser concluído. O valor padrão 4 é um meio-termo conservador validado com base no exemplo HP do MSDN.

Exemplo

O trecho a seguir configura um servidor HTTP.sys para um backend IoT de alta concorrência: uma grande fila de kernel para absorver tempestades de reconexão, dispatch HighPerf com uma janela de recebimento pré-postada ampliada, e dispatch de conclusão inline habilitado.

```
#include "sgcWebSocket_Server_HTTPAPI.hpp"
#include "sgcWebSocket_HTTPAPI_Server.hpp" // TsgcHTTPAPIOperatingMode
TsgcWSServer_HTTPAPI *oServer = new TsgcWSServer_HTTPAPI(NULL);
oServer->Host = "0.0.0.0";
oServer->Port = 8080;
// absorb 10,000-device reconnect bursts before kernel-level 503
oServer->FineTune->QueueLength = 10000;
// switch from single-acceptor to pre-posted IOCP workers
oServer->FineTune->OperatingMode = ompHighPerf;
// widen the per-worker pre-posted receive window (32 threads * 8 = 256)
oServer->FineTune->HighPerfAcceptsPerWorker = 8;
// dispatch inline on sync-success completions; skip the IOCP round-trip
oServer->FineTune->SkipIOCPOnSuccess = true;
oServer->Active = true;
```

Para uma API interna típica ou de baixo tráfego, deixe cada propriedade `FineTune` em seu padrão:

```
TsgcWSServer_HTTPAPI *oServer = new TsgcWSServer_HTTPAPI(NULL);
oServer->Host = "localhost";
oServer->Port = 8080;
// FineTune defaults: QueueLength=1000, SkipIOCPOnSuccess=False,
//                    OperatingMode=ompClassic, HighPerfAcceptsPerWorker=4
oServer->Active = true;
```


TsgcWebSocketServer_HTTPAPI | HTTPAPI Server SSL

O servidor pode ser configurado para usar **certificados SSL**. Para obter um servidor de produção com um certificado de servidor, você deve **adquirir** um certificado de um **provedor conhecido**: Namecheap, GoDaddy, Thawte, etc. Para **fins de teste**, você pode usar um **certificado autoassinado** (confira o exemplo Demos/Chat, que usa um certificado autoassinado). Leia o seguinte artigo [Como criar um certificado autoassinado](#).

Depois de ter seu certificado, você deve configurar o servidor para especificar qual certificado usar para criptografar as conexões.

Hash do certificado

Primeiro você precisa conhecer o hash do seu certificado. Encontrar o hash de um certificado é tão fácil no **powershell** quanto executar um comando **dir** no contêiner de certificados.

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

O hash é o valor hexadecimal do **Thumbprint**.

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

Assim que você tiver o valor do Thumbprint, basta defini-lo na propriedade **TsgcWebSocketServer_HTTPAPI.TLSOptions.Hash**.

Uma vez que você tenha definido o hash, basta definir **TsgcWebSocketServer_HTTPAPI.SSL = true** e seu servidor está pronto para iniciar.

Se você deseja registrar o certificado manualmente utilizando o netsh, use o seguinte comando:

```
netsh http add sslcert ipport=<IP>:<PORT> certhash=<THUMBPRINT> appid="{<GUID>}"
```

<IP>: Especifica o endereço IP local para o binding. Não use um binding curinga. Use um endereço IP válido.

<PORT>: Especifica a porta para o binding.

<THUMBPRINT>: O thumbprint do certificado X.509.

<GUID>: Um GUID gerado pelo desenvolvedor para representar o app para fins informativos.

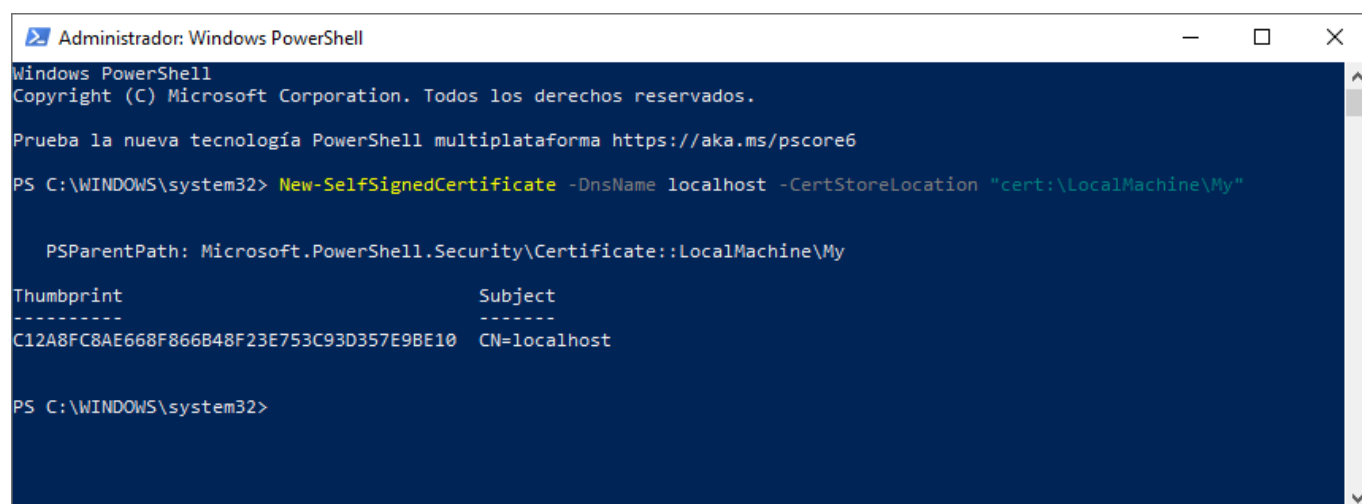
TsgcWebSocketServer_HTTPAPI | Self-Signed Certificates

Se você precisar de um certificado para seus próprios testes, pode criar um certificado autoassinado na sua máquina de teste. Siga estas etapas:

1. Execute o **Powershell** como **Administrador**.
2. Execute o seguinte comando para criar o certificado:

```
New-SelfSignedCertificate -DnsName localhost -CertStoreLocation "cert:\LocalMachine\My"
```

Se for bem-sucedido, você receberá uma confirmação sobre o novo certificado criado. Basta copiar o Thumbprint e colá-lo na propriedade **TsgcWebSocketServer_HTTPAPI.TLSOptions.Hash**.



```
Administrador: 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. **Opcional**, você pode adicionar seu certificado autoassinado como uma **autoridade certificadora confiável**

```
Run MMC -32 as administrator
```

- 3.1. Selecione **File / Add or Remove Snap-in**

- 3.2. Selecione Certificate e então clique em Add

- 3.2. Selecione conta de computador e pressione Avançar.

- 3.4. Selecione Local computer e pressione Ok. Você verá agora seus Certificados.

- 4.5. Selecione o seu certificado em **Pessoal / Certificados** e Cole em **Autoridades de Certificação Raiz Confiáveis / Certificados**.

TsgcWebSocketServer_HTTPAPI | Desabilitar HTTP/2

O protocolo HTTP/2 é habilitado por padrão no **Server 2016+** e no SO **Windows 10+**. Em alguns navegadores ou clientes HTTP mais antigos, você pode encontrar um erro porque o protocolo não é totalmente suportado. Você pode evitar esses erros desabilitando o protocolo HTTP/2.

Como Desabilitar o HTTP/2

- Abra o Editor do Registro do Windows
- Vá para a seguinte chave de registro:

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\HTTP\Parameters

- Adicione os seguintes valores DWORD e defina ambos os valores como zero.
 - EnableHttp2Tls
 - EnableHttp2Cleartext
- Reinicia o computador.

TsgcWebSocketServer_HTTPAPI | Custom Headers

Você pode personalizar a resposta do servidor HTTP.SYS usando a propriedade **CustomHeaders** do objeto de resposta.

Defina o valor de CustomHeaders com o Nome do Cabeçalho e o Valor do Cabeçalho separados por caracteres de nova linha.

Exemplo: se você quiser adicionar os seguintes cabeçalhos, veja abaixo um código de exemplo

```
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: *\r\n" + "Access-Control-Allow-Methods: " +  
        "GET, POST, OPTIONS, PUT, PATCH, DELETE";  
}
```

TsgcWebSocketServer_HTTPAPI | Send Text Response

Utilize o evento **OnHTTPRequest** para tratar as Requisições HTTP.

A classe **THttpRequest** contém os Dados da Requisição HTTP.

- **Document:** o Document que o peer está tentando acessar.
- **Method:** o método HTTP ('GET', 'POST', etc.)
- **Headers:** os cabeçalhos da requisição HTTP.
- **AcceptEncoding:** variável de accept encoding, exemplo: "gzip, deflate, br".
- **ContentType:** exemplo: "text/html"
- **Content:** conteúdo da requisição, se existir.
- **QueryParams:** os parâmetros de query.
- **Cookies:** os cookies, se existirem.
- **ContentLength:** tamanho do conteúdo.
- **AuthExists, AuthUsername, AuthPassword:** dados da requisição de autenticação.
- **Stream:** se a requisição http tem um corpo, este é o stream do corpo.

A classe **THttpResponse** contém os Dados da resposta HTTP.

- **ContentText:** é a resposta como texto.
- **ContentType:** exemplo: "text/html". Se você quiser codificar o ContentText com UTF8, defina o charset='utf-8'. Exemplo: text/html; charset=utf-8
- **CustomHeaders:** se você precisar enviar seus próprios cabeçalhos, utilize esta variável
- **AuthRealm:** se o servidor exigir autenticação, defina esta variável.
- **ResponseNo:** o número da resposta HTTP. Exemplo: 200 significa que a resposta foi bem-sucedida.
- **ContentStream:** se a resposta contém um stream, defina aqui (não libere o stream, ele será liberado automaticamente).
- **FileName:** se a resposta for um nome de arquivo, defina aqui o caminho completo do arquivo.
- **Date, Expires, LastModified:** variáveis de data e hora da resposta.
- **CacheControl:** permite personalizar o comportamento de cache.

Exemplo: se o servidor receber uma requisição GET pelo documento "/test.html", envia uma resposta OK; caso contrário, envia um 404 se for uma requisição GET por outro documento, ou erro 500 se for um método diferente.

```
void __fastcall OnHTTPRequest(TsgcWSConnection_HTTPAPI *aConnection,
    const THttpRequest *aRequestInfo,
    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 | Send File Response

Use a propriedade `FileName` do objeto **THttpRequestResponse** se você quiser enviar um arquivo como resposta a uma requisição HTTP.

```
void __fastcall OnHTTPRequest(TsgcWSConnection_HTTPAPI *aConnection,
    const THttpRequest *aRequestInfo,
    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;
    }
}
```

Downloads retomáveis

Uma resposta HTTP 206 Partial Content é utilizada quando um servidor está atendendo a uma requisição de uma porção específica (intervalo) de um recurso, em vez de enviar o arquivo inteiro. Isso é comumente utilizado para downloads que podem ser retomados, streaming de mídia e transferências de arquivos grandes.

Como funciona:

O Cliente Solicita um Recurso Parcial: O cliente (navegador, downloader ou media player) envia um cabeçalho `Range` especificando o intervalo de bytes que deseja. Exemplo de requisição:

```
GET /video.mp4 HTTP/1.1
Host: example.com
Range: bytes=1000-5000
```

Isto requisita os bytes 1000 a 5000 de video.mp4.

Servidor Responde com HTTP 206: Se o servidor suportar requisições de range, ele responde com 206 Partial Content e inclui um cabeçalho `Content-Range`. Exemplo de resposta:

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 1000-5000/1000000
Content-Length: 4001
Content-Type: video/mp4
```

O cabeçalho `Content-Range` mostra:

O intervalo servido (1000-5000)

O tamanho total do arquivo (1000000 bytes).

O cabeçalho `Content-Length` é o tamanho da porção retornada (4001 bytes).

O Cliente Pode Solicitar Mais Chunks:

O cliente pode enviar múltiplas requisições para partes diferentes.

Isto habilita downloads retomáveis e streaming eficiente.

```

void __fastcall OnHttpRequest(TIdContext* AContext, TIdHTTPRequestInfo* ARequestInfo, TIdHTTPResponseInfo* AResponseInfo)
{
    std::unique_ptr<TFileStream> oStream(new TFileStream("test.pdf", fmOpenRead | fmShareDenyWrite));
    std::unique_ptr<TIdEntityRangeStrings> oRanges(new TIdEntityRangeStrings(nullptr));
    try {
        oRanges->Text = ARequestInfo->RawHeaders->Values["Range"];
        AResponseInfo->ContentType = "application/pdf";
        if (oRanges->Count > 0) {
            AResponseInfo->ResponseNo = 206; // Partial Content
            AResponseInfo->AcceptRanges = "bytes";
            AResponseInfo->ContentRangeStart = oRanges->Items[0]->StartPos;
            AResponseInfo->ContentRangeEnd = oRanges->Items[0]->EndPos;
            AResponseInfo->ContentRangeInstanceLength = oStream->Size;

            AResponseInfo->ContentStream = new TIdHTTPRangeStream(oStream.release(),
                AResponseInfo->ContentRangeStart, AResponseInfo->ContentRangeEnd);
        } else {
            AResponseInfo->ResponseNo = 200; // OK
            AResponseInfo->ContentStream = oStream.release();
        }
    } catch (...) {
        // Handle exceptions
    }
}

```

TsgcWebSocketServer_HTTPAPI | OnDisconnect not fired

Ao trabalhar pela primeira vez com o Servidor HTTPAPI, é muito comum observar que o evento OnDisconnect não é disparado imediatamente quando um cliente fecha a conexão. O motivo é que o Servidor HTTPAPI funciona de forma um pouco diferente de outros servidores como o Indy. No **servidor Indy** existe **uma thread para cada conexão** e essa thread verifica a cada x milissegundos se a **conexão está ativa**. O **Servidor HTTPAPI** utiliza um **thread-pool** que trata todas as conexões e **não verifica** para cada conexão se ela está ativa ou não.

Para ser notificado quando o cliente fechar a conexão, faça a seguinte configuração:

1. Se você utiliza um [TsgcWebSocketClient](#), defina **Options.CleanDisconnect := True**. Isso significa que, antes de a conexão ser fechada, o cliente tentará enviar uma notificação ao servidor de que a conexão será fechada. Se o servidor receber esta mensagem, o evento OnDisconnect será chamado.
2. Para outras desconexões, a única solução é gravar algo no socket; se falhar, a conexão está desconectada. **Habilite o HeartBeat** no servidor HTTPAPI, e defina um intervalo de 60 segundos, por exemplo, e um timeout de 0. Esta configuração significa que a cada 60 segundos todas as conexões receberão ping, e se alguma estiver desconectada, o evento **OnDisconnect** será disparado. Você pode definir um valor menor de HeartBeat.Interval, mas não o defina muito baixo (1 segundo, por exemplo, é muito baixo) porque o desempenho do servidor será afetado.

TsgcWebSocketClient_WinHTTP

O TsgcWebSocketClient_WinHTTP implementa um componente WebSocket VCL Cliente que pode conectar a um servidor WebSocket.

Introdução

O TsgcWebSocketClient_WinHTTP implementa um Componente Cliente WebSocket VCL que pode conectar a um WebSocket Server. É baseado na WinHTTP API e requer Windows 8 ou superior. Siga os passos abaixo para configurar este componente:

1. Solte um componente TsgcWebSocketClient_WinHTTP no formulário
2. Defina Host e Port (o padrão é 80) para se conectar a um Servidor WebSocket disponível. Você pode definir a propriedade URL e Host, Port, Parameters... serão atualizados a partir da URL. Exemplo: wss://127.0.0.1:8080/ws/ resultará em:

```
oClient = new TsgcWebSocketClient_WinHTTP();
oClient->Host = "127.0.0.1";
oClient->Port = 80;
oClient->TLS = true;
oClient->Options->Parameters = "/ws/";
```

3. Você pode selecionar se deseja TLS (conexão segura) ou não, desabilitado por padrão.
4. Os seguintes eventos podem ser usados para personalizar o fluxo do cliente websocket:
5. Crie um procedimento e defina a propriedade Active como True.

Referência

- Propriedades — 16 propriedades disponíveis no componente.
- Métodos — 8 métodos públicos.
- Eventos — 9 eventos que você pode tratar.

TsgcWebSocketFirewall

O TsgcWebSocketFirewall implementa um componente de firewall abrangente que protege servidores WebSocket contra uma ampla gama de ameaças de segurança.

Introdução

O TsgcWebSocketFirewall implementa um componente de firewall abrangente que protege servidores WebSocket contra uma ampla gama de ameaças de segurança. Ele fornece dezesseis módulos de proteção, incluindo filtragem por blacklist/whitelist de IP, proteção contra força bruta com banimento automático de IP, detecção de SQL injection, detecção de XSS (Cross-Site Scripting), rate limiting de conexões, proteção contra flood de mensagens, filtragem por país baseada em GeoIP, detecção de bots baseada em IP, threat scoring dinâmico, detecção de path traversal, detecção de command injection, limitação de tamanho de payload, proteções específicas de WebSocket (validação de origin, limites de tamanho de frame, filtragem de subprotocolo), escalonamento progressivo de ban, um motor de regras personalizadas e estatísticas em tempo real.

O firewall integra-se automaticamente aos componentes de servidor através da propriedade **Firewall** disponível em TsgcWebSocketServer, TsgcWebSocketHTTPServer e TsgcWebSocketServer_HTTPAPI. Uma vez atribuído e configurado, ele intercepta conexões e mensagens sem exigir nenhuma conexão manual de eventos.

Início rápido

1. Coloque um componente **TsgcWebSocketFirewall** no formulário.
2. Configure os módulos de proteção desejados:

Blacklist: Defina Enabled como True e adicione IPs ou faixas CIDR à lista IPs (por exemplo, "10.0.0.0/8", "192.168.1.100").

Whitelist: Defina Enabled como True e adicione IPs confiáveis. IPs na lista de permissões contornam todas as outras verificações.

BruteForce: Defina Enabled como True. Configure MaxAttempts (padrão 5), TimeWindowSec (padrão 60) e BanDurationSec (padrão 300).

SQLInjection: Defina Enabled como True. Detecta padrões comuns de injeção de SQL em mensagens.

XSS: Defina Enabled como True. Detecta padrões de cross-site scripting nas mensagens.

RateLimit: Defina Enabled como True. Limita conexões simultâneas por IP (padrão 10 por 60 segundos).

FloodProtection: Defina Enabled como True. Limita as mensagens por segundo por IP (padrão 100).

PayloadLimit: Defina Enabled como True. Bloqueia mensagens que excedam MaxSizeBytes (padrão 1MB).

PathTraversal: Defina Enabled como True. Detecta padrões de directory traversal.

CommandInjection: Defina Enabled como True. Detecta padrões de injeção de comando de shell.

ThreatScore: Defina Enabled como True. Atribui pontuações dinâmicas aos IPs com base nas violações.

BanEscalation: Defina Enabled como True. Escala a duração dos banimentos para infratores reincidentes.

GeoIP: Defina Enabled como True. Bloqueia ou permite conexões por código de país.

BotDetection: Defina Enabled como True. Classifica os IPs que se conectam como crawlers verificados, hosts de datacenter, bots suspeitos, em lista de bloqueio ou humanos.

WebSocketProtection: Defina Enabled como True. Valida origens, tamanhos de frame e subprotocolos.

CustomRules: Defina Enabled como True. Motor de regras configurável para filtragem complexa.

3. Atribua o firewall a um componente de servidor:

```
sgcWebSocketHTTPServer1.Firewall := sgcWebSocketFirewall1;
```

4. Inicie o servidor. O firewall protege automaticamente todas as conexões e mensagens.

Properties

Property	Descrição
Enabled	Habilita ou desabilita o firewall. Padrão: True.
Blacklist	Configuração de blacklist de IP. Contém as propriedades Enabled (Boolean) e IPs (TStringList). Suporta IPs exatos e notação CIDR (por exemplo, "192.168.0.0/16").
Whitelist	Configuração da whitelist de IP. IPs na whitelist ignoram todas as verificações de segurança. Mesma estrutura da Blacklist.
Brute-Force	Proteção contra força bruta. Propriedades: Enabled, MaxAttempts (padrão 5), TimeWindowSec (padrão 60), BanDurationSec (padrão 300). Bane automaticamente IPs que excedam o limite de tentativas.
SQLInjection	Detecção de injeção de SQL. Propriedades: Enabled, Action (faDeny/faAllow/faLog), CustomPatterns (TStringList para padrões adicionais).
XSS	Detecção de cross-site scripting. Propriedades: Enabled, Action (faDeny/faAllow/faLog).
RateLimit	Rate limiting de conexão. Propriedades: Enabled, MaxConnectionsPerIP (padrão 10), TimeWindowSec (padrão 60).
Flood-Protection	Proteção contra flood de mensagens. Propriedades: Enabled, MaxMessagesPerSec (padrão 100), Action (faDeny/faAllow/faLog).
PayloadLimit	Limitação de tamanho de mensagem. Propriedades: Enabled, MaxSizeBytes (padrão 1048576), Action (faDeny/faAllow/faLog).
PathTraversal	Detecção de path traversal. Propriedades: Enabled, Action (faDeny/faAllow/faLog).
CommandInjection	Detecção de command injection. Propriedades: Enabled, Action (faDeny/faAllow/faLog), CustomPatterns (TStringList para padrões adicionais).
ThreatScore	Pontuação dinâmica de ameaças. Propriedades: Enabled, AutoBanThreshold (padrão 80), DecayPerHour (padrão 5), além de pesos por tipo de violação para o ajuste fino dos cálculos de pontuação.

BanEscalation	Escalonamento progressivo de banimentos. Propriedades: Enabled, Levels (TStringList de durações em segundos), ResetAfterSec (padrão 86400). Cada banimento subsequente utiliza a duração do próximo nível. Um valor de nível 0 significa banimento permanente.
GeoIP	Filtragem geográfica de IP. Propriedades: Enabled, Mode (gmBlockList/gmAllowList), Countries (TStringList de códigos de país ISO 3166-1 alpha-2), DatabaseFile (caminho para um banco de dados GeoIP CSV).
BotDetection	Deteção de bots baseada em IP. Propriedades: Enabled, KnownBotsFile, CacheDurationSec (padrão 3600), DNSTimeoutMS (padrão 2000), além dos subobjetos ReverseDNS, Datacenter e DNSBL. Apenas classificação: classifica IPs como crawlers verificados, hosts de datacenter, bots suspeitos, blocklisted ou humanos sem bloquear.
WebSocketProtection	Proteções específicas do WebSocket. Propriedades: Enabled, AllowedOrigins (TStringList), MaxFrameSize (Integer), AllowedSubprotocols (TStringList).
CustomRules	Motor de regras personalizadas. Propriedades: Enabled, Rules (TCollection de TsgcFirewallRuleItem). Cada regra define condições e ações para uma lógica de filtragem complexa.

Eventos

OnFiltered: Disparado quando uma conexão ou mensagem é bloqueada. O parâmetro *Allow* (var Boolean) permite que você sobrescreva a decisão do firewall.

OnViolation: Disparado quando uma violação de segurança é detectada. Fornece o endereço IP, o tipo de violação e detalhes para fins de logging.

OnResolveCountry: Disparado quando o GeoIP precisa resolver um IP para um código de país. Atribua um handler para fornecer resolução de país personalizada, ou carregue um arquivo de banco de dados GeoIP para consultas automáticas.

OnBotDetected: Disparado quando a detecção de bots classifica um IP. Fornece o endereço IP, o veredito TsgcBotClassification e o nome de bot correspondente para registro ou análise.

OnResolveBot: Disparado quando a detecção de bots precisa classificar um IP. Atribua um handler para fornecer um veredito personalizado em vez do resolver integrado.

OnThreatScoreChanged: Disparado quando a pontuação de ameaça de um IP muda. Fornece o endereço IP junto com os valores de pontuação antigo e novo.

Métodos Públicos

Method	Descrição
IsConnectionAllowed(IP)	Verifica se uma conexão do IP fornecido é permitida. Chamado automaticamente quando integrado a um servidor.

IsMessageAllowed(IP, Message)	Verifica se uma mensagem do IP fornecido passa por todos os filtros de mensagem. Chamado automaticamente quando integrado a um servidor.
RegisterConnection(IP)	Registra uma nova conexão para rastreamento de rate limiting. Chamado automaticamente.
UnregisterConnection(IP)	Remove uma conexão do rastreamento. Chamado automaticamente na desconexão.
RegisterFailedAttempt(IP)	Registra uma tentativa de autenticação falha para detecção de força bruta. Chame isto a partir do seu handler de evento OnAuthentication quando a autenticação falhar.
BanIP(IP, DurationSec)	Bane manualmente um endereço IP. DurationSec = 0 significa banimento permanente.
UnbanIP(IP)	Remove um ban do endereço IP especificado.
IsBanned(IP)	Retorna True se o IP especificado está atualmente banido.
ClearBans	Remove todos os bans ativos.
ClearTracking	Reinicia todos os dados de rastreamento interno (contagens de conexão, logs de tentativas, contagens de mensagens).
SaveBansToFile(FileName)	Salva todos os bans ativos em um arquivo para persistência entre reinicializações.
LoadBansFromFile(FileName)	Restaura banimentos de um arquivo previamente salvo.
LoadGeoIPDatabase(FileName)	Carrega um banco de dados GeoIP CSV. Formato esperado: start_ip,end_ip,country_code.
LookupCountry(IP)	Retorna o código de país ISO de 2 letras para um endereço IP.
GetBotClassification(IP)	Retorna a TsgcBotClassification de um endereço IP (cache-first; um cache miss pode retornar bcUnknown e acionar uma consulta assíncrona).
GetThreatScore(IP)	Retorna a pontuação de ameaça atual (0-100) para um endereço IP.
ResetThreatScore(IP)	Redefine a pontuação de ameaça de um IP para 0.
GetTopThreats(Count)	Retorna uma lista de IPs com os threat scores mais altos.
IsOriginAllowed(Origin)	Verifica se um cabeçalho WebSocket Origin é permitido pela configuração WebSocketProtection.
IsFrameSizeAllowed(Size)	Verifica se o tamanho de uma mensagem está dentro dos limites definidos por WebSocketProtection.MaxFrameSize.
IsSubprotocolAllowed(Subprotocol)	Verifica se um subprotocolo WebSocket é permitido pela configuração de WebSocketProtection.

Stats

Objeto TsgcFirewallStats público e somente leitura com contadores em tempo real para conexões bloqueadas, mensagens filtradas, banimentos emitidos, threat scores e contagens de hits por módulo.

Integração Automática

Quando atribuído à propriedade Firewall de um servidor, o componente automaticamente:

- **Em nova conexão:** Verifica blacklist, whitelist, banimentos, limites de taxa, país GeoIP e avalia regras personalizadas. Rejeita a conexão antes do evento OnTCPConnect se bloqueada.
- **On message received:** Verifica SQL injection, padrões XSS, tamanho da carga útil, path traversal, command injection, limites de flood e avalia regras personalizadas. Atualiza os threat scores. Desconecta o cliente se uma violação for detectada.
- **On disconnect:** Desregistra a conexão do rastreamento interno.

CIDR Support

A blacklist e a whitelist suportam a notação CIDR para faixas de IP:

- **192.168.1.0/24** — corresponde a 192.168.1.0 até 192.168.1.255
- **10.0.0.0/8** — corresponde a todos os endereços 10.x.x.x
- **172.16.0.0/16** — corresponde a endereços 172.16.x.x

Exemplo

```
// Configure firewall
sgcWebSocketFirewall1->Blacklist->Enabled = true;
sgcWebSocketFirewall1->Blacklist->IPs->Add("10.0.0.0/8");
sgcWebSocketFirewall1->Whitelist->Enabled = true;
sgcWebSocketFirewall1->Whitelist->IPs->Add("192.168.1.1");
sgcWebSocketFirewall1->BruteForce->Enabled = true;
sgcWebSocketFirewall1->BruteForce->MaxAttempts = 3;
sgcWebSocketFirewall1->BruteForce->BanDurationSec = 600;
sgcWebSocketFirewall1->SQLInjection->Enabled = true;
sgcWebSocketFirewall1->XSS->Enabled = true;
sgcWebSocketFirewall1->RateLimit->Enabled = true;
sgcWebSocketFirewall1->RateLimit->MaxConnectionsPerIP = 5;
sgcWebSocketFirewall1->FloodProtection->Enabled = true;

// GeoIP: block connections from specific countries
sgcWebSocketFirewall1->GeoIP->Enabled = true;
sgcWebSocketFirewall1->GeoIP->Mode = gmBlockList;
sgcWebSocketFirewall1->GeoIP->Countries->Add("CN");
sgcWebSocketFirewall1->GeoIP->Countries->Add("RU");
sgcWebSocketFirewall1->LoadGeoIPDatabase("geoip.csv");

// Threat scoring with auto-ban
sgcWebSocketFirewall1->ThreatScore->Enabled = true;
sgcWebSocketFirewall1->ThreatScore->AutoBanThreshold = 80;

// Progressive ban escalation (5min, 30min, 2hr, 24hr, permanent)
sgcWebSocketFirewall1->BanEscalation->Enabled = true;
sgcWebSocketFirewall1->BanEscalation->Levels->Add("300");
sgcWebSocketFirewall1->BanEscalation->Levels->Add("1800");
sgcWebSocketFirewall1->BanEscalation->Levels->Add("7200");
sgcWebSocketFirewall1->BanEscalation->Levels->Add("86400");
sgcWebSocketFirewall1->BanEscalation->Levels->Add("0");

// Additional content protection
sgcWebSocketFirewall1->PayloadLimit->Enabled = true;
```

```

sgcWebSocketFirewall1->PayloadLimit->MaxSizeBytes = 65536;
sgcWebSocketFirewall1->PathTraversal->Enabled = true;
sgcWebSocketFirewall1->CommandInjection->Enabled = true;

// Persistent bans
sgcWebSocketFirewall1->SaveBansToFile("bans.dat");
sgcWebSocketFirewall1->LoadBansFromFile("bans.dat");

// Assign to server
sgcWebSocketHTTPServer1->Firewall = sgcWebSocketFirewall1;

// Optional: Track failed login attempts
void __fastcall TForm1::ServerAuthentication(TsgcWSConnection *Connection,
String aUser, String aPassword, bool &Authenticated)
{
    Authenticated = (aUser == "admin") && (aPassword == "secret");
    if (!Authenticated)
        sgcWebSocketFirewall1->RegisterFailedAttempt(Connection->IP);
}

// Optional: Handle firewall events for logging
void __fastcall TForm1::FirewallViolation(TObject *Sender, const String aIP,
const TsgcFirewallViolationType aViolationType, const String aDetails)
{
    Log("Firewall violation from " + aIP + ": " + aDetails);
}

```

SQL Injection Patterns Detected

O detector de injeção de SQL integrado verifica estes padrões (sem distinção de maiúsculas/minúsculas):

- ' OR ', ' AND ' — Injeção booleana
- UNION SELECT — Injeção baseada em union
- '; DROP, '; DELETE, '; INSERT, '; UPDATE — Injeção de statement
- -- (comentários SQL)
- EXEC(, EXECUTE(— Execução de comando
- xp_cmdshell — Procedimentos armazenados estendidos
- CAST(, CONVERT(— Abuso de conversão de tipo
- 1=1, 1='1 — Injeção de tautologia

Padrões personalizados adicionais podem ser adicionados através da propriedade `SQLInjection.CustomPatterns`.

XSS Patterns Detected

- <script — Injeção de Script
- javascript: — XSS baseado em protocolo
- onerror=, onload=, onclick=, onmouseover= — Injeção de event handler
- eval(— Avaliação JavaScript
- document.cookie — Roubo de cookie
- <iframe, <object, <embed — Injeção de elemento
- <svg onload — XSS baseado em SVG
- expression(— injeção de expressão CSS

Padrões de Path Traversal Detectados

O detector de travessia de caminho integrado verifica os seguintes padrões:

- ../ e ..\ — Directory traversal relativo
- %2e%2e%2f, %2e%2e/, ..%2f — variantes URL-encoded
- %00 — Injeção de null byte
- /etc/passwd — acesso a arquivo sensível do Unix
- c:\windows — Acesso ao diretório de sistema do Windows

Padrões de Injeção de Comando Detectados

O detector de injeção de comandos integrado verifica estes padrões:

- `;` `|` `&&` `||` — Operadores de shell (encadeamento de comandos)
- ``command`` — Execução com backtick
- `$(command)` — Execução de subshell
- `rm -rf` — Operações destrutivas de arquivos
- `wget`, `curl` — Download remoto de arquivos
- `/bin/sh`, `/bin/bash` — invocação de shell Unix
- `cmd.exe` — Interpretador de comandos do Windows
- `powershell` — invocação do PowerShell

Padrões personalizados adicionais podem ser adicionados por meio da propriedade `CommandInjection.CustomPatterns`.

Segurança de Threads

O componente de firewall é totalmente thread-safe. Todos os métodos públicos utilizam critical sections internas para proteger o acesso concorrente aos dados de rastreamento. Ele foi submetido a testes de estresse com 20 threads concorrentes realizando 100.000 operações com zero erros e zero vazamentos de memória.

O componente pode ser compartilhado com segurança entre várias instâncias de servidor e acessado a partir de qualquer thread (handlers de evento do servidor, threads de timer, etc.) sem sincronização externa.

Referência

- Propriedades — 16 propriedades disponíveis no componente.
- Métodos — 21 métodos públicos.
- Eventos — 6 eventos que você pode tratar.

Firewall: Blacklist e Whitelist

O TsgcWebSocketFirewall fornece controle de acesso baseado em IP por meio de dois mecanismos complementares: uma blacklist que bloqueia IPs específicos e uma whitelist que concede acesso incondicional a IPs confiáveis.

Blacklist

A blacklist evita conexões de endereços IP ou intervalos de IP especificados. Quando habilitada, qualquer conexão recebida de um IP na blacklist é rejeitada antes de alcançar os eventos de conexão do servidor.

Property	Descrição
Blacklist.Enabled	Habilita ou desabilita a verificação da blacklist. Padrão: False.
Blacklist.IPs	TStringList contendo endereços IP ou intervalos CIDR bloqueados.

Adicionando IPs em tempo de design

Clique na propriedade IPs no Object Inspector para abrir o editor de String List. Adicione um IP ou faixa CIDR por linha:

```
192.168.1.100
10.0.0.0/8
172.16.0.0/12
```

Adicionando IPs em tempo de execução

```
sgcWebSocketFirewall1->Blacklist->Enabled = true;
sgcWebSocketFirewall1->Blacklist->IPs->Add("192.168.1.100");
sgcWebSocketFirewall1->Blacklist->IPs->Add("10.0.0.0/8");
```

Whitelist

A whitelist concede acesso incondicional aos endereços IP especificados. **IPs na lista de permissões contornam todas as outras verificações do firewall**, incluindo blacklist, banimentos por força bruta, limites de taxa e filtragem de mensagens.

Property	Descrição
Whitelist.Enabled	Habilita ou desabilita a verificação de whitelist. Padrão: False.
Whitelist.IPs	TStringList contendo endereços IP ou faixas CIDR confiáveis.

Exemplo

```
// Allow internal network unconditionally
sgcWebSocketFirewall1->Whitelist->Enabled = true;
sgcWebSocketFirewall1->Whitelist->IPs->Add("192.168.1.0/24");
sgcWebSocketFirewall1->Whitelist->IPs->Add("127.0.0.1");
```

Ordem de Prioridade

Quando tanto a blacklist quanto a whitelist estão habilitadas, o firewall as avalia nesta ordem:

1. Se o IP estiver na whitelist, a conexão é **permitida imediatamente**. Nenhuma verificação adicional é realizada.
2. Se o IP estiver na blacklist, a conexão é **negada**.
3. Se o IP não estiver em nenhuma das listas, a conexão prossegue para outras verificações (força bruta, limite de taxa etc.).

Isso significa que uma entrada na whitelist sempre tem prioridade sobre uma entrada na blacklist para o mesmo IP.

Notação CIDR

Tanto a blacklist quanto a whitelist suportam a notação CIDR (Classless Inter-Domain Routing) para especificar faixas de IP:

CIDR	Range	Addresses
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

Você pode misturar IPs exatos e faixas CIDR na mesma lista:

```
sgcWebSocketFirewall1->Blacklist->IPs->Add("203.0.113.50"); // single IP
sgcWebSocketFirewall1->Blacklist->IPs->Add("198.51.100.0/24"); // entire subnet
```

Combinando Blacklist e Whitelist

Um padrão comum é bloquear um intervalo amplo, mas permitir IPs específicos dentro desse intervalo:

```
// Block the entire 10.x.x.x range
sgcWebSocketFirewall1->Blacklist->Enabled = true;
sgcWebSocketFirewall1->Blacklist->IPs->Add("10.0.0.0/8");

// But allow the monitoring server
sgcWebSocketFirewall1->Whitelist->Enabled = true;
sgcWebSocketFirewall1->Whitelist->IPs->Add("10.1.1.50");
```

Neste exemplo, todos os IPs no intervalo 10.x.x.x são bloqueados, exceto 10.1.1.50, que está na whitelist e ignora todas as verificações.

Firewall: Proteção contra Força Bruta

O módulo de proteção contra força bruta detecta tentativas de autenticação malsucedidas repetidas a partir do mesmo endereço IP e bane automaticamente o IP infrator por uma duração configurável.

Como funciona

O firewall rastreia as tentativas malsucedidas por endereço IP dentro de uma janela de tempo deslizando. Quando o número de falhas excede o limiar configurado, o IP é banido automaticamente. IPs banidos são rejeitados no nível da conexão, antes de qualquer evento do servidor ser disparado.

1. Um cliente conecta e tenta a autenticação.
2. Se a autenticação falhar, o seu código chama **RegisterFailedAttempt(IP)** para notificar o firewall.
3. O firewall incrementa o contador de falhas para aquele IP dentro da janela de tempo atual.
4. Se o contador atingir **MaxAttempts**, o IP é banido por **BanDurationSec** segundos.
5. Qualquer nova conexão de um IP banido é rejeitada imediatamente.
6. Após a duração do banimento expirar, o IP é automaticamente desbanido e o contador de falhas é resetado.

Properties

Property	De-fault	Descrição
BruteForce.Enabled	False	Habilita ou desabilita a proteção contra força bruta.
BruteForce.MaxAttempts	5	Número máximo de tentativas falhadas permitidas dentro da janela de tempo antes de o IP ser banido.
BruteForce.TimeWindowSec	60	Duração da janela de tempo deslizando em segundos. Tentativas com falha mais antigas que isto são descartadas.
BruteForce.BanDurationSec	300	Duração do banimento automático em segundos. Defina como 0 para um banimento permanente (até ser removido manualmente ou o servidor ser reiniciado).

RegisterFailedAttempt

O firewall não sabe quais conexões representam logins com falha. Você deve chamar **RegisterFailedAttempt** a partir do manipulador do seu evento de autenticação sempre que uma tentativa de login falhar:

```
void __fastcall TForm1::ServerAuthentication(TsgcWSCConnection *Connection,
String aUser, String aPassword, bool &Authenticated)
{
    Authenticated = CheckCredentials(aUser, aPassword);
    if (!Authenticated)
        sgcWebSocketFirewall1->RegisterFailedAttempt(Connection->IP);
}
```

Gerenciamento Manual de Banimentos

Além do banimento automático, você pode gerenciar banimentos manualmente:

Method	Descrição

BanIP(IP, Duration-Sec)	Bane manualmente um IP. Utilize DurationSec = 0 para um banimento permanente.
UnbanIP(IP)	Remove um banimento do IP especificado.
IsBanned(IP)	Retorna True se o IP estiver atualmente banido.
ClearBans	Remove todos os bans ativos (tanto automáticos quanto manuais).

Exemplo: Ban Manual

```
// Ban an IP for 1 hour
sgcWebSocketFirewall1->BanIP("203.0.113.50", 3600);

// Permanent ban
sgcWebSocketFirewall1->BanIP("198.51.100.25", 0);

// Check and unban
if (sgcWebSocketFirewall1->IsBanned("203.0.113.50"))
    sgcWebSocketFirewall1->UnbanIP("203.0.113.50");
```

Exemplo de Configuração

```
// Strict configuration: 3 attempts in 30 seconds, ban for 10 minutes
sgcWebSocketFirewall1->BruteForce->Enabled = true;
sgcWebSocketFirewall1->BruteForce->MaxAttempts = 3;
sgcWebSocketFirewall1->BruteForce->TimeWindowSec = 30;
sgcWebSocketFirewall1->BruteForce->BanDurationSec = 600;
```

```
// Lenient configuration: 10 attempts in 5 minutes, ban for 1 minute
sgcWebSocketFirewall1->BruteForce->Enabled = true;
sgcWebSocketFirewall1->BruteForce->MaxAttempts = 10;
sgcWebSocketFirewall1->BruteForce->TimeWindowSec = 300;
sgcWebSocketFirewall1->BruteForce->BanDurationSec = 60;
```

Notas

- IPs na lista de permissões (whitelist) nunca são banidos, mesmo que RegisterFailedAttempt seja chamado para eles.
- Os banimentos são armazenados em memória por padrão. Use **SaveBansToFile** e **LoadBansFromFile** para persistir os banimentos entre reinícios do servidor.
- A janela de tempo é deslizante, não fixa. Cada tentativa com falha é rastreada individualmente e expira após TimeWindowSec segundos.

Firewall: Detecção de SQL Injection e XSS

O TsgcWebSocketFirewall inclui detecção integrada baseada em padrões para ataques de SQL injection e Cross-Site Scripting (XSS) em mensagens WebSocket. Esses módulos verificam o conteúdo das mensagens recebidas e tomam medidas quando um padrão de ameaça é detectado.

Detecção de Injeção SQL

O módulo de SQL injection examina todas as mensagens WebSocket de entrada em busca de padrões comuns de SQL injection. A detecção não diferencia maiúsculas de minúsculas.

Properties

Property	Descrição
SQLInjection.Enabled	Habilita ou desabilita a detecção de injeção de SQL. Padrão: False.
SQLInjection.Action	Ação a tomar quando um padrão é detectado: faDeny (bloqueia a mensagem e desconecta, padrão), faLog (dispara o evento OnViolation mas permite a mensagem), faAllow (nenhuma ação, detecção desabilitada).
SQLInjection.CustomPatterns	TStringList de padrões regex adicionais a serem verificados. Estes são avaliados além dos padrões integrados.

Padrões Integrados

Os seguintes padrões são verificados por padrão (sem distinção de maiúsculas/minúsculas):

Pattern	Tipo de Ataque
' OR ', ' AND '	Injeção baseada em booleano
UNION SELECT	Injeção baseada em union
'; DROP, '; DELETE, '; INSERT, '; UPDATE	Injeção de instruções / manipulação de dados
-- (double dash)	Injeção de comentário SQL
EXEC(, EXECUTE(	Execução de comando
xp_cmdshell	Abuso de stored procedure estendida
CAST(, CONVERT(	Abuso de conversão de tipo
1=1, 1='1	Injeção por tautologia

Custom Patterns

Você pode adicionar padrões específicos da aplicação usando a propriedade CustomPatterns:

```
sgcWebSocketFirewall1->SQLInjection->Enabled = true;
sgcWebSocketFirewall1->SQLInjection->CustomPatterns->Add("WAITFOR DELAY");
sgcWebSocketFirewall1->SQLInjection->CustomPatterns->Add("BENCHMARK\\(");
sgcWebSocketFirewall1->SQLInjection->CustomPatterns->Add("SLEEP\\(");
```

XSS Detection

O módulo XSS detecta padrões de Cross-Site Scripting em mensagens WebSocket, prevenindo a injeção de scripts maliciosos que poderiam ser renderizados por outros clientes conectados.

Properties

Property	Descrição
XSS.Enabled	Habilita ou desabilita a detecção de XSS. Padrão: False.
XSS.Action	Ação a tomar quando um padrão é detectado: faDeny (bloquear e desconectar, padrão), faLog (dispara OnViolation mas permite), faAllow (nenhuma ação).

Padrões Integrados

Os seguintes padrões XSS são detectados (sem distinção entre maiúsculas e minúsculas):

Pattern	Tipo de Ataque
<script	Injeção de tag de script
javascript:	XSS baseado em protocolo
onerror=, onload=, onclick=, onmouseover=	Injeção de manipulador de evento
eval(	Avaliação de código JavaScript
document.cookie	Roubo de cookie
<iframe, <object, <embed	Injeção de elemento
<svg onload	XSS baseado em SVG
expression(	Injeção de expressão CSS

Action Property

Tanto o módulo SQLInjection quanto o XSS suportam uma propriedade Action do tipo TsgcFirewallAction:

Value	Comportamento
faDeny	A mensagem é bloqueada e o cliente é desconectado. O evento OnViolation é disparado. Este é o padrão.
faLog	A mensagem é permitida, mas o evento OnViolation é disparado para fins de registro em log.

faAllow	Nenhuma ação é tomada. Na prática, desabilita a detecção mantendo a flag Enabled como True.

OnViolation Event

Quando um padrão de SQL injection ou XSS é detectado (com Action definido como faDeny ou faLog), o evento OnViolation dispara com detalhes sobre a violação:

```
void __fastcall TForm1::FirewallViolation(TObject *Sender, const String aIP,
const TsgcFirewallViolationType aViolationType, const String aDetails)
{
    switch (aViolationType)
    {
        case fvSQLInjection:
            LogWarning("SQL injection attempt from " + aIP + ": " + aDetails);
            break;
        case fvXSS:
            LogWarning("XSS attempt from " + aIP + ": " + aDetails);
            break;
    }
}
```

Exemplo de Configuração

```
// Enable SQL injection detection with deny action
sgcWebSocketFirewall1->SQLInjection->Enabled = true;
sgcWebSocketFirewall1->SQLInjection->Action = faDeny;

// Add custom patterns for time-based SQL injection
sgcWebSocketFirewall1->SQLInjection->CustomPatterns->Add("WAITFOR DELAY");
sgcWebSocketFirewall1->SQLInjection->CustomPatterns->Add("SLEEP\\(");

// Enable XSS detection in log-only mode (monitor without blocking)
sgcWebSocketFirewall1->XSS->Enabled = true;
sgcWebSocketFirewall1->XSS->Action = faLog;

// Assign violation handler
sgcWebSocketFirewall1->OnViolation = FirewallViolation;
```

Notas

- A detecção de padrões é aplicada a todas as mensagens de texto de entrada. As mensagens binárias não são verificadas.
- IPs na whitelist contornam completamente a filtragem de mensagens.
- Os padrões integrados cobrem os vetores de ataque mais comuns. Para ameaças específicas da aplicação, adicione padrões personalizados via SQLInjection.CustomPatterns.
- Use o modo faLog durante a implantação inicial para monitorar as detecções sem impactar os clientes e, em seguida, mude para faDeny depois que você tiver validado os padrões para a sua aplicação.

Firewall: Rate Limiting e Proteção contra Flood

TsgcWebSocketFirewall fornece dois mecanismos complementares para controlar o tráfego: limitação de taxa de conexão (controlando quantas conexões um IP pode abrir) e proteção contra inundação de mensagens (controlando quantas mensagens um cliente conectado pode enviar por segundo).

Limitação de Taxa de Conexões

O rate limiting restringe o número de conexões que um único endereço IP pode estabelecer dentro de uma janela de tempo. Isso impede que um único cliente consuma todas as conexões disponíveis do servidor.

Properties

Property	De-fault	Descrição
RateLimit.Enabled	False	Habilita ou desabilita a limitação de taxa de conexões.
RateLimit.MaxConnectionsPerIP	10	Número máximo de conexões simultâneas permitidas a partir de um único endereço IP.
RateLimit.TimeWindowSec	60	Janela de tempo em segundos para rastrear conexões. Conexões mais antigas que isto são removidas da contagem.

Como funciona

1. Quando uma nova conexão chega, o firewall conta o número de conexões ativas a partir daquele IP.
2. Se a contagem estiver igual ou acima de MaxConnectionsPerIP, a nova conexão é rejeitada.
3. Quando uma conexão se desconecta, ela é removida da contagem de rastreamento.
4. Conexões mais antigas que TimeWindowSec são removidas automaticamente do rastreamento.

Exemplo

```
// Allow maximum 5 connections per IP
sgcWebSocketFirewall1->RateLimit->Enabled = true;
sgcWebSocketFirewall1->RateLimit->MaxConnectionsPerIP = 5;
sgcWebSocketFirewall1->RateLimit->TimeWindowSec = 60;
```

Message Flood Protection

A proteção contra flood limita o número de mensagens que um único IP pode enviar por segundo. Isto evita que clientes abusivos sobrecarreguem o servidor com bursts rápidos de mensagens.

Properties

Property	De-fault	Descrição

FloodProtection.Enabled	False	Habilita ou desabilita a proteção contra flood de mensagens.
FloodProtection.MaxMessagesPerSec	100	Número máximo de mensagens permitidas por segundo a partir de um único endereço IP.
FloodProtection.Action	faDeny	Ação quando o limite é excedido: faDeny (desconectar o cliente), faLog (disparar OnViolation mas permitir), faAllow (nenhuma ação).

Como funciona

1. O firewall mantém um contador de mensagens por IP que é redefinido a cada segundo.
2. Cada mensagem recebida incrementa o contador do IP do remetente.
3. Quando o contador excede MaxMessagesPerSec, a Action configurada é tomada.
4. Com a ação faDeny, o cliente é desconectado imediatamente. Com faLog, o evento OnViolation dispara, mas a mensagem é processada normalmente.

Exemplo

```
// Limit to 50 messages per second per IP
sgcWebSocketFirewall1->FloodProtection->Enabled = true;
sgcWebSocketFirewall1->FloodProtection->MaxMessagesPerSec = 50;
sgcWebSocketFirewall1->FloodProtection->Action = faDeny;
```

Combinando Ambas as Proteções

A limitação de taxa e a proteção contra flood trabalham juntas para fornecer um controle de tráfego abrangente:

```
// Connection limiting: max 5 connections per IP
sgcWebSocketFirewall1->RateLimit->Enabled = true;
sgcWebSocketFirewall1->RateLimit->MaxConnectionsPerIP = 5;

// Message limiting: max 50 messages/sec per IP
sgcWebSocketFirewall1->FloodProtection->Enabled = true;
sgcWebSocketFirewall1->FloodProtection->MaxMessagesPerSec = 50;

// Assign to server
sgcWebSocketHTTPServer1->Firewall = sgcWebSocketFirewall1;
```

Com esta configuração, cada IP pode abrir no máximo 5 conexões concorrentes e, em todas essas conexões, pode enviar um máximo combinado de 50 mensagens por segundo.

Tuning Guidelines

Scenario	MaxConnectionsPerIP	MaxMessagesPerSec
Aplicação de chat	3-5	10-30
Feed de dados em tempo real	2-3	100-500
Servidor de jogos	1-2	50-100
API gateway	10-20	200-1000

Comece com limites conservadores e aumente-os com base no monitoramento. Utilize o modo faLog inicialmente para observar os padrões de tráfego antes de habilitar o faDeny.

Notas

- IPs na whitelist contornam tanto a limitação de taxa quanto a proteção contra flood.
- Os dados de rastreamento de limite de taxa são limpos automaticamente quando as conexões são fechadas. Use ClearTracking para redefinir manualmente todos os contadores.
- Atrás de um reverse proxy, todas as conexões podem parecer vir do IP do proxy. Configure o seu proxy para passar o IP real do cliente (por exemplo, cabeçalho X-Forwarded-For) e certifique-se de que o servidor o resolva corretamente.

Firewall: Detecção de Bots

TsgcWebSocketFirewall pode classificar os endereços IP conectados como bots, crawlers, hosts de datacenter ou visitantes humanos. A Detecção de Bots combina um banco de dados de bots conhecidos, verificação de DNS reverso confirmado, correspondência de faixas de datacenter/ASN e consultas de blocklist DNSBL para atribuir uma classificação a cada IP. É um módulo **somente de classificação**: ele nunca bloqueia uma conexão por conta própria. Ele relata o veredito para que sua aplicação possa decidir o que fazer.

Como funciona

1. Um IP conecta ao servidor.
2. O firewall busca o IP no cache de resultados. Se existir uma classificação em cache (e não tiver expirado), ela é utilizada imediatamente.
3. Em uma falha de cache, o IP é comparado com o arquivo de bots conhecidos (crawlers verificados) e com o arquivo de ASN de datacenter (faixas de cloud/hosting).
4. Se a verificação ReverseDNS ou DNSBL estiver habilitada, uma consulta é agendada em uma thread em segundo plano para que nunca bloqueie a thread de accept. Até que a consulta seja concluída, o IP pode ser classificado como **bcUnknown**.
5. Quando uma classificação é resolvida, ela é armazenada no cache (por CacheDurationSec segundos) e o evento **OnBotDetected** é disparado.
6. A classificação atualiza as Stats do firewall (o contador de violação **fvBot**) e está disponível para as Custom Rules através da propriedade **BotType** da regra.

Properties

Property	Descrição
BotDetection.Enabled	Habilita ou desabilita a detecção de bots. Padrão: False.
BotDetection.KnownBotsFile	Caminho para um arquivo CSV de crawlers verificados. Cada linha é CIDR,botname (por exemplo, um intervalo de IP e o crawler a que pertence).
BotDetection.CacheDurationSec	Número de segundos que uma classificação resolvida é mantida no cache de resultados antes de ser consultada novamente. Padrão: 3600.
BotDetection.DNSTimeoutMS	Timeout em milissegundos para cada lookup DNS (DNS reverso e DNSBL). Padrão: 2000.
BotDetection.ReverseDNS.Enabled	Habilita a verificação forward-confirmed reverse DNS (FCrDNS) de crawlers declarados. Padrão: False.
BotDetection.Datacenter.Enabled	Habilita a detecção de faixas de datacenter / cloud-hosting. Padrão: False.
BotDetection.Datacenter.ASNFile	Caminho para um arquivo CSV de ranges de ASN de datacenter. Cada linha é CIDR,asname (por exemplo, um range de provedor de nuvem e o nome de seu ASN).
BotDetection.DNSBL.Enabled	Habilita as consultas DNSBL (lista de bloqueio DNS). Padrão: False.

BotDetection.DNSBL.Zones

TStringList de zonas DNSBL a consultar (por exemplo zen.spamhaus.org).

Classification Values

A enumeração **TsgcBotClassification** descreve o veredito atribuído a um IP:

Value	Descrição
bcUnknown	O IP ainda não foi classificado. Retornado em uma falha de cache enquanto uma busca assíncrona está em andamento.
bcVerified-Crawler	Um crawler conhecido e legítimo (por exemplo, um bot de mecanismo de busca) confirmado pelo arquivo de bots conhecidos ou por DNS reverso confirmado por forward.
bcDatacenter	O IP pertence a um intervalo de provedor de nuvem ou hospedagem listado no arquivo ASN. Frequentemente tráfego automatizado em vez de um usuário residencial.
bcSuspectedBot	O IP mostra características de bot, mas não pôde ser associado a um crawler verificado.
bcBlocklisted	O IP está listado em uma das zonas DNSBL configuradas.
bcHuman	O IP parece um visitante humano comum (sem correspondência de bot, datacenter ou blocklist).

Formatos de arquivo CSV

O arquivo de known-bots (**KnownBotsFile**) lista faixas de crawlers verificados. Cada linha é uma faixa CIDR seguida pelo nome do bot:

```
66.249.64.0/19,Googlebot
157.55.0.0/16,Bingbot
17.58.0.0/16,Applebot
```

O arquivo de ASN de datacenter (**Datacenter.ASNFile**) lista os ranges de nuvem/hospedagem. Cada linha é um range CIDR seguido pelo nome do ASN:

```
3.0.0.0/15,AMAZON-AES
13.64.0.0/11,MICROSOFT-AZURE
34.64.0.0/10,GOOGLE-CLOUD
```

Habilitando a Detecção de Bots

Habilite o módulo, carregue os arquivos de known-bot e ASN, e ative os submódulos de verificação de que você precisa. As buscas DNS são executadas em uma thread em segundo plano, portanto habilitar ReverseDNS ou DNSBL não desacelera a thread de accept.

```
// Enable bot detection
sgcWebSocketFirewall1->BotDetection->Enabled = true;
sgcWebSocketFirewall1->BotDetection->KnownBotsFile = "C:\\data\\known_bots.csv";
sgcWebSocketFirewall1->BotDetection->CacheDurationSec = 3600;
sgcWebSocketFirewall1->BotDetection->DNSTimeoutMS = 2000;
```

```
// Forward-confirmed reverse DNS verification
sgcWebSocketFirewall1->BotDetection->ReverseDNS->Enabled = true;

// Datacenter / cloud-hosting detection
sgcWebSocketFirewall1->BotDetection->Datacenter->Enabled = true;
sgcWebSocketFirewall1->BotDetection->Datacenter->ASNFile = "C:\\data\\datacenter_asn.csv";

// DNSBL blocklist lookups
sgcWebSocketFirewall1->BotDetection->DNSBL->Enabled = true;
sgcWebSocketFirewall1->BotDetection->DNSBL->Zones->Add("zen.spamhaus.org");
```

Comportamento Classify-Only e OnBotDetected

A Detecção de Bots nunca rejeita uma conexão por conta própria. Quando um IP é classificado, ela dispara o evento **OnBotDetected**, atualiza as Stats do firewall (o contador **fvBot**) e armazena o resultado no cache. Utilize o evento para registrar o tráfego, criar análises ou alimentar sua própria lógica:

```
void __fastcall TForm1::FirewallBotDetected(TObject *Sender, const String aIP,
    TsgcBotClassification aClassification, const String aBotName)
{
    switch (aClassification)
    {
        case bcVerifiedCrawler: Log("Verified crawler " + aBotName + " from " + aIP); break;
        case bcDatacenter:     Log("Datacenter host from " + aIP); break;
        case bcBlocklisted:    Log("Blocklisted IP " + aIP); break;
    }
}
```

Agir sobre uma Classificação com uma Custom Rule

Como a Bot Detection apenas classifica, o bloqueio ou banimento é feito através do motor de Custom Rules. Cada **TsgcFirewallRuleItem** expõe uma propriedade **BotType**: quando definida, a regra corresponde a conexões cuja classificação é igual àquele valor, e você pode então negá-las, bani-las ou registrá-las.

```
sgcWebSocketFirewall1->CustomRules->Enabled = true;

// Deny connections coming from blocklisted IPs
TsgcFirewallRuleItem *vRule =
    (TsgcFirewallRuleItem *)sgcWebSocketFirewall1->CustomRules->Rules->Add();
vRule->BotType = bcBlocklisted;
vRule->ActionType = raDeny;
```

Veredictos Personalizados com OnResolveBot

Se você tiver sua própria inteligência de bots (um serviço de reputação de terceiros, uma lista interna de permissão/negação, etc.), atribua o evento **OnResolveBot**. Quando atribuído, seu manipulador fornece o veredito em vez do resolvidor integrado. Defina os parâmetros *var aClassification* e *aBotName* para retornar seu resultado:

```
void __fastcall TForm1::FirewallResolveBot(TObject *Sender, const String aIP,
    TsgcBotClassification &aClassification, String &aBotName)
{
    // Custom lookup - query your own bot intelligence service
    if (MyBotService->IsCrawler(aIP))
    {
        aClassification = bcVerifiedCrawler;
        aBotName = MyBotService->GetName(aIP);
    }
    else
        aClassification = bcHuman;
}
```

Nota: Quando **OnResolveBot** está atribuído, o resolvidor integrado (arquivo de bots conhecidos, DNS reverso, verificações de datacenter e DNSBL) é ignorado para esse IP. O veredito que seu handler retorna é armazenado em cache e repassado para **OnBotDetected** e Custom Rules exatamente como um resultado integrado.

TsgcWSRateLimiter

TsgcWSRateLimiter implementa um componente completo de rate limiting que protege endpoints do servidor contra tráfego excessivo, abuso e scraping.

Introdução

O TsgcWSRateLimiter implementa um componente completo de limitação de taxa que protege os endpoints do servidor contra tráfego excessivo, abuso e scraping. Ele suporta três algoritmos (Token Bucket, Sliding Window, Fixed Window), regras com escopo (por IP, por chave de API, por usuário, por endpoint), cotas de longo prazo (hora / dia / mês), proteção contra rajadas, um formato de resposta HTTP 429 configurável e estatísticas em tempo real. O estado interno pode ser persistido em um arquivo ou stream para que os contadores sobrevivam a reinicializações.

O rate limiter se integra automaticamente aos componentes de servidor através da propriedade **RateLimiter** disponível em TsgcWebSocketHTTPServer e TsgcWSServer_HTTPAPI. Uma vez atribuído, ele avalia cada conexão e cada mensagem sem exigir o cabeamento manual de eventos.

Início rápido

1. Solte um componente **TsgcWSRateLimiter** no formulário.
2. Habilite uma ou mais estratégias e escopos (veja a documentação por seção abaixo).
3. Atribua o rate limiter a um componente servidor:

```
sgcWebSocketHTTPServer1.RateLimiter := sgcWSRateLimiter1;
```

4. Inicie o servidor. O rate limiter avalia automaticamente cada conexão e mensagem, rejeita o tráfego excedente com HTTP 429 e atualiza seus contadores de estatísticas.

Propriedades de nível superior

O que faz: Interruptor mestre do componente e arquivo de armazenamento opcional usado pelos métodos de persistência de estado. Quando Enabled é False, cada método público faz curto-circuito para "permitido" e nenhum contador é atualizado.

Quando utilizá-lo: Desabilite o limitador de taxa em tempo de execução durante um teste de carga controlado ou durante uma janela de manutenção, sem desatribuí-lo do servidor. Defina StorageFile para que os contadores sobrevivam a uma reinicialização da aplicação. Por exemplo, um Token Bucket rolante continua reabastecendo de onde parou, em vez de começar do zero e aceitar uma rajada enorme.

Pro- perty	Type	De- fault	Descrição
Enabled	Boole- an	True	Interruptor mestre. Quando False, IsAllowed / Consume / IsConnectionAllowed / IsMessageAllowed sempre retornam True e nenhum contador é atualizado.

Storage-File	string	"	Caminho de arquivo opcional usado por SaveStateToFile / LoadStateFromFile para a persistência de contadores entre reinicializações.

```
sgcWSRateLimiter1->Enabled = true;
sgcWSRateLimiter1->StorageFile = "ratelimit.dat";
sgcWSRateLimiter1->LoadStateFromFile(sgcWSRateLimiter1->StorageFile);
```

TokenBucket Strategy

O que ele faz: Mantém um bucket de até *Capacity* tokens por chave. O bucket é reabastecido a *RefillRate* tokens a cada *RefillIntervalMs* milissegundos. Cada requisição consome um token (ou o argumento *Cost* passado a *Consume*). Quando o bucket está vazio, a requisição é rejeitada e um hint *RetryAfterSec* é calculado a partir do cronograma de reabastecimento.

Quando usá-lo: A API anuncia uma taxa média de 10 req/s, mas você quer permitir que clientes bem-comportados façam rajadas de até 50 requisições consecutivas sem serem limitados. Um Token Bucket com *Capacity*=50 e *RefillRate*=10 por 1000 ms permite que um cliente envie 50 requisições instantaneamente e, em seguida, força-o a reduzir para 10 req/s até que o bucket seja reabastecido.

Property	Type	De-fault	Descrição
Enabled	Boolean	False	Habilita o algoritmo Token Bucket como a estratégia padrão global.
Capacity	Integer	100	Número máximo de tokens que o bucket pode conter. Controla o tamanho máximo do burst.
RefillRate	Integer	10	Número de tokens adicionados ao bucket a cada <i>RefillIntervalMs</i> .
RefillIntervalMs	Integer	1000	Período de reabastecimento em milissegundos. Juntamente com <i>RefillRate</i> , determina a taxa sustentada de requisições.

```
// Average 10 req/sec, bursts up to 50
sgcWSRateLimiter1->TokenBucket->Enabled = true;
sgcWSRateLimiter1->TokenBucket->Capacity = 50;
sgcWSRateLimiter1->TokenBucket->RefillRate = 10;
sgcWSRateLimiter1->TokenBucket->RefillIntervalMs = 1000;
// First 50 requests succeed immediately; afterwards throttled to 10/sec until refill
```

SlidingWindow Strategy

O que faz: Armazena um timestamp para cada requisição e conta os timestamps que se enquadram em uma janela móvel de *WindowSec* segundos. Rejeita quando a contagem excederia *MaxRequests*. A janela desliza continuamente, portanto não há um momento de reset abrupto em que uma rajada possa passar despercebida.

Quando usar: Você publica um SLA estrito garantindo "não mais que 100 requisições em qualquer janela deslizando de 60 segundos". O Fixed Window permitiria que um cliente disparasse 100 requisições às 12:00:59 e outras 100 às 12:01:00 (200 dentro de um segundo). O Sliding Window evita isso e impõe o SLA exatamente.

Property	Type	Default	Descrição
Enabled	Boolean	False	Habilita o algoritmo Sliding Window como a estratégia padrão global.
Window-Sec	Integer	60	Tamanho da janela deslizando em segundos.
MaxRequests	Integer	1000	Número máximo de requisições permitidas em qualquer janela deslizando de WindowSec de largura.

```
// Strict SLA: max 100 requests in any rolling 60 seconds
sgcWSRateLimiter1->SlidingWindow->Enabled = true;
sgcWSRateLimiter1->SlidingWindow->WindowSec = 60;
sgcWSRateLimiter1->SlidingWindow->MaxRequests = 100;
// Client that sends a burst of 100 requests near t=59s will be throttled at t=60s
```

Estratégia FixedWindow

O que faz: Conta as requisições dentro de uma janela de relógio fixa de *WindowSec* segundos e zera o contador no limite da janela. É o algoritmo mais simples e o mais econômico em memória; no entanto, pode permitir até $2 \times \text{MaxRequests}$ em um curto período em torno do cruzamento de um limite.

Quando utilizá-lo: Cotas no estilo de cobrança, como "1000 requisições por hora, contador zera na virada da hora". Os clientes esperam um horário de reset claramente visível, e o leve comportamento de burst nos limites é aceitável porque a unidade de cobrança é a hora, não o minuto móvel.

Property	Type	Default	Descrição
Enabled	Boolean	False	Habilita o algoritmo Fixed Window como a estratégia padrão global.
Window-Sec	Integer	60	Tamanho da janela fixa em segundos. O contador é resetado para 0 quando a janela decorre.
MaxRequests	Integer	1000	Número máximo de requisições permitidas dentro de uma única janela fixa.

```
// Billing-style: 1000 requests per hour, resets on the hour
sgcWSRateLimiter1->FixedWindow->Enabled = true;
sgcWSRateLimiter1->FixedWindow->WindowSec = 3600;
sgcWSRateLimiter1->FixedWindow->MaxRequests = 1000;
```


PerIP

O que faz: Define um limite de taxa com escopo no endereço IP de origem. Quando uma chave de requisição não corresponde a PerAPIKey / PerUser / PerEndpoint, PerIP se torna a regra ativa e substitui os padrões globais Token/Sliding/Fixed. Cada IP obtém seu próprio contador independente.

Quando usá-lo: Uma API pública com muitos clientes. Sem PerIP, um único cliente ruidoso pode consumir o orçamento global e privar todos os outros. Dê a cada IP seu próprio bucket (por exemplo, 100 requisições por minuto) para que um cliente não possa afetar os outros.

Property	Type	Default	Descrição
Enabled	Boolean	False	Habilita o limite de taxa por IP.
MaxRequests	Integer	100	Máximo de requisições por janela (ou capacidade do bucket quando Strategy é rlsTokenBucket).
WindowSec	Integer	60	Tamanho da janela em segundos. Para Token Bucket, também utilizado para derivar a taxa de reabastecimento (MaxRequests / WindowSec tokens por segundo).
Strategy	TsgcRateLimitStrategy	rlsTokenBucket	Algoritmo utilizado para este escopo: rlsTokenBucket, rlsSliding ou rlsFixed.

```
// Every client IP gets its own 100 req/min bucket
sgcWSRateLimiter1->PerIP->Enabled = true;
sgcWSRateLimiter1->PerIP->Strategy = rlsTokenBucket;
sgcWSRateLimiter1->PerIP->MaxRequests = 100;
sgcWSRateLimiter1->PerIP->WindowSec = 60;
```

PerAPIKey

O que faz: Mesma semântica de PerIP, mas com escopo de um identificador de chave de API. O componente reconhece chaves que começam com o prefixo apikey: como tendo escopo de chave de API. Projetado para integrar-se com TsgcWSAPIKeyManager para que cada cliente receba um orçamento de taxa independente, independentemente do IP de origem.

Quando utilizá-lo: SaaS com preços em camadas. Cada cliente tem sua própria chave de API e sua própria cota, independente de onde se conecta. Utilize PerAPIKey como a camada padrão e diferencie as camadas dos clientes (Free / Pro / Enterprise), seja por meio de uma regra PerEndpoint indexada por prefixo de caminho, seja sobrecrevendo a decisão dentro do handler do evento OnThrottled.

Property	Type	Default	Descrição
Enabled	Boolean	False	Habilita a limitação de taxa por chave de API.

MaxRequests	Integer	100	Máximo de requisições por janela (ou capacidade do bucket para Token Bucket).
WindowSec	Integer	60	Tamanho da janela em segundos.
Strategy	TsgcRate-LimitStrategy	rlsToken-Bucket	Algoritmo utilizado para este escopo.

```
// Default API-key tier: 1000 req/min
sgcWSRateLimiter1->PerAPIKey->Enabled = true;
sgcWSRateLimiter1->PerAPIKey->Strategy = rlsSliding;
sgcWSRateLimiter1->PerAPIKey->MaxRequests = 1000;
sgcWSRateLimiter1->PerAPIKey->WindowSec = 60;
// Tier-differentiation can be done via PerEndpoint rules or OnThrottled
```

PerUser

O que faz: Mesma estrutura que PerIP / PerAPIKey, mas indexada por um identificador de usuário definido pela aplicação. O componente reconhece como tendo escopo de usuário as chaves que começam com o prefixo `user:.`. Útil quando um único usuário pode se conectar a partir de múltiplos dispositivos ou IPs e você quer um único orçamento compartilhado para esse usuário.

Quando usá-lo: Aplicativo web/mobile autenticado. A usuária `alice@example.com` tem permissão para 1000 requisições por dia, independentemente de chamar do telefone, do laptop ou do desktop do trabalho. Agregar por IP contaria em dobro; agregar por usuário fornece o orçamento que você realmente quer cobrar.

Property	Type	Default	Descrição
Enabled	Boolean	False	Habilita o rate limiting por usuário.
MaxRequests	Integer	100	Máximo de requisições por janela (ou capacidade do bucket para Token Bucket).
WindowSec	Integer	60	Tamanho da janela em segundos.
Strategy	TsgcRate-LimitStrategy	rlsToken-Bucket	Algoritmo utilizado para este escopo.

```
// 1000 requests/day per authenticated user across all their devices
sgcWSRateLimiter1->PerUser->Enabled = true;
sgcWSRateLimiter1->PerUser->Strategy = rlsFixed;
sgcWSRateLimiter1->PerUser->MaxRequests = 1000;
sgcWSRateLimiter1->PerUser->WindowSec = 86400;
// Pass keys like "user:alice@example.com" to Consume / IsAllowed
```

PerEndpoint

O que ele faz: Coleção de regras baseadas em padrões. Cada item tem seu próprio *Pattern* (wildcards * e ? suportados), *Strategy*, *MaxRequests* e *WindowSec*. O primeiro item habilitado cujo padrão corresponde à chave vence — nenhum item adicional é avaliado. Tem precedência sobre PerAPIKey, PerUser e PerIP.

Quando utilizar: Você tem um endpoint custoso `/api/v1/expensive-report` que deve ser limitado a 10 req/min por cliente, enquanto um endpoint de status barato `/api/v1/status` permite 1000 req/min. Endpoints diferentes têm orçamentos diferentes. Configure um item PerEndpoint por padrão de caminho.

Property	Type	Default	Descrição
Enabled	Boolean	False	Habilita o conjunto de regras PerEndpoint. Quando False, nenhum item é avaliado.
Regras	TCollection	vazio	Coleção de TsgcRateLimitRuleItem. A ordem importa — a primeira correspondência vence.

Cada **TsgcRateLimitRuleItem** expõe:

Property	Type	Default	Descrição
Name	string	"	Nome amigável exibido no object inspector.
Enabled	Boolean	True	Desabilita este item sem excluí-lo.
Pattern	string	"	Padrão wildcard comparado com a chave (não diferencia maiúsculas de minúsculas). Suporta * e ?.
MaxRequests	Integer	100	Máximo de requisições por janela / capacidade do bucket.
WindowSec	Integer	60	Tamanho da janela em segundos.
Strategy	TsgcRateLimitStrategy	rlsTokenBucket	Algoritmo a ser utilizado para requisições correspondentes.

```
sgcWSRateLimiter1->PerEndpoint->Enabled = true;

TsgcRateLimitRuleItem *ruleReport = (TsgcRateLimitRuleItem*)
    sgcWSRateLimiter1->PerEndpoint->Rules->Add();
ruleReport->Name = "expensive-report";
ruleReport->Pattern = "*/api/v1/expensive-report*";
ruleReport->Strategy = rlsSliding;
ruleReport->MaxRequests = 10;
ruleReport->WindowSec = 60;

TsgcRateLimitRuleItem *ruleStatus = (TsgcRateLimitRuleItem*)
    sgcWSRateLimiter1->PerEndpoint->Rules->Add();
ruleStatus->Name = "status";
ruleStatus->Pattern = "*/api/v1/status*";
ruleStatus->Strategy = rlsTokenBucket;
ruleStatus->MaxRequests = 1000;
```

```
ruleStatus->WindowSec = 60;
```

Cotas

O que faz: Limites de longo prazo que persistem além da janela móvel / de recarga da estratégia principal. Cada item limita quantas requisições uma chave (ou todo o sistema) pode fazer dentro de um período de calendário (*qpHour*, *qpDay* ou *qpMonth*). Uma vez que o *Limit* é atingido, a requisição é rejeitada e o evento *OnQuotaExceeded* dispara.

Quando utilizá-lo: Plano de nível gratuito limitado a um teto rígido de 10.000 requisições por mês. O token bucket contínuo ainda suaviza o tráfego de curto prazo, mas quando a cota mensal é esgotada, cada chamada subsequente falha até o próximo mês do calendário.

Property	Type	Default	Descrição
Enabled	Boolean	False	Habilita o conjunto de regras de cota. Quando False, nenhum item é avaliado.
Cotas	TCollection	vazio	Coleção de TsgcRateLimitQuotaItem. Todos os itens habilitados são avaliados em ordem.

Cada **TsgcRateLimitQuotaItem** expõe:

Property	Type	Default	Descrição
Name	string	"	Nome amigável reportado em <i>OnQuotaExceeded</i> e em <i>Result.Reason</i> ("quota:Name").
Enabled	Boolean	True	Desabilita esta quota sem excluí-la.
Scope	TsgcRateLimitQuotaScope	qscPerKey	qscPerKey conta independentemente por chave; qscGlobal conta em todas as chaves combinadas.
Period	TsgcRateLimitQuotaPeriod	qpHour	qpHour, qpDay ou qpMonth. O contador é reiniciado automaticamente no início de cada período.
Limit	Int64	10000	Número máximo de requisições permitidas dentro do período.

```
sgcWSRateLimiter1->Quotas->Enabled = true;

TsgcRateLimitQuotaItem *quota = (TsgcRateLimitQuotaItem*)
    sgcWSRateLimiter1->Quotas->Quotas->Add();
quota->Name = "free-tier-monthly";
quota->Scope = qscPerKey;
quota->Period = qpMonth;
```

```
quota->Limit = 10000;
```

BurstProtection

O que faz: Um detector de picos de curta escala de tempo que é executado em paralelo com a estratégia principal. Se mais de *BurstThreshold* requisições chegarem da mesma chave dentro de *BurstWindowMs* milissegundos, a chave é colocada em um período de espera (cooldown) por *CooldownSec* segundos, durante o qual toda requisição é rejeitada, mesmo que a estratégia principal as permitisse.

Quando utilizar: Detectar clientes scrapers e scanners que inundam um servidor com mais de 100 requisições em bursts de 100 ms, mas permanecem abaixo de um limite por minuto. A proteção contra bursts captura esse padrão de metralhadora que um usuário humano nunca produz.

Property	Type	Default	Descrição
Enabled	Boolean	False	Habilita o detector de rajadas (burst).
BurstThreshold	Integer	50	Número de requisições dentro de <i>BurstWindowMs</i> que aciona um cooldown.
BurstWindowMs	Integer	500	Janela de observação de burst em milissegundos.
CooldownSec	Integer	30	Duração do período de cooldown após um burst ser detectado. Durante o cooldown, cada requisição é rejeitada.

```
// More than 50 requests in 500ms triggers a 30-second cooldown
sgcWSRateLimiter1->BurstProtection->Enabled = true;
sgcWSRateLimiter1->BurstProtection->BurstThreshold = 50;
sgcWSRateLimiter1->BurstProtection->BurstWindowMs = 500;
sgcWSRateLimiter1->BurstProtection->CooldownSec = 30;
```

Resposta

O que faz: Configura o formato da resposta HTTP retornada aos clientes quando uma requisição é limitada (throttled). Controla o código de status, a mensagem do corpo e se os headers *Retry-After* e *X-RateLimit-** são adicionados à resposta.

Quando usá-lo: Você deseja que clientes bem comportados façam o back off corretamente. Retorne o status padrão 429 com um cabeçalho *Retry-After* indicando quanto tempo eles devem esperar, e inclua *X-RateLimit-Limit* / *X-RateLimit-Remaining* / *X-RateLimit-Reset* para que os clientes possam regular seu ritmo de forma proativa, em vez de reagir às rejeições.

Property	Type	Default	Descrição

StatusCo- de	Inte- ger	429	Código de status HTTP retornado para requisições limitadas (throttled). Utilize 429 (padrão) ou 503 para rejeição no estilo de manutenção.
Message	string	"Too Many Requests"	Corpo da resposta HTTP para requisições limitadas (throttled).
RetryAf- terHeader	Boole- an	True	Quando True, adiciona um cabeçalho <code>Retry-After</code> com o atraso de retry calculado em segundos.
IncludeRa- teLimitHe- aders	Boole- an	True	Quando True, adiciona os cabeçalhos <code>X-RateLimit-Limit</code> / <code>X-RateLimit-Remaining</code> / <code>X-RateLimit-Reset</code> a cada resposta.

```
sgcWSRateLimiter1->Response->StatusCode = 429;
sgcWSRateLimiter1->Response->Message = "Too Many Requests - please slow down";
sgcWSRateLimiter1->Response->RetryAfterHeader = true;
sgcWSRateLimiter1->Response->IncludeRateLimitHeaders = true;
```

Stats

O que faz: Objeto contador somente leitura que expõe estatísticas ao vivo sobre o rate limiter. Todos os contadores são atualizados atomicamente dentro da seção crítica interna do componente.

Quando usá-lo: Painel ao vivo em uma interface de administração. Faça polling de Stats a partir de um TTimer e exiba os valores atuais de TotalRequests / TotalThrottled / ActiveKeys em labels, medidores ou gráficos para que os operadores possam ver a pressão de tráfego em tempo real.

Property	Type	Descrição
TotalRequests	Int64	Número total de requisições avaliadas desde o último Reset.
TotalThrottled	Int64	Número total de requisições rejeitadas pela estratégia ou pela proteção contra bursts.
TotalQuotaExceeded	Int64	Número total de requisições rejeitadas porque uma quota foi atingida.
ActiveKeys	Inte- ger	Número de chaves atualmente rastreadas (buckets + contadores de janela + conexões ativas).
Uptime	Int64	Segundos desde que o objeto de estatísticas foi resetado pela última vez.

```
// Poll from a TTimer and update a dashboard
void __fastcall TForm1::StatsTimer(TObject *Sender)
{
    LabelRequests->Caption = "Requests: " + IntToStr(sgcWSRateLimiter1->Stats->TotalRequests);
    LabelThrottled->Caption = "Throttled: " + IntToStr(sgcWSRateLimiter1->Stats->TotalThrottled);
    LabelActiveKeys->Caption = "Active keys: " + IntToStr(sgcWSRateLimiter1->Stats->ActiveKeys);
}
```

```
LabelUptime->Caption = "Uptime: " + IntToStr(sgcWSRateLimiter1->Stats->Uptime) + " s";
}
```

Eventos

OnThrottled: Disparado quando uma requisição é rejeitada pela estratégia ou pela proteção de burst. Fornece a chave (IP, chave de API ou usuário) e a string de motivo ("token_bucket", "sliding_window", "fixed_window", "burst", "connection_rate" ou "message_rate"). O parâmetro *Allow* (var Boolean) permite que você sobreponha a decisão e deixe a requisição passar mesmo assim.

OnQuotaExceeded: Disparado quando uma cota de longo prazo é atingida. Fornece a chave e o Nome da cota que foi excedida.

OnStateChange: Disparado quando o tamanho do Token Bucket muda para uma chave. Fornece a chave junto com os tamanhos de bucket antigo e novo. Útil para telemetria, depuração e visualizações em tempo real.

Métodos Públicos

Method	Descrição
IsAllowed(Key, Cost)	Retorna True se uma requisição do Cost fornecido (padrão 1) for permitida para a chave especificada.
Consume(Key, Cost)	Tenta consumir Cost tokens e retorna um TsgcRateLimitResult (Allowed, Remaining, RetryAfterSec, Reason).
Reset(Key)	Limpa os contadores da chave especificada.
ResetAll	Limpa todos os contadores internos de cada chave rastreada e reseta o objeto Stats.
GetRemaining(Key)	Retorna o número de requisições ainda disponíveis para a chave especificada.
GetRetryAfter(Key)	Retorna o número de segundos que o chamador deve aguardar antes de tentar novamente.
SaveStateToFile(FileName)	Persiste todos os buckets e contadores no arquivo fornecido.
LoadStateFromFile(FileName)	Restaura buckets e contadores previamente salvos em um arquivo.
SaveStateToStream(Stream)	Persiste o estado interno em um stream arbitrário.
LoadStateFromStream(Stream)	Restaura o estado interno a partir de um stream arbitrário.
IsConnectionAllowed(IP)	Hook de integração de servidor. Retorna True se uma nova conexão de IP passar pelo rate limiting. Chamado automaticamente pelo servidor.

IsMessageAllowed(IP, Message)	Hook de integração com o servidor. Retorna True se uma mensagem do IP passar pelo rate limiting. Chamado automaticamente pelo servidor.
RegisterConnection(IP)	Registra uma nova conexão para rastreamento. Chamado automaticamente.
UnregisterConnection(IP)	Remove uma conexão do rastreamento. Chamado automaticamente na desconexão.

Integração Automática

Quando atribuído à propriedade RateLimiter de um servidor, o componente automaticamente:

- **Em nova conexão:** Chama IsConnectionAllowed para avaliar PerIP, proteção contra burst e a estratégia global. Rejeita a conexão com HTTP 429 antes do evento OnConnect se o limite for excedido. Chama RegisterConnection para rastrear o cliente.
- **Ao receber mensagem:** Chama IsMessageAllowed para verificar PerIP, PerEndpoint, PerAPIKey, PerUser, quotas e proteção de burst. Desconecta ou rejeita a mensagem quando o limite é excedido.
- **Ao desconectar:** Chama UnregisterConnection para liberar os recursos de rastreamento.

Exemplo Completo

```
// Enable rate limiter
sgcWSRateLimiter1->Enabled = true;

// Global Token Bucket: 50 burst, 10/sec sustained
sgcWSRateLimiter1->TokenBucket->Enabled = true;
sgcWSRateLimiter1->TokenBucket->Capacity = 50;
sgcWSRateLimiter1->TokenBucket->RefillRate = 10;
sgcWSRateLimiter1->TokenBucket->RefillIntervalMs = 1000;

// Per-IP sliding window: 60 requests per minute per IP
sgcWSRateLimiter1->PerIP->Enabled = true;
sgcWSRateLimiter1->PerIP->Strategy = rlsSliding;
sgcWSRateLimiter1->PerIP->MaxRequests = 60;
sgcWSRateLimiter1->PerIP->WindowSec = 60;

// Per-endpoint: expensive report endpoint = 10 req/min
sgcWSRateLimiter1->PerEndpoint->Enabled = true;
TsgcRateLimitRuleItem *rule = (TsgcRateLimitRuleItem*)
    sgcWSRateLimiter1->PerEndpoint->Rules->Add();
rule->Name = "expensive-report";
rule->Pattern = "*/api/v1/expensive-report*";
rule->Strategy = rlsSliding;
rule->MaxRequests = 10;
rule->WindowSec = 60;

// Burst protection: 50 requests in 500ms triggers 30s cooldown
sgcWSRateLimiter1->BurstProtection->Enabled = true;
sgcWSRateLimiter1->BurstProtection->BurstThreshold = 50;
sgcWSRateLimiter1->BurstProtection->BurstWindowMs = 500;
sgcWSRateLimiter1->BurstProtection->CooldownSec = 30;

// Monthly quota per API key: 10000 requests/month
sgcWSRateLimiter1->Quotas->Enabled = true;
TsgcRateLimitQuotaItem *quota = (TsgcRateLimitQuotaItem*)
    sgcWSRateLimiter1->Quotas->Quotas->Add();
quota->Name = "free-tier-monthly";
quota->Scope = qscPerKey;
quota->Period = qpMonth;
quota->Limit = 10000;

// HTTP 429 response with Retry-After and X-RateLimit-* headers
sgcWSRateLimiter1->Response->StatusCode = 429;
```



```

sgcWSRateLimiter1->Response->Message = "Too Many Requests";
sgcWSRateLimiter1->Response->RetryAfterHeader = true;
sgcWSRateLimiter1->Response->IncludeRateLimitHeaders = true;

// Persist state across restarts
sgcWSRateLimiter1->StorageFile = "ratelimit.dat";
sgcWSRateLimiter1->LoadStateFromFile("ratelimit.dat");

// Assign to server
sgcWebSocketHTTPServer1->RateLimiter = sgcWSRateLimiter1;

// Handle throttled events
void __fastcall TForm1::RateLimiterThrottled(TObject *Sender,
    const String aKey, const String aReason, bool &Allow)
{
    Log("Throttled " + aKey + ": " + aReason);
}

// Handle quota exhausted events
void __fastcall TForm1::RateLimiterQuotaExceeded(TObject *Sender,
    const String aKey, const String aQuotaName)
{
    Log("Quota " + aQuotaName + " exceeded for " + aKey);
}

```

Segurança de Threads

O rate limiter é totalmente thread-safe. Todos os métodos públicos utilizam seções críticas internas e contadores thread-safe para proteger o acesso concorrente. O componente pode ser compartilhado por múltiplas instâncias de servidor e acessado a partir de qualquer thread (handlers de eventos do servidor, threads de timer, workers em segundo plano) sem sincronização externa.

Referência

- Propriedades — 13 propriedades disponíveis no componente.
- Methods — 12 métodos públicos.
- Eventos — 3 eventos que você pode tratar.

TsgcWSCircuitBreaker

TsgcWSCircuitBreaker implementa o padrão de resiliência Circuit Breaker para chamadas a APIs HTTP no lado do cliente.

Introdução

TsgcWSCircuitBreaker implementa o padrão de resiliência Circuit Breaker para chamadas do lado do cliente a APIs HTTP. Ele previne falhas em cascata quando um endpoint remoto fica indisponível, rastreando falhas em uma janela deslizante, abrindo o circuito quando limiares configuráveis são atingidos e rejeitando chamadas imediatamente durante o período de cooldown. Suporta limiares de contagem de falhas e de taxa de falhas, detecção de chamadas lentas, chamadas de teste em estado half-open, classificação de exceções (registrar / ignorar), substituições por endpoint, respostas de fallback e métricas em tempo real.

O circuit breaker integra-se automaticamente a qualquer descendente de **TsgcHTTPAPI_client** (OpenAI, Anthropic, Gemini, Grok, DeepSeek, Mistral, Ollama, Stripe, Binance, Bybit, Coinbase, Kraken e os outros wrappers de API) através da propriedade **CircuitBreaker**. Uma vez atribuído, cada chamada Get / Post / Put / Patch / Delete é encapsulada automaticamente. Um circuito independente é mantido por nome de host único.

Início rápido

1. Solte um componente **TsgcWSCircuitBreaker** no formulário.

2. Configure Thresholds (quando abrir) e Recovery (como recuperar):

```
sgcWSCircuitBreaker1.Thresholds.FailureCount := 5;  
sgcWSCircuitBreaker1.Recovery.CooldownSec := 30;
```

3. Atribua o circuit breaker a um cliente HTTP API (qualquer descendente de TsgcHTTPAPI_client):

```
sgcAI_OpenAI1.CircuitBreaker := sgcWSCircuitBreaker1;
```

4. Chame Get / Post / Delete / Patch / Put no cliente. O breaker registra sucessos e falhas automaticamente e rejeita novas chamadas enquanto o circuito está Open.

States

O que faz: O circuit breaker é uma máquina de três estados. Cada host (chave) tem seu próprio estado independente que controla se as chamadas de saída são permitidas, rejeitadas ou enviadas como tentativas.

State	Comportamento	Transition
csClosed	Todas as chamadas passam. As falhas são contadas na janela móvel.	Move para csOpen quando FailureCount, FailureRatePercent ou SlowCallRatePercent é atingido.
csOpen	Toda chamada é rejeitada imediatamente. OnCallRejected é disparado para cada tentativa.	Move para csHalfOpen após decorridos Recovery.CooldownSec segundos desde a abertura.

csHalfOpen	Até Recovery.HalfOpenTrialCalls chamadas de teste são permitidas para sondar a recuperação.	Move para csClosed quando as tentativas têm sucesso; volta para csOpen se qualquer tentativa falhar.

Propriedades de nível superior

O que faz: Interruptor mestre do componente mais os dois nomes de chave usados quando nenhuma chave por chamada é fornecida. A integração do lado do cliente usa o nome de host de destino como a chave automaticamente — DefaultKey e ServerKey só se aplicam a chamadas manuais Execute/RecordSuccess/RecordFailure e aos helpers opcionais de integração com servidor.

Quando utilizar: Desabilite o breaker em tempo de execução durante um teste de carga controlado ou durante uma janela de manutenção sem desvinculá-lo do cliente HTTP. Defina um DefaultKey significativo (por exemplo 'openai-api') quando você acionar Execute manualmente dentro de código personalizado.

Pro- perty	Type	De- fault	Descrição
Enabled	Boole- an	True	Interruptor principal. Quando False, cada método público faz um curto-circuito para "permitido" e nenhum contador é atualizado.
Default- Key	string	'de- fault'	Chave utilizada pelas sobrecargas sem parâmetros de Execute, RecordSuccess e RecordFailure.
Server- Key	string	'ser- ver'	Chave utilizada pelos hooks opcionais de integração com o servidor (IsConnectionAllowed, IsMessageAllowed, RecordMessageError).

```
sgcWSCircuitBreaker1->Enabled = true;
sgcWSCircuitBreaker1->DefaultKey = "openai-api";
sgcWSCircuitBreaker1->ServerKey = "server";
```

Thresholds

O que faz: Define as condições que movem o circuito de Closed para Open. O breaker abre quando FailureCount falhas brutas são atingidas, ou quando FailureRatePercent das chamadas na janela falham (após MinCalls), ou quando SlowCallRatePercent das chamadas excede SlowCallDurationMs.

Quando utilizá-lo: Seu servidor chama a API da OpenAI. Quando ocorrem 5 falhas dentro de 60 segundos (ou 50% das chamadas falham, uma vez que pelo menos 10 chamadas tenham sido observadas), abra o circuito e pare de sobrecarregar a OpenAI — deixe o serviço remoto se recuperar em vez de acumular novas tentativas.

Property	Type	De- fault	Descrição

Enabled	Boolean	True	Habilita a abertura baseada em limiar. Quando False, o circuito nunca abre automaticamente.
FailureCount	Integer	5	Número absoluto de falhas na janela deslizante que aciona o disjuntor (breaker).
FailureRatePercent	Integer	50	Porcentagem de falhas (0-100) que aciona o breaker quando MinCalls é atingido.
SlowCallDurationMs	Integer	3000	Limiar (ms) acima do qual uma chamada bem-sucedida também é contabilizada como "lenta".
SlowCallRatePercent	Integer	0	Percentual (0-100) de chamadas lentas que aciona o breaker. 0 desabilita a detecção de chamadas lentas.
MinCalls	Integer	10	Mínimo de chamadas observadas antes de os thresholds de taxa serem avaliados (evita acionamentos em amostras pequenas).

```
// Open after 5 failures, or 50% failure rate on at least 10 calls
sgcWSCircuitBreaker1->Thresholds->Enabled = true;
sgcWSCircuitBreaker1->Thresholds->FailureCount = 5;
sgcWSCircuitBreaker1->Thresholds->FailureRatePercent = 50;
sgcWSCircuitBreaker1->Thresholds->MinCalls = 10;
// Also treat any call slower than 3 seconds as a "slow" call
sgcWSCircuitBreaker1->Thresholds->SlowCallDurationMs = 3000;
sgcWSCircuitBreaker1->Thresholds->SlowCallRatePercent = 60;
```

TimeWindow

O que faz: Define a janela deslizante (rolling-window) utilizada para a avaliação do threshold. A janela é dividida em BucketCount sub-buckets, de modo que contadores antigos expirem suavemente conforme o tempo avança. Os thresholds são sempre avaliados em relação à soma dos buckets ativos.

Quando utilizá-lo: Utilize uma janela móvel de 30 segundos com 6 buckets para que picos transitórios não acionem o circuito, mas falhas sustentadas o façam. Janelas mais longas suavizam o ruído; janelas mais curtas reagem mais rápido a interrupções. Mais buckets dão um envelhecimento mais suave, mas consomem um pouco mais de memória por host.

Property	Type	Default	Descrição
RollingWindowSec	Integer	60	Largura total da janela deslizante em segundos. Amostras mais antigas são descartadas.
BucketCount	Integer	10	Número de sub-buckets em que a janela é dividida. Cada bucket mantém contadores de sucesso / falha / lentidão.

```
// 30-second window split into 6 buckets (5 seconds each)
sgcWSCircuitBreaker1->TimeWindow->RollingWindowSec = 30;
```

```
sgcWSCircuitBreaker1->TimeWindow->BucketCount = 6;
```

Recuperação

O que faz: Controla como o circuito se recupera do estado Open. Após decorrer CooldownSec, o breaker transita para HalfOpen e permite a passagem de HalfOpenTrialCalls chamadas de teste. Se os testes forem bem-sucedidos, o breaker fecha; se falharem, ele reabre e o timer de cooldown reinicia.

Quando utilizá-lo: Após abrir, aguarde 30 segundos e então deixe passar uma chamada de teste para testar a recuperação antes de fechar novamente. Um cooldown mais longo protege um upstream em dificuldades de outra onda de tráfego; um cooldown curto reabre rapidamente para serviços que apenas oscilam.

Property	Type	De-fault	Descrição
CooldownSec	Integer	30	Segundos a aguardar no estado Open antes de transicionar para HalfOpen.
HalfOpenTrialCalls	Integer	1	Número de chamadas de teste permitidas em HalfOpen antes de decidir fechar ou reabrir.
AutoReset	Boolean	True	Quando True, os contadores são zerados automaticamente nas transições de estado.
MaxRetries	Integer	3	Teto informativo consumido pelos helpers de retry. Nenhum retry automático é feito pelo próprio breaker.

```
// After opening: wait 30s, then allow 1 trial call, then decide
sgcWSCircuitBreaker1->Recovery->CooldownSec = 30;
sgcWSCircuitBreaker1->Recovery->HalfOpenTrialCalls = 1;
sgcWSCircuitBreaker1->Recovery->AutoReset = true;
sgcWSCircuitBreaker1->Recovery->MaxRetries = 3;
```

Fallback

O que faz: Configura um payload alternativo servido enquanto o circuito está Open. Quando Fallback.Enabled é True e Execute / ExecuteWithResult rejeita uma chamada, o evento OnFallback dispara com uma resposta inicializada a partir de CachedResponse (ou CustomMessage). Os manipuladores podem substituir esta string antes de ela ser retornada ao chamador.

Quando utilizá-lo: Retornar uma resposta em cache do último estado bom conhecido quando a OpenAI está fora do ar, para que os usuários vejam um comportamento degradado, mas funcional (por exemplo, o resumo gerado anteriormente) em vez de um erro. Utilize CustomMessage para um corpo JSON estático de "modo de manutenção".

Property	Type	De-fault	Descrição
Enabled	Boolean	False	Habilita o caminho de fallback. Quando False, as chamadas rejeitadas simplesmente retornam sem disparar OnFallback.
Cache-Response	string	"	Payload estático retornado enquanto o circuito está Open. Tem precedência sobre CustomMessage.
Custom-Message	string	"	Payload de fallback usado quando CachedResponse está vazio.
UseLast-Success	Boolean	False	Quando True, a última resposta bem-sucedida observada para a chave é servida como fallback.

```
// When api.openai.com is down, serve a safe JSON body instead of erroring
sgcWSCircuitBreaker1->Fallback->Enabled = true;
sgcWSCircuitBreaker1->Fallback->CachedResponse =
    "{\"error\":\"service unavailable\",\"retry_after\":30}";
sgcWSCircuitBreaker1->Fallback->CustomMessage =
    "{\"error\":\"ai service temporarily offline\"}";
sgcWSCircuitBreaker1->Fallback->UseLastSuccess = true;
```

Classificação

O que faz: Classifica as exceções geradas como "registradas como falha" ou "ignoradas". Apenas exceções registradas avançam os contadores de falha. MatchMode seleciona como as strings em RecordAsFailure / IgnoreExceptions são comparadas ao texto da exceção (ClassName: Message).

Quando utilizá-lo: Trate `EIdHTTPProtocolException` com 500/502/503/504 como falhas, mas ignore `EIdHTTPProtocolException` 400/401/404 (erros de negócio não devem acionar o circuito). Deixar `RecordAsFailure` vazio significa que toda exceção não ignorada é registrada.

Property	Type	De-fault	Descrição
RecordAs-Failure	TStringList	vazio	Padrões comparados com o texto da exceção; exceções correspondentes contam como falhas. Lista vazia = registrar tudo que não for ignorado.
IgnoreExceptions	TStringList	vazio	Padrões comparados com o texto da exceção; exceções correspondentes são descartadas (nem sucesso nem falha).
MatchMode	TsgcCircuitBreakerExceptionMatch	cem-Contains	Modo de comparação: <code>cemExact</code> , <code>cemContains</code> , <code>cemStartsWith</code> ou <code>cemWildcard</code> (* / ?).

```
// Record HTTP 5xx and timeouts as failures; ignore client-side 4xx
sgcWSCircuitBreaker1->Classification->MatchMode = cemContains;
sgcWSCircuitBreaker1->Classification->RecordAsFailure->Add("HTTP/1.1 500");
sgcWSCircuitBreaker1->Classification->RecordAsFailure->Add("HTTP/1.1 502");
sgcWSCircuitBreaker1->Classification->RecordAsFailure->Add("HTTP/1.1 503");
sgcWSCircuitBreaker1->Classification->RecordAsFailure->Add("HTTP/1.1 504");
sgcWSCircuitBreaker1->Classification->RecordAsFailure->Add("timeout");
sgcWSCircuitBreaker1->Classification->IgnoreExceptions->Add("HTTP/1.1 400");
sgcWSCircuitBreaker1->Classification->IgnoreExceptions->Add("HTTP/1.1 401");
sgcWSCircuitBreaker1->Classification->IgnoreExceptions->Add("HTTP/1.1 404");
```

PerEndpoint

O que faz: Coleção de substituições baseadas em padrões. Cada `TsgcCircuitBreakerEndpointItem` tem seu próprio `Pattern` (curingas `*` e `?` suportados), um bloco `OverrideThresholds` e um bloco `OverrideRecovery`. Quando a chave ativa (`hostname`) corresponde a um item habilitado, as configurações desse item substituem os `Thresholds` / `Recovery` de nível superior durante a chamada.

Quando usá-lo: Circuitos separados por API: `api.openai.com` tem limiares brandos (ocasionalmente fica lento sob carga), mas `api.stripe.com` é crítico para pagamentos e deve abrir mais rápido para falhar de forma fechada (fail closed). Adicione um item de endpoint por host com limiares ajustados.

Property	Type	Default	Descrição
Enabled	Boolean	False	Habilita a coleção de substituições de endpoint. Quando False, nenhum item é avaliado.
Items	TCollection	vazio	Coleção de <code>TsgcCircuitBreakerEndpointItem</code> . A primeira correspondência habilitada vence.

Cada `TsgcCircuitBreakerEndpointItem` expõe:

Property	Type	Default	Descrição
Pattern	string	"	Padrão com wildcard comparado à chave (<code>hostname</code>). Suporta <code>*</code> e <code>?</code> .
Enabled	Boolean	True	Desabilita este item sem excluí-lo.
OverrideThresholds	TsgcCircuitBreakerThresholds	defaults	Thresholds aplicados quando o padrão corresponde. Ativos apenas quando seu próprio <code>Enabled</code> é True.
OverrideRecovery	TsgcCircuitBreakerRecovery	defaults	Configurações de recovery aplicadas quando o padrão corresponde.

```
sgcWSCircuitBreaker1->PerEndpoint->Enabled = true;
```

```
// OpenAI - lenient: 10 failures, 60s cooldown
TsgcCircuitBreakerEndpointItem *oOpenAI = (TsgcCircuitBreakerEndpointItem*)
sgcWSCircuitBreaker1->PerEndpoint->Items->Add();
oOpenAI->Pattern = "api.openai.com";
oOpenAI->Enabled = true;
oOpenAI->OverrideThresholds->Enabled = true;
oOpenAI->OverrideThresholds->FailureCount = 10;
oOpenAI->OverrideThresholds->FailureRatePercent = 70;
oOpenAI->OverrideRecovery->CooldownSec = 60;

// Stripe - payment critical: open fast, recover slow
TsgcCircuitBreakerEndpointItem *oStripe = (TsgcCircuitBreakerEndpointItem*)
sgcWSCircuitBreaker1->PerEndpoint->Items->Add();
oStripe->Pattern = "api.stripe.com";
oStripe->Enabled = true;
oStripe->OverrideThresholds->Enabled = true;
oStripe->OverrideThresholds->FailureCount = 3;
oStripe->OverrideThresholds->FailureRatePercent = 30;
oStripe->OverrideRecovery->CooldownSec = 120;
```

Metrics

O que ele faz: Contadores somente leitura atualizados atonicamente conforme as chamadas fluem pelo breaker. Agregados em todas as chaves / hostnames. CurrentOpenBreakers é atualizado após cada transição de estado; AverageLatencyMs é uma média móvel das durações observadas por Execute.

Quando utilizá-lo: Dashboard ao vivo mostrando os estados de circuito, taxas de falha e contagens de rejeição por host. Faça polling de Metrics a partir de um TTimer para pintar medidores para os operadores, ou emita-os para o Prometheus / StatsD para alertas.

Property	Type	Descrição
TotalCalls	Int64	Total de chamadas processadas (sucessos + falhas registradas).
TotalFailures	Int64	Total de falhas registradas em todas as chaves.
TotalSuccesses	Int64	Total de chamadas bem-sucedidas registradas em todas as chaves.
TotalRejected	Int64	Total de chamadas rejeitadas porque o circuito estava Open.
CurrentOpen-Breakers	Integer	Número de chaves atualmente no estado Open.
AverageLatencyMs	Double	Latência média corrente (ms) medida em torno das chamadas Execute / ExecuteWithResult.

```
// Poll from a TTimer and update a dashboard
void __fastcall TForm1::MetricsTimer(TObject *Sender)
{
    LabelCalls->Caption = "Calls: " +
        IntToStr(sgcWSCircuitBreaker1->Metrics->TotalCalls) + " (ok " +
        IntToStr(sgcWSCircuitBreaker1->Metrics->TotalSuccesses) + " / fail " +
        IntToStr(sgcWSCircuitBreaker1->Metrics->TotalFailures) + " / rejected " +
        IntToStr(sgcWSCircuitBreaker1->Metrics->TotalRejected) + ")";
    LabelOpen->Caption = "Open breakers: " +
        IntToStr(sgcWSCircuitBreaker1->Metrics->CurrentOpenBreakers);
    LabelLatency->Caption = "Avg latency: " +
        FormatFloat("0.0", sgcWSCircuitBreaker1->Metrics->AverageLatencyMs) + " ms";
}
```


Eventos

OnStateChange: Disparado quando um circuito transita entre estados. Fornece a chave, o estado antigo e o novo estado (csClosed, csOpen ou csHalfOpen). Útil para logging, alertas e atualização de interface ao vivo.

OnCallRejected: Disparado quando uma chamada é rejeitada porque o circuito está Open ou o orçamento de tentativas HalfOpen se esgotou. Fornece a chave e uma string de motivo (por exemplo, `Circuit open`).

OnFallback: Disparado logo antes de uma resposta de fallback ser retornada. O parâmetro *aResponse* (var string) é inicializado a partir de `Fallback.CachedResponse / CustomMessage` e pode ser substituído pelo handler.

OnFailureRecorded: Disparado toda vez que uma falha é registrada. Fornece a chave e o texto da exceção (`ClassName: Message`). Utilize para logging estruturado ou para correlacionar com o rastreamento de incidentes upstream.

OnSlowCall: Disparado quando uma chamada bem-sucedida excedeu `Thresholds.SlowCallDurationMs`. Fornece a chave e a duração medida em milissegundos.

Métodos Públicos

Method	Descrição
<code>Execute(Key, Action)</code>	Executa o TProc sob a chave fornecida. Verifica <code>IsCallAllowed</code> primeiro, registra sucesso / falha / lentidão automaticamente e retorna <code>True</code> se a ação foi executada.
<code>Execute(Action)</code>	Sobrecarga usando <code>DefaultKey</code> .
<code>ExecuteWithResult(Key, Action, out Result)</code>	(D2009+) Executa um <code>TFunc<TObject></code> e retorna seu resultado por meio do parâmetro <code>out</code> ; mesma contabilização que <code>Execute</code> .
<code>RecordSuccess(Key)</code>	Registra uma chamada bem-sucedida. Avança de <code>HalfOpen</code> -> <code>Closed</code> quando as tentativas têm sucesso.
<code>RecordSuccess</code>	Sobrecarga usando <code>DefaultKey</code> .
<code>RecordFailure(Key, Exception)</code>	Registra uma falha com texto de exceção opcional; aciona a avaliação de transição.
<code>RecordFailure(Exception)</code>	Sobrecarga usando <code>DefaultKey</code> .
<code>ForceOpen(Key)</code>	Força manualmente o circuito para o estado <code>Open</code> .
<code>ForceClose(Key)</code>	Força manualmente o circuito para o estado <code>Closed</code> .
<code>Reset(Key)</code>	Limpa os contadores e o estado para a chave fornecida.
<code>ResetAll</code>	Limpa todos os circuitos rastreados e reinicia o <code>Metrics</code> .
<code>GetState(Key)</code>	Retorna o <code>TsgcCircuitState</code> atual para a chave.

IsCallAllowed(Key)	Retorna True se uma nova chamada é atualmente permitida. Chamado automaticamente pelo cliente HTTP API antes de cada requisição.
GetFailureRate(Key)	Retorna o percentual de falhas atual (0.0 - 100.0) observado na janela móvel.
GetTotalCalls(Key)	Retorna o número de chamadas observadas na janela deslizante para a chave.
GetLastStateChange(Key)	Retorna o TDateTime da última transição de estado.
GetOpenBreakers	Retorna um TArray<string> das chaves atualmente no estado Open.
SaveStateToFile(FileName)	Persiste o estado e os contadores de cada circuito em um arquivo.
LoadStateFromFile(FileName)	Restaura os estados de circuito a partir de um arquivo salvo anteriormente.
IsConnectionAllowed(IP)	Hook de autoproteção no lado do servidor. Secundário em relação ao uso principal no lado do cliente. Retorna True se o circuito ServerKey não estiver Open.
IsMessageAllowed(IP, Message)	Hook de autoproteção do lado do servidor. Retorna True se o circuito ServerKey não está Open.
RegisterConnection(IP)	Hook do lado do servidor reservado para métricas futuras por IP (atualmente sem operação).
UnregisterConnection(IP)	Hook do lado do servidor reservado para métricas futuras por IP (atualmente sem operação).
RecordMessageError(IP, Exception)	Hook de autoproteção do lado do servidor. Registra uma falha em ServerKey.
RecordMessageSuccess(IP)	Hook de autoproteção do lado do servidor. Registra um sucesso em ServerKey.

Integração Automática

Quando atribuído a um descendente de TsgcHTTPAPI_client através da propriedade CircuitBreaker, o cliente automaticamente:

- **Antes de cada Get / Post / Put / Patch / Delete:** Deriva a chave a partir do hostname da URL de destino (por exemplo `api.openai.com`) e chama `IsCallAllowed(host)`. Se o circuito estiver Open, a chamada é rejeitada com a exceção *"Circuit breaker open for <host> - call rejected"* e `OnCallRejected` dispara.
- **Em resposta HTTP bem-sucedida:** Chama `RecordSuccess(host)` para atualizar os contadores, permitindo que os testes HalfOpen fechem o circuito quando o remoto se recupera.
- **Em exceção:** Chama `RecordFailure(host, exception.Message)`, aplica as regras de Classificação para decidir se conta ou ignora, avalia as transições e então relança a exceção ao chamador.
- **Um circuito por host:** Chamadas a hosts diferentes (por exemplo `api.openai.com`, `api.anthropic.com`, `api.stripe.com`) rastreiam circuitos totalmente independentes.

Exemplo Completo

```
// Enable the circuit breaker
sgcWSCircuitBreaker1->Enabled = true;

// Thresholds: open after 5 failures, or 50% failure rate on 10+ calls
sgcWSCircuitBreaker1->Thresholds->Enabled = true;
sgcWSCircuitBreaker1->Thresholds->FailureCount = 5;
sgcWSCircuitBreaker1->Thresholds->FailureRatePercent = 50;
sgcWSCircuitBreaker1->Thresholds->SlowCallDurationMs = 3000;
sgcWSCircuitBreaker1->Thresholds->MinCalls = 10;

// Rolling window: 60 seconds / 10 buckets
sgcWSCircuitBreaker1->TimeWindow->RollingWindowSec = 60;
sgcWSCircuitBreaker1->TimeWindow->BucketCount = 10;

// Recovery: wait 30s in Open, then 1 trial call in HalfOpen
sgcWSCircuitBreaker1->Recovery->CooldownSec = 30;
sgcWSCircuitBreaker1->Recovery->HalfOpenTrialCalls = 1;

// Classification: record HTTP 5xx and timeouts, ignore 4xx
sgcWSCircuitBreaker1->Classification->MatchMode = cemContains;
sgcWSCircuitBreaker1->Classification->RecordAsFailure->Add("HTTP/1.1 5");
sgcWSCircuitBreaker1->Classification->RecordAsFailure->Add("timeout");
sgcWSCircuitBreaker1->Classification->IgnoreExceptions->Add("HTTP/1.1 404");

// Fallback: serve a cached payload while the circuit is Open
sgcWSCircuitBreaker1->Fallback->Enabled = true;
sgcWSCircuitBreaker1->Fallback->CustomMessage =
    "{ \"error\": \"service unavailable\" }";

// Per-endpoint overrides
sgcWSCircuitBreaker1->PerEndpoint->Enabled = true;
TsgcCircuitBreakerEndpointItem *oStripe = (TsgcCircuitBreakerEndpointItem*)
    sgcWSCircuitBreaker1->PerEndpoint->Items->Add();
oStripe->Pattern = "api.stripe.com";
oStripe->OverrideThresholds->Enabled = true;
oStripe->OverrideThresholds->FailureCount = 3;
oStripe->OverrideRecovery->CooldownSec = 120;

// Persist circuit state across restarts
sgcWSCircuitBreaker1->LoadStateFromFile("circuit.dat");

// Attach to an HTTP API client (any TsgcHTTAPI_client descendant)
sgcAI_OpenAPI1->CircuitBreaker = sgcWSCircuitBreaker1;

// Use the client normally; the breaker intercepts Get / Post / etc.
try
{
    String Response = sgcAI_OpenAPI1->Get("https://api.openai.com/v1/models");
}
catch (Exception &E)
{
    if (E.Message.Pos("Circuit breaker open") > 0)
        Log("Circuit open for api.openai.com - using fallback");
    else
        Log("Call failed: " + E.Message);
}

// Handle state changes
void __fastcall TForm1::CircuitStateChange(TObject *Sender,
    const String aKey, TsgcCircuitState aOldState, TsgcCircuitState aNewState)
{
    Log("Circuit " + aKey + ": " + IntToStr((int)aOldState) + " -> " +
        IntToStr((int)aNewState));
}

// Handle fallback responses
void __fastcall TForm1::CircuitFallback(TObject *Sender,
    const String aKey, String &aResponse)
{
    Log("Fallback used for " + aKey);
}
```

Segurança de Threads

O circuit breaker é totalmente thread-safe. Todos os métodos públicos usam seções críticas internas e listas thread-safe para proteger o acesso concorrente ao estado por chave, à janela rolante e aos contadores de métricas. Um único componente pode ser compartilhado por múltiplos clientes de HTTP API e acessado a partir de qualquer thread (manipuladores de eventos, timers, workers em segundo plano) sem sincronização externa.

Referência

- Propriedades — 10 propriedades disponíveis no componente.
- Métodos — 17 métodos públicos.
- Eventos — 5 eventos que você pode tratar.

TsgcWSAPIKeyManager

O *TsgcWSAPIKeyManager* fornece o gerenciamento de ciclo de vida completo para API keys emitidas pelos servidores *sgcWebSockets*.

Introdução

O *TsgcWSAPIKeyManager* fornece gerenciamento de ciclo de vida completo para API keys emitidas por servidores *sgcWebSockets*. Ele gera chaves criptograficamente aleatórias com um prefixo configurável legível por humanos, faz hash delas em repouso com SHA-256, SHA-512 ou Bcrypt, valida requisições de entrada por cabeçalho ou query string, rotaciona chaves com um período de carência configurável, revoga e renova chaves, aplica autorização baseada em scope, mantém um log de auditoria persistente e suporta uma IP allowlist opcional. As chaves são armazenadas em memória ou em um arquivo criptografado que sobrevive a reinicializações.

O gerenciador se conecta ao *TsgcWebSocketHTTPServer* e ao *TsgcWSServer_HTTPAPI* através da propriedade **APIKeyManager**. Como os hooks automáticos do servidor só veem o IP do peer, a autenticação real da requisição é realizada chamando **IsRequestAuthorized** a partir do evento *OnConnect* do servidor — isso dá acesso ao bloco completo de cabeçalhos HTTP e à URL/query string, de modo que o cabeçalho **X-API-Key** configurado (ou parâmetro de query) possa ser parseado, validado e, se solicitado, comparado a um escopo obrigatório.

Início rápido

1. Coloque um componente **TsgcWSAPIKeyManager** no formulário.
2. Configure Generation (prefix, length), Hashing (algorithm, salt) e Validation (header name, IP allowlist, FailClosed). Veja a documentação por seção abaixo.
3. Atribua o manager a um componente de servidor:

```
sgcWebSocketHTTPServer1.APIKeyManager := sgcWSAPIKeyManager1;
```

4. Conecte o *OnConnect* do servidor para chamar **IsRequestAuthorized** usando os cabeçalhos, a URL e o IP da conexão. Desconecte o cliente quando ele retornar *False*.
5. Emita a primeira chave para um cliente e entregue a ele o valor em texto puro exatamente uma vez, somente a forma com hash permanece armazenada depois disso.

Propriedades de nível superior

O que faz: Chave geral (master switch) de todo o componente. Quando *Enabled* é *False*, cada método público encurta para "permitido" e os auto-hooks do servidor tornam-se inoperantes (no-ops), então o gerenciador de chaves é efetivamente ignorado.

Quando usá-lo: Desabilitar temporariamente a imposição de chaves durante uma janela de manutenção controlada ou uma implantação de staging sem desvincular o componente do servidor. Reabilite-o quando a alteração estiver concluída — nenhuma chave é perdida, nenhum escopo muda e o log de auditoria continua funcionando no momento em que você o reativa.

Pro- perty	Type	De- fault	Descrição

Enabled	Boolean	True	Chave geral (master switch). Quando False, ValidateKey / IsRequestAuthorized sempre retornam True e os auto-hooks do servidor tornam-se inoperantes (no-ops).

```
sgcWSAPIKeyManager1->Enabled = true;
```

Generation

O que faz: Controla como a string da chave em texto simples é construída por *IssueKey*. A chave é um *KeyPrefix* seguido por *KeyLength* caracteres aleatórios extraídos de *Charset*. Quando *IncludeChecksum* é True, um checksum curto é anexado, de modo que chaves malformadas possam ser rejeitadas de forma econômica antes de uma busca por hash.

Quando utilizá-lo: Chaves com marca para o seu SaaS — utilize `sgc_live_aB3...` em produção e `sgc_test_...` no cluster de homologação. O prefixo torna trivial para a equipe de suporte identificar a qual ambiente pertence uma chave vazada e para os redatores de log censurá-la.

Property	Type	Default	Descrição
Enabled	Boolean	True	Quando False, <i>IssueKey</i> gera uma exceção em vez de gerar uma nova chave.
KeyPrefix	string	"	Prefixo legível por humanos adicionado ao corpo aleatório (por exemplo <code>sgc_live_</code>).
KeyLength	Integer	32	Número de caracteres aleatórios no corpo da chave.
Charset	string	A-Za-z0-9	Alfabeto usado para gerar o corpo aleatório. O padrão são caracteres alfanuméricos URL-safe quando vazio.
IncludeChecksum	Boolean	False	Quando True, anexa um sufixo de checksum curto para que chaves obviamente malformadas possam ser rejeitadas sem um lookup de hash.

```
// Branded production keys: sgc_live_aB3...
sgcWSAPIKeyManager1->Generation->Enabled = true;
sgcWSAPIKeyManager1->Generation->KeyPrefix = "sgc_live_";
sgcWSAPIKeyManager1->Generation->KeyLength = 32;
sgcWSAPIKeyManager1->Generation->IncludeChecksum = true;
```

Storage

O que faz: Controla onde residem as chaves com hash e o log de auditoria. *kstMemory* mantém tudo em processo (limpo no shutdown). *kstFile* persiste em *FileName*, opcionalmente criptografado com *EncryptionKey*, e ressalva a cada *AutoSaveSeconds* segundos. *kstCustom* permite que você implemente seus próprios ganchos *SaveToFile* / *LoadFromFile*.

Quando usar: Mantenha as chaves em memória durante o desenvolvimento para iteração rápida. Mude para *kstFile* com *EncryptAtRest* := *True* para produção, para que as chaves sobrevivam a um reinício do servidor e nunca sejam expostas como texto puro em disco.

Property	Type	Default	Descrição
StorageType	TsgcAPIKeyStorageType	kstMemory	kstMemory (volátil), kstFile (persiste em FileName) ou kstCustom (SaveToFile / LoadFromFile fornecidos pelo usuário).
FileName	string	"	Path do arquivo utilizado por SaveToFile / LoadFromFile quando StorageType é kstFile.
EncryptAtRest	Boolean	False	Quando True, criptografa o key store em disco usando EncryptionKey.
EncryptionKey	string	"	Segredo utilizado para criptografar o arquivo de chave persistido. Deve ser fornecido quando EncryptAtRest é True.
AutoSaveSeconds	Integer	0	Quando > 0, o gerenciador re-salva o armazenamento de chaves em disco a cada AutoSaveSeconds segundos.

```
// Production: encrypted file that survives restarts, auto-saved every 60s
sgcWSAPIKeyManager1->Storage->StorageType = kstFile;
sgcWSAPIKeyManager1->Storage->FileName = "apikeys.dat";
sgcWSAPIKeyManager1->Storage->EncryptAtRest = true;
sgcWSAPIKeyManager1->Storage->EncryptionKey = "my-very-secret-master-key";
sgcWSAPIKeyManager1->Storage->AutoSaveSeconds = 60;
sgcWSAPIKeyManager1->LoadFromFile("apikeys.dat");
```

Hashing

O que faz: Chaves em texto simples nunca são persistidas. *IssueKey* faz o hash da chave gerada utilizando *Algorithm* com um *Salt* opcional e uma contagem de *Iterations*; *ValidateKey* refaz o hash do valor de entrada e realiza uma comparação de tempo constante contra o digest armazenado.

Quando usá-lo: Nunca armazene texto puro. SHA-256 é o padrão sensato para a maioria das implantações. Use SHA-512 para perfis de maior segurança (FIPS 140-2 Level 3). Use Bcrypt quando precisar de key-stretching deliberado, validação mais lenta, mas força bruta offline de um repositório de chaves vazado dramaticamente mais difícil.

Pro-property	Type	Default	Descrição
Algoritmo	TsgcAPI-KeyHashAlgorithm	khaSHA256	Função de hash utilizada para armazenamento em repouso: khaSHA256, khaSHA512 ou khaBcrypt.
Salt	string	"	Salt estático misturado a cada hash. Alterá-lo invalida todas as chaves existentes.
Iterações	Integer	1	Número de rounds de hashing. Aumente para key-stretching ao custo de latência de validação.

```
// SHA-512 with a static salt and 10k iterations for key stretching
sgcWSAPIKeyManager1->Hashing->Algorithm = khaSHA512;
sgcWSAPIKeyManager1->Hashing->Salt = "my-salt";
sgcWSAPIKeyManager1->Hashing->Iterations = 10000;
```

Scopes

O que faz: Catálogo de strings de scope permitidas. Quando *Enabled* é True, cada scope anexado a uma chave emitida deve existir nesta lista; scopes desconhecidos são rejeitados. Entradas individuais do catálogo podem ser desabilitadas sem excluí-las. *ValidateKey* / *IsRequestAuthorized* podem exigir um scope específico por requisição.

Quando utilizá-lo: Permissões granulares. A chave de um cliente carrega apenas `read:orders` e será rejeitada em um endpoint de escrita. Uma integração de parceiro recebe `read:orders + write:shipments`, mas nunca `admin:*`. Cada requisição apresenta um hint de scope e o gerenciador o aplica de forma consistente.

Property	Type	Default	Descrição
Enabled	Boolean	True	Habilita a aplicação de escopo. Quando False, os parâmetros de escopo são ignorados.
Scopes	TCollection	vazio	Coleção de entradas <i>TsgcAPIKeyScopeItem</i> que definem os scopes permitidos.

Cada *TsgcAPIKeyScopeItem* expõe:

Property	Type	Default	Descrição
Name	string	"	Identificador de escopo (por exemplo <code>read:orders</code>). Sensível a maiúsculas e minúsculas.
Descrição	string	"	Descrição legível por humanos exibida no object inspector e nas UIs de administração.

Enabled	Boolean	True	Desabilita este escopo sem excluí-lo. Escopos desabilitados nunca satisfazem HasScope.

```
sgcWSAPIKeyManager1->Scopes->Enabled = true;

TsgcAPIKeyScopeItem *scopeRead = (TsgcAPIKeyScopeItem*)
    sgcWSAPIKeyManager1->Scopes->Scopes->Add();
scopeRead->Name = "read:orders";
scopeRead->Description = "List and read order history";

TsgcAPIKeyScopeItem *scopeWrite = (TsgcAPIKeyScopeItem*)
    sgcWSAPIKeyManager1->Scopes->Scopes->Add();
scopeWrite->Name = "write:shipments";
scopeWrite->Description = "Create and update shipment records";
```

Rotation

O que faz: Torna a rotação de chaves transparente. RotateKey emite uma nova chave para o mesmo proprietário e escopos e marca a antiga como *kksRotated*. Durante a janela de *GracePeriodSec*, ambas as chaves validam, de modo que o cliente pode trocar sem downtime. Quando *AutoRotateDays* é > 0, as chaves mais antigas que esse limiar são rotacionadas pela varredura em segundo plano; *NotifyBeforeExpirySec* dispara OnKeyExpired antecipadamente, para que você possa enviar um e-mail ao cliente.

Quando utilizá-la: Um cliente reporta que sua chave foi vazada, mas sua integração ainda está ativa. Rotacione a chave: ele recebe uma nova, a antiga continua funcionando por 24 horas para que ele possa propagá-la por sua infraestrutura, depois a antiga é invalidada automaticamente.

Property	Type	Default	Descrição
Enabled	Boolean	False	Habilita os recursos de rotação (grace period e varredura de auto-rotação).
GracePeriodSec	Integer	86400	Segundos durante os quais a chave anterior permanece válida após RotateKey. O padrão é 24 horas.
AutoRotateDays	Integer	0	Quando > 0, a varredura em segundo plano rotaciona as chaves mais antigas do que este número de dias.
NotifyBeforeExpirySec	Integer	604800	Dispara OnKeyExpired este número de segundos antes de uma chave realmente expirar. O padrão é 7 dias.

```
// 24h grace period, auto-rotate keys older than 90 days, notify 7 days ahead
sgcWSAPIKeyManager1->Rotation->Enabled = true;
sgcWSAPIKeyManager1->Rotation->GracePeriodSec = 86400;
sgcWSAPIKeyManager1->Rotation->AutoRotateDays = 90;
sgcWSAPIKeyManager1->Rotation->NotifyBeforeExpirySec = 7 * 86400;
```

Expiração

O que ele faz: Controla como as chaves emitidas expiram. *DefaultTTLSec* é aplicado a *IssueKey* quando o chamador passa 0 como *ExpiresInSec*. *EnforceExpiry* alterna se as chaves expiradas são rejeitadas. *CheckIntervalSec* controla com que frequência a varredura em segundo plano busca por entradas expiradas, dispara *OnKeyExpired* e atualiza *Stats*.

Quando utilizar: Chaves de integração de parceiros devem expirar após 90 dias, os parceiros precisam reassinar o acordo e buscar uma nova chave. Definir *DefaultTTLSec* como 90×86400 significa que cada chamada *IssueKey* respeita essa política, a menos que o chamador a sobrescreva explicitamente.

Property	Type	De-fault	Descrição
DefaultTTL-Sec	Integer	0	Time-to-live padrão para chaves recém-emitidas em segundos. 0 significa que nunca expira.
EnforceExpiry	Boolean	True	Quando True, <i>ValidateKey</i> rejeita chaves cujo <i>ExpiresAt</i> está no passado.
CheckIntervalSec	Integer	60	Com que frequência a varredura em segundo plano busca chaves expiradas e dispara <i>OnKeyExpired</i> .

```
// Partner keys expire after 90 days, enforced strictly, swept every minute
sgcWSAPIKeyManager1->Expiration->DefaultTTLSec = 90 * 86400;
sgcWSAPIKeyManager1->Expiration->EnforceExpiry = true;
sgcWSAPIKeyManager1->Expiration->CheckIntervalSec = 60;
```

RateLimit

O que faz: Armazena defaults de rate-limit por chave (máximo de requisições por janela) junto a cada chave. Esses valores são hints informativos que uma instância de *TsgcWSRateLimiter* pode ler por meio do scope *PerAPIKey* para aplicar throttling apropriado ao tier — o próprio gerenciador de API key não faz throttling de tráfego.

Quando usá-lo: SaaS por níveis. Clientes do Free-tier são provisionados a 100 req/hora, do Pro-tier a 10.000 req/hora, do Enterprise a 1.000.000 req/hora. Configure *DefaultMaxRequests* / *DefaultWindowSec* no gerenciador de chaves de API e combine-o com *TsgcWSRateLimiter.PerAPIKey* para que o rate limiter leia o orçamento por chave automaticamente.

Property	Type	Default	Descrição
Enabled	Boolean	False	Habilita metadados de rate-limit por chave junto a cada chave emitida.
DefaultMaxRequests	Integer	100	Máximo padrão de requisições por janela atribuído a novas chaves.

DefaultWindowSec	Integer	60	Tamanho padrão da janela em segundos associado a DefaultMaxRequests.

```
// Default tier: 100 requests per hour per API key
sgcWSAPIKeyManager1->RateLimit->Enabled = true;
sgcWSAPIKeyManager1->RateLimit->DefaultMaxRequests = 100;
sgcWSAPIKeyManager1->RateLimit->DefaultWindowSec = 3600;
// Pair with TsgcWSRateLimiter.PerAPIKey for actual throttling
```

Audit

O que faz: Registra cada ação do ciclo de vida das chaves (emitida, validada, revogada, rotacionada, expirada, escopo concedido/revogado, validação falha) no ring buffer em memória e, quando *LogFile* está definido, anexa as mesmas entradas a um arquivo. *IncludeIP* registra o IP do requisitante; *IncludePayload* captura detalhes adicionais. *RetentionDays* remove entradas mais antigas; *MaxMemoryEntries* limita o ring buffer.

Quando utilizar: Requisito de conformidade: toda operação de chave deve ser registrada e retida por 12 meses. Habilite a auditoria com *LogFile* apontando para a zona de despejo do seu SIEM e *RetentionDays* definido como 365. *OnAuditEvent* é disparado para cada entrada, de modo que você também pode enviá-las para um message bus.

Property	Type	Default	Descrição
Enabled	Boolean	True	Habilita o log de auditoria. Quando False, nenhuma entrada é registrada e <i>OnAuditEvent</i> nunca é disparado.
LogFile	string	"	Caminho de arquivo opcional ao qual o log de auditoria é anexado.
IncludeIP	Boolean	True	Quando True, cada entrada registra o IP do solicitante.
Include-Payload	Boolean	False	Quando True, o campo Details de cada entrada de auditoria carrega contexto adicional.
Retention-Days	Integer	90	Entradas mais antigas do que esta quantidade de dias são removidas pela varredura em segundo plano.
MaxMemoryEntries	Integer	10000	Número máximo de entradas mantidas no ring buffer em memória.

```
// Compliance: 12-month retention, log to file, include IP + payload
sgcWSAPIKeyManager1->Audit->Enabled = true;
sgcWSAPIKeyManager1->Audit->LogFile = "apikey-audit.log";
sgcWSAPIKeyManager1->Audit->IncludeIP = true;
sgcWSAPIKeyManager1->Audit->IncludePayload = true;
sgcWSAPIKeyManager1->Audit->RetentionDays = 365;
sgcWSAPIKeyManager1->Audit->MaxMemoryEntries = 50000;
```

Validação

O que faz: Informa ao gerenciador como os clientes apresentam suas chaves e sob quais condições as requisições são aceitas. *HeaderName* (padrão *X-API-Key*) e *QueryParamName* (padrão *api_key*) são os dois locais que *ExtractKeyFromHeaders* / *ExtractKeyFromQuery* examinam. *RequireHTTPS* rejeita tráfego em texto simples. *IPAllowlist* restringe as chaves a IPs de origem específicos. *FailClosed* (padrão *True*) rejeita qualquer requisição que o gerenciador não consiga autorizar positivamente.

Quando utilizá-lo: API corporativa: exija HTTPS para cada requisição e restrinja as chaves ao range CIDR de egresso do escritório, de modo que mesmo uma chave vazada não possa ser abusada de fora da rede corporativa.

Property	Type	Default	Descrição
HeaderName	string	X-API-Key	Nome do cabeçalho HTTP utilizado para ler a chave de entrada.
QueryParamName	string	api_key	Parâmetro de query-string utilizado como alternativa quando o cabeçalho está ausente.
RequireHTTPS	Boolean	False	Quando True, rejeita toda requisição não-HTTPS, independentemente da validade da chave.
IPAllowlist	TStringList	vazio	Quando não vazio, apenas requisições cujo IP aparece na lista são autorizadas.
FailClosed	Boolean	True	Quando True, qualquer requisição que o gerenciador não consiga autorizar positivamente é rejeitada.

```
// Corporate API: HTTPS-only, allow only office egress IPs, fail closed
sgcWSAPIKeyManager1->Validation->HeaderName = "X-API-Key";
sgcWSAPIKeyManager1->Validation->QueryParamName = "api_key";
sgcWSAPIKeyManager1->Validation->RequireHTTPS = true;
sgcWSAPIKeyManager1->Validation->FailClosed = true;
sgcWSAPIKeyManager1->Validation->IPAllowlist->Add("203.0.113.10");
sgcWSAPIKeyManager1->Validation->IPAllowlist->Add("203.0.113.11");
```

Stats

O que faz: Objeto contador somente leitura que expõe estatísticas ao vivo sobre o key store. Todos os contadores são atualizados atomicamente dentro da seção crítica interna do componente.

Quando usá-lo: Dashboard de administração ao vivo. Faça polling de Stats a partir de um *TTimer* e pinte os *ActiveKeys* / *RevokedKeys* / *TotalValidations* / *TotalValidationFailures* atuais para que os operadores possam ver a pressão de autenticação em tempo real.

Property	Type	Descrição

TotalKeys	Int64	Número total de chaves já emitidas por esta instância do gerenciador.
ActiveKeys	Int64	Número de chaves atualmente no status kksActive.
RevokedKeys	Int64	Número de chaves atualmente no status kksRevoked.
ExpiredKeys	Int64	Número de chaves atualmente no status kksExpired.
TotalValidations	Int64	Número total de chamadas ValidateKey / IsRequestAuthorized.
TotalValidationFailures	Int64	Número total de chamadas de validação que rejeitaram a chave.
LastIssuedAt	TDa- teTi- me	Timestamp da chamada IssueKey mais recente.

```
// Poll from a TTimer and update a dashboard
void __fastcall TForm1::StatsTimer(TObject *Sender)
{
    LabelActive->Caption = "Active: " + IntToStr(sgcWSAPIKeyManager1->Stats->ActiveKeys);
    LabelRevoked->Caption = "Revoked: " + IntToStr(sgcWSAPIKeyManager1->Stats->RevokedKeys);
    LabelValidations->Caption = "Validations: " + IntToStr(sgcWSAPIKeyManager1->Stats->TotalValidations);
    LabelFailures->Caption = "Failures: " + IntToStr(sgcWSAPIKeyManager1->Stats->TotalValidationFailures);
}
```

Eventos

OnKeyIssued: Disparado quando uma nova chave foi emitida. Fornece o proprietário, a chave bruta em texto simples (última chance de observá-la) e o array de escopos. Entregue o texto simples ao cliente a partir deste handler.

OnKeyValidated: Disparado toda vez que ValidateKey é concluído. Fornece a chave, uma flag booleana *aValid* e uma string de motivo legível por máquina.

OnKeyRevoked: Disparado quando uma chave foi revogada. Fornece a chave e o motivo da revogação.

OnKeyRotated: Disparado quando uma chave foi rotacionada. Fornece a chave antiga e a chave nova para que você possa notificar o cliente.

OnKeyExpired: Disparado pela varredura de expiração em segundo plano quando o ExpiresAt de uma chave foi atingido (ou NotifyBeforeExpirySec segundos antes).

OnValidation: Hook de decisão final disparado durante ValidateKey. Defina *Allow* := False e forneça um *Reason* para rejeitar uma chave de outra forma válida (regras de negócio personalizadas).

OnAuditEvent: Disparado para cada entrada do log de auditoria. Fornece o registro TsgcAPIKeyAuditEntry (Timestamp, KeyPrefix, Action, IP, Details) — ideal para encaminhar a um SIEM externo.

Métodos Públicos

Method	Descrição
IssueKey(Owner, Scopes, ExpiresInSec)	Gera, faz hash e armazena uma nova chave; retorna o texto simples (única vez em que pode ser visto). Passe ExpiresInSec = 0 para herdar Expiration.DefaultTTLSec.
ValidateKey(Key, RequiredScope, IP)	Valida uma chave em texto simples. Opcionalmente impõe um escopo obrigatório e registra o IP do solicitante. Retorna True em caso de sucesso.
RevokeKey(Key, Reason)	Marca uma chave como revogada; chamadas ValidateKey subsequentes retornam False.
RotateKey(OldKey, out NewKey)	Emite uma nova chave para o mesmo proprietário e escopos; a chave antiga é marcada como kksRotated e honrada durante GracePeriodSec.
RenewKey(Key, NewExpiresInSec)	Estende a expiração da chave pelo número de segundos fornecido a partir de agora.
GetKeyInfo(Key)	Retorna um registro TsgcAPIKeyInfo com proprietário, escopos, status, timestamps e contagem de uso.
GetKeysByOwner(Owner)	Retorna os identificadores de chave com hash associados ao owner fornecido.
ListScopes(Key)	Retorna os scopes anexados a uma chave.
IsExpired(Key)	Retorna True se a chave passou de sua expiração.
IsRevoked(Key)	Retorna True se a chave estiver revogada.
KeyExists(Key)	Retorna True se a chave for conhecida (ativa, revogada ou expirada).
GrantScope(Key, Scope)	Adiciona um escopo a uma chave existente.
RevokeScope(Key, Scope)	Remove um escopo de uma chave existente.
HasScope(Key, Scope)	Retorna True se a chave possuir o escopo fornecido.
GetAuditLog(Key, MaxEntries)	Retorna entradas de auditoria; quando Key está vazio, retorna entradas de todas as chaves.
ClearAuditLog(Key)	Limpa o log de auditoria para uma chave específica ou (quando vazio) para todas as chaves.
SaveToFile(FileName)	Persiste as chaves e o log de auditoria no arquivo fornecido (recorre a Storage.FileName quando vazio).
LoadFromFile(FileName)	Restaura chaves e log de auditoria a partir de um arquivo.
ExportKeys(Stream)	Exportação de chaves e log de auditoria baseada em stream.
ImportKeys(Stream)	Importação de chaves e log de auditoria baseada em stream.

PurgeExpired	Remove todas as chaves cujo ExpiresAt está no passado.
Count	Retorna o número de chaves armazenadas atualmente.
IsRequestAuthorized(Headers, QueryString, IP, RequiredScope)	Verificação de autorização única: extrai a chave dos cabeçalhos ou da query string, valida-a e, opcionalmente, impõe um escopo.
ExtractKeyFromHeaders(Headers)	Analisa o valor de Validation.HeaderName de um bloco de cabeçalho HTTP bruto.
ExtractKeyFromQuery(QueryString)	Analisa o valor de Validation.QueryParamName a partir de uma query string bruta.
IsConnectionAllowed(IP)	Hook do servidor: retorna False quando FailClosed é True e IPAllowlist não está vazia e o IP não está listado. Chamado automaticamente.
IsMessageAllowed(IP, Message)	Hook do servidor chamado automaticamente para cada mensagem de entrada. Por padrão, é um no-op em nível de IP.
RegisterConnection(IP)	Rastreia uma nova conexão. Chamado automaticamente pelo servidor.
UnregisterConnection(IP)	Libera o rastreamento de uma conexão. Chamado automaticamente na desconexão.

Integração Automática

Quando atribuído à propriedade APIKeyManager de um servidor, o componente automaticamente:

- **Em uma nova conexão:** O servidor chama IsConnectionAllowed(IP), que impõe a IPAllowlist opcional quando FailClosed é True (caso contrário, é um no-op). RegisterConnection(IP) é invocado para rastrear a conexão ativa.
- **Ao receber uma mensagem:** O servidor chama IsMessageAllowed(IP, Message), também um no-op em nível de IP, a menos que a lista de permissões rejeite o IP. Esses hooks *não* inspecionam cabeçalhos HTTP, portanto a autenticação por API-key não é realizada aqui.
- **Request authorization (o usuário deve conectar):** Chame *IsRequestAuthorized(Connection.HeadersRequest.Text, Connection.URL, Connection.IP, RequiredScope)* a partir do evento OnConnect do servidor. Este é o ponto onde os cabeçalhos HTTP brutos estão disponíveis, então é aqui que o cabeçalho X-API-Key é extraído, validado e opcionalmente verificado quanto ao escopo. Desconecte a conexão quando a função retornar False.
- **Lado do cliente:** Defina o cabeçalho X-API-Key (ou o HeaderName configurado) a partir do evento OnHandshake / OnBeforeConnect do cliente para que o servidor possa lê-lo durante o upgrade HTTP.
- **Ao desconectar:** UnregisterConnection(IP) é chamado para liberar o rastreamento interno.

Exemplo Completo

```
// Enable and configure the API key manager
sgcWSAPIKeyManager1->Enabled = true;

// Branded keys
```

```

sgcWSAPIKeyManager1->Generation->KeyPrefix = "sgc_live_";
sgcWSAPIKeyManager1->Generation->KeyLength = 32;

// Persist to an encrypted file that survives restarts
sgcWSAPIKeyManager1->Storage->StorageType = kstFile;
sgcWSAPIKeyManager1->Storage->FileName = "apikeys.dat";
sgcWSAPIKeyManager1->Storage->EncryptAtRest = true;
sgcWSAPIKeyManager1->Storage->EncryptionKey = "master-secret";
sgcWSAPIKeyManager1->Storage->AutoSaveSeconds = 60;

// SHA-256 with a static salt
sgcWSAPIKeyManager1->Hashing->Algorithm = khaSHA256;
sgcWSAPIKeyManager1->Hashing->Salt = "my-salt";

// Scope catalog
sgcWSAPIKeyManager1->Scopes->Enabled = true;
TsgcAPIKeyScopeItem *scopeRead = (TsgcAPIKeyScopeItem*)
    sgcWSAPIKeyManager1->Scopes->Scopes->Add();
scopeRead->Name = "read:orders";
scopeRead->Description = "Read orders";
TsgcAPIKeyScopeItem *scopeWrite = (TsgcAPIKeyScopeItem*)
    sgcWSAPIKeyManager1->Scopes->Scopes->Add();
scopeWrite->Name = "write:orders";
scopeWrite->Description = "Create orders";

// 90-day default TTL and 24-hour rotation grace period
sgcWSAPIKeyManager1->Expiration->DefaultTTLSec = 90 * 86400;
sgcWSAPIKeyManager1->Rotation->Enabled = true;
sgcWSAPIKeyManager1->Rotation->GracePeriodSec = 86400;

// Full audit log, 12-month retention
sgcWSAPIKeyManager1->Audit->Enabled = true;
sgcWSAPIKeyManager1->Audit->LogFile = "apikeys-audit.log";
sgcWSAPIKeyManager1->Audit->IncludeIP = true;
sgcWSAPIKeyManager1->Audit->RetentionDays = 365;

// Validation: header-based, fail closed
sgcWSAPIKeyManager1->Validation->HeaderName = "X-API-Key";
sgcWSAPIKeyManager1->Validation->QueryParamName = "api_key";
sgcWSAPIKeyManager1->Validation->FailClosed = true;

// Load previously saved keys and attach to server
sgcWSAPIKeyManager1->LoadFromFile("apikeys.dat");
sgcWebSocketHTTPServer1->APIKeyManager = sgcWSAPIKeyManager1;
sgcWebSocketHTTPServer1->Active = true;

// Issue a new key for a customer (plaintext returned only here)
DynamicArray<String> scopes;
scopes.Length = 2;
scopes[0] = "read:orders";
scopes[1] = "write:orders";
String LKey = sgcWSAPIKeyManager1->IssueKey("customer-123", scopes, 30 * 86400);
String LNewKey;
ShowMessage("Deliver to customer: " + LKey);

// Authorize every incoming connection in OnConnect
void __fastcall TForm1::ServerConnect(TsgcWSConnection *Connection)
{
    if (!sgcWSAPIKeyManager1->IsRequestAuthorized(
        Connection->HeadersRequest->Text,
        Connection->URL, Connection->IP, "read:orders"))
        Connection->Disconnect();
}

// Rotate a leaked key (old one valid 24h during grace)
sgcWSAPIKeyManager1->RotateKey(LKey, LNewKey);

// Revoke a key on subscription cancellation
sgcWSAPIKeyManager1->RevokeKey(LNewKey, "customer cancelled subscription");

```

Segurança de Threads

O gerenciador de chaves de API é totalmente thread-safe. Todos os métodos públicos usam seções críticas internas e listas thread-safe para proteger o acesso concorrente ao armazenamento de chaves, índice de escopos, índice de proprietários e log de auditoria. Um único componente pode ser compartilhado por várias instâncias de servidor e acessado a partir de qualquer thread (handlers de evento do servidor, threads de timer, workers em segundo plano) sem sincronização externa.

Referência

- Propriedades — 10 propriedades disponíveis no componente.
- Métodos — 26 métodos públicos.
- Eventos — 7 eventos que você pode tratar.

TsgcWebSocketLoadBalancerServer

Componente *TsgcWebSocketLoadBalancerServer*.

Introdução

O componente **TsgcWebSocketLoadBalancerServer** permite que você faça o balanceamento de carga dos protocolos **WebSocket** e **HTTP**. Para o protocolo WebSocket, ele distribui as mensagens entre um grupo de servidores e distribui as conexões de clientes usando uma sequência aleatória ou um algoritmo de menos conexões.

O Load Balancer Server herda todos os métodos e propriedades do [TsgcWebSocketHTTPServer](#).

Load Balancer Configuration

O servidor Load Balancer é descendente de [TsgcWebSocketHTTPServer](#), então leia a documentação sobre o [TsgcWebSocketHTTPServer](#) para saber como configurá-lo.

Além disso, o Load Balancer possui a propriedade **LoadBalancer**, que tem as seguintes propriedades:

- **LoadBalancing**: configure aqui como distribuir as conexões
 - **IbRandom**: (padrão) toda vez que um novo cliente solicita uma conexão, retornará um servidor aleatório.
 - **IbConnections**: toda vez que um novo cliente solicita uma conexão, retornará o servidor com o menor número de clientes conectados.
- **Protocols**: configure quais protocolos estão habilitados
 - **WebSocket**: se true, as conexões websocket serão tratadas pelo Load Balancer Server.
 - **HTTP**: se true, as conexões http serão tratadas pelo Load Balancer Server.

Configuração do Servidor de Backup

Os Backup Servers (os servidores atrás do balanceador de carga) podem ser um [TsgcWebSocketServer](#), [TsgcWebSocketHTTPServer](#) ou um [Datsnap Server](#).

Esses servidores têm uma propriedade chamada **LoadBalancer** onde você pode configurar a conexão entre o LoadBalancer Server e os Servidores de Backup.

- **Enabled**: defina como true se você quiser usar como um servidor de backup.
- **Host**: o host onde o LoadBalancer está localizado.
- **Port**: a porta de escuta do LoadBalancer.
- **Guid**: id único que identifica este servidor.
- **Bindings**: os endereços públicos para onde as conexões serão encaminhadas. Exemplo: se o servidor WebSocket de Backup está escutando na porta 8000 e o endereço IP é 1.1.1.1, utilize o seguinte: ws://1.1.1.1:8000;
- **AutoRegisterBindings**: se habilitado, o LoadBalancer Server usará a propriedade Bindings do servidor de backup para configurar os bindings públicos.
- **AutoRestart**: em segundos; se maior que zero, o cliente do load balancer do servidor de backup habilitará um watchdog interno que, a cada x segundos, verificará se a conexão está ativa; se estiver fechada, tentará reconectar.

Referência

- Propriedades — 18 propriedades disponíveis no componente.
- Métodos — 9 métodos públicos.

- Eventos — 22 eventos que você pode tratar.

TsgcWebSocketProxyServer

TsgcWebSocketProxyServer implementa um Componente Servidor WebSocket que escuta conexões WebSocket de clientes e encaminha as conexões de dados para um servidor TCP/IP normal.

Introdução

TsgcWebSocketProxyServer implementa um Componente Servidor WebSocket que escuta conexões WebSocket de clientes e encaminha as conexões de dados para um servidor TCP/IP normal. Isso é especialmente útil para conexões de navegador, pois permite que um navegador se conecte virtualmente a qualquer servidor.

Referência

- Propriedades — 19 propriedades disponíveis no componente.
- Métodos — 7 métodos públicos.
- Eventos — 22 eventos que você pode tratar.

TsgclWWebSocketClient

O TsgclWWebSocketClient implementa um Componente WebSocket IntraWeb que pode conectar-se a um Servidor WebSocket.

Introdução

O TsgclWWebSocketClient implementa um Componente WebSocket IntraWeb que pode conectar a um Servidor WebSocket. Siga os passos abaixo para configurar este componente:

1. Solte um componente TsgclWWebSocketClient no formulário
2. Defina Host e Port (o padrão é 80) para se conectar a um Servidor WebSocket disponível. Você pode definir a propriedade URL e Host, Port, Parameters... serão atualizados a partir da URL. Exemplo: wss://127.0.0.1:8080/ws/ resultará em:

```
oClient = new TsgclWWebSocketClient();
oClient->Host = "127.0.0.1";
oClient->Port = 80;
oClient->TLS = true;
oClient->Options->Parameters = "/ws/";
```

3. Você pode selecionar se deseja TLS (conexão segura) ou não, por padrão não está ativado.

4. Set Transports allowed.

WebSockets: it will use standard WebSocket implementation

Emulation: se o navegador não suportar WebSockets, então ele utilizará uma conexão de callback AJAX em loop

6. Create an Async Procedure and set property Active := True

Methods

Open: Opens a WebSocket Connection.

Close: Closes a WebSocket Connection.

WriteData: sends a message to WebSocket Server.

Properties

Connected: é uma variável somente leitura e retorna True se a conexão estiver Ativa, caso contrário retorna False.

JSOpen: aqui você pode incluir código JavaScript no lado do cliente quando uma conexão é aberta.

JSClose: aqui você pode incluir código JavaScript no lado do cliente quando uma conexão é fechada.

JSMessage: aqui você pode incluir código JavaScript no lado do cliente quando os clientes recebem uma mensagem do servidor. Você pode obter a string da mensagem, utilizando a variável JavaScript "text".

JSError: aqui você pode incluir código JavaScript no lado do cliente quando um erro é gerado. Você pode obter a mensagem de erro, utilizando a variável JavaScript "text".

Referência

- Propriedades — 11 propriedades disponíveis no componente.
- Métodos — 4 métodos públicos.
- Eventos — 5 eventos que você pode tratar.

TsgcWSConnection

TsgcWSConnection é um wrapper para conexões WebSocket de cliente. Você pode acessar este objeto nos Eventos do Servidor ou do Cliente.

Métodos

WriteData: envia uma mensagem ao cliente.

Close: envia uma mensagem de fechamento ao outro peer. Um "CloseCode" pode ser especificado opcionalmente. Por padrão, o valor enviado é o código de fechamento NORMAL. Se você enviar um código de fechamento negativo, o motivo do fechamento não será enviado.

Disconnect: fecha a conexão do cliente a partir do lado do servidor. Um "CloseCode" pode ser especificado opcionalmente.

Ping: envia um ping ao cliente.

AddTCPEndOfFrame: se a conexão for TCP puro, permite definir qual(is) byte(s) define(m) o fim da mensagem. A mensagem é armazenada em buffer até ser recebida completamente.

Subscribed: retorna se a conexão está assinada em um canal personalizado.

Subscribe: assina esta conexão em um canal. Posteriormente você pode fazer Broadcast de uma mensagem do componente servidor para todas as conexões assinadas neste canal.

UnSubscribe: cancela a inscrição desta conexão em um canal.

Properties

Protocol: retorna o subprotocolo usado nesta conexão.

IP: retorna o endereço IP do peer.

Port: retorna a Porta do Peer.

LocalIP: retorna o endereço IP do host.

LocalPort: retorna a Port do Host.

URL: retorna a URL requisitada pelo cliente.

Guid: retorna o ID da conexão.

HeadersRequest: retorna uma lista de Headers recebidos na Request.

HeadersResponse: retorna uma lista de Headers enviados como Resposta.

RecBytes: número de bytes recebidos.

SendBytes: número de bytes enviados.

Transport: retorna o tipo de transporte da conexão:

trpRFC6455: uma conexão WebSocket normal.

trpHixie76: uma conexão WebSocket utilizando a spec draft do WebSocket.

trpFlash: uma conexão WebSocket utilizando Flash como FallBack.

trpSSE: uma conexão Server-Sent Events.

trpTCP: conexão TCP simples.

TCPEndOfFrameScanBuffer: permite definir qual método utilizar para encontrar o fim da mensagem (se utilizar trpTCP como transporte).

eofScanNone: sempre que um novo pacote chega, o evento OnBinary é chamado.

eofScanLatestBytes: se os últimos bytes forem iguais aos bytes adicionados com o método AddTCPEndOfFrame, a mensagem OnBinary é chamada; caso contrário, este pacote é armazenado em buffer

eofScanAllBytes: procura em todo o pacote por bytes iguais aos bytes adicionados com o método AddTCPEndOfFrame. Se encontrados, o evento OnBinary é chamado; caso contrário, este pacote é armazenado em buffer

Data: objeto de dados de sessão do usuário, aqui você pode passar um objeto e acessá-lo sempre que precisar, por exemplo: você pode passar uma conexão com um banco de dados, propriedades de sessão do usuário...

Protocolos

Com WebSockets, você pode implementar subprotocolos, permitindo criar comunicações personalizadas. **Por exemplo**, você pode implementar um subprotocolo sobre o protocolo WebSocket para se comunicar com uma aplicação personalizada usando mensagens JSON, e pode implementar outro subprotocolo usando mensagens XML.

Quando uma conexão é aberta no lado do Servidor, ele valida se o subprotocolo enviado pelo cliente é suportado pelo servidor; se não, ele fecha a conexão. Um servidor pode implementar vários subprotocolos, mas apenas um pode ser usado em uma única conexão.

Os subprotocolos são muito úteis para criar aplicações personalizadas e garantir que todos os clientes suportem a mesma interface de comunicação.

Embora o nome do protocolo seja arbitrário, é recomendável usar nomes exclusivos como "dataset.essece.com"

Com o pacote `sgcWebSockets`, você pode construir seus próprios protocolos e pode utilizar os subprotocolos integrados fornecidos:

- 1. Protocol MQTT:** O MQTT é um protocolo de transporte de mensagens publish/subscribe Cliente Servidor. É leve, aberto, simples e projetado para ser fácil de implementar.
- 2. Protocolo AppRTC:** é uma aplicação de demonstração webrtc desenvolvida pelo Google e pela Mozilla, ela permite que ambos os navegadores "conversem" entre si usando a WebRTC API.
- 3. Protocolo WebRTC:** projeto de código aberto que visa habilitar a web com recursos de Comunicação em Tempo Real (RTC).
- 4. Protocol Files:** implementado utilizando mensagens binárias, fornece suporte para envio de arquivos: tamanho do pacote, autorização, QoS, confirmação de mensagem e mais.
- 5. Protocolo SGC:** implementado usando mensagens JSON-RPC 2.0, fornece os seguintes padrões: RPC, Pub-Sub, Mensagens Transacionais, Confirmação de Mensagens (Messages Acknowledgment) e mais.
- 6. Protocolo Dataset:** herda do Protocolo Padrão, pode enviar alterações de dataset (novo registro, salvar registro ou excluir um registro) do servidor para os clientes.
- 7. Protocolo Presence:** permite que você saiba quem está assinado em um canal, por exemplo: salas de chat, colaboradores em um documento, pessoas visualizando a mesma página web, competidores em um jogo...
- 8. Protocol WAMP 1.0:** subprotocolo WebSocket aberto que fornece dois padrões de mensagens assíncronas: RPC e PubSub.
- 9. Protocol WAMP 2.0:** subprotocolo WebSocket aberto que fornece dois padrões de mensagens assíncronas: RPC e PubSub.
- 10. Protocol STOMP:** O STOMP é o Simple (ou Streaming) Text Orientated Messaging Protocol. O STOMP fornece um formato de fio interoperável para que os clientes STOMP possam se comunicar com qualquer message broker STOMP, proporcionando interoperabilidade de mensageria fácil e amplamente difundida entre muitas linguagens, plataformas e brokers.
 - 10.1 STOMP para RabbitMQ:** cliente para o broker RabbitMQ.
 - 10.2 STOMP para ActiveMQ:** cliente para o Broker ActiveMQ.
- 11. Protocolo AMQP:** O Advanced Message Queuing Protocol (AMQP 0.9.1) é criado como um protocolo de padrão aberto que permite a interoperabilidade de mensagens entre sistemas, independentemente do fornecedor do message broker ou da plataforma utilizada.

12. Protocolo AMQP1: O Advanced Message Queuing Protocol (AMQP 1.0.0) foi criado como um protocolo padrão aberto que permite a interoperabilidade de mensageria entre sistemas, independentemente do fornecedor do message broker ou da plataforma utilizada.

13. Protocolo E2EE: As mensagens enviadas por uma conexão WebSocket são criptografadas de ponta a ponta no nível da aplicação, de modo que apenas os clientes em comunicação possam lê-las, mesmo que o servidor WebSocket retransmita os dados.

Se você precisar usar **mais de um protocolo usando uma única conexão** (exemplo: você pode precisar usar o **protocolo padrão** para tratar Remote Procedure Calls e o **protocolo Dataset** para tratar conexões de banco de dados), você pode atribuir um "Broker" a cada componente de protocolo e todas as mensagens serão trocadas usando esse protocolo intermediário (você pode conferir a "Tickets Demo" para obter um exemplo simples disso).

Protocolos podem ser registrados em **tempo de execução**, basta chamar o método **RegisterProtocol** e passar o componente de protocolo como parâmetro.

Referência JavaScript

Aqui você pode obter [mais informações](#) sobre a biblioteca JavaScript comum usada no sgcWebSockets.

Protocols Javascript

Por padrão, o JavaScript do sgcWebSockets utiliza o arquivo **sgcWebSocket.js**.

Aqui você pode encontrar os métodos disponíveis; você precisa substituir as variáveis `{%host%}` e `{%port%}` conforme necessário, exemplo: se você configurou seu servidor sgcWebSocket para escutar a porta 80 no site `www.example.com`, você precisa configurar seu acesso ao arquivo `sgcWebSocket.js` como:

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
```

Open Connection

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  var socket = new sgcWebSocket('ws://{%host%}:{%port%}');
</script>
```

O sgcWebSocket tem 3 parâmetros, apenas o primeiro é obrigatório:

```
sgcWebSocket(url, protocol, transport)
```

- **URL:** localização do servidor WebSocket, você pode usar "ws:" para conexões WebSocket normais e "wss:" para conexões WebSocket seguras.

```
sgcWebSocket('ws://127.0.0.1')
```

```
sgcWebSocket('wss://127.0.0.1')
```

- **Protocol:** se o servidor aceitar um ou mais protocolos, você pode definir qual protocolo deseja utilizar.

```
sgcWebSocket('ws://127.0.0.1', 'esegece.com')
```

- **Transport:** por padrão, primeiro tenta conectar usando conexão WebSocket e, se não for implementada pelo navegador, então tenta Server Sent Events como Transport.

Utilize WebSocket se estiver implementado; caso contrário, utilize Server Sent Events:

```
sgcWebSocket('ws://127.0.0.1')
```

Usar apenas WebSocket como transporte:

```
sgcWebSocket('ws://127.0.0.1', '', ['websocket'])
```

Utilizar somente Server Sent como transporte:

```
sgcWebSocket('ws://127.0.0.1', '', ['sse'])
```

Abrir Conexão Com Autenticação

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  var socket = new sgcWebSocket({ "host": "ws://{%host%}:{%port%}", "user": "admin", "password": "1234" });
</script>
```

Enviar mensagem

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  var socket = new sgcWebSocket('ws://{%host%}:{%port%}');
  socket.send('Hello sgcWebSockets!');
</script>
```

Show Alert with Message Received

```
<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>
```

Binary Message Received

```
<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>
```

Binary (Header + Image) Message Received

```
<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>
```

Show Alert OnConnect, OnDisconnect and OnError Events

```
<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>
```

Close Connection

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
    socket.close();
</script>
```

Obter Status da Conexão

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
    socket.state();
</script>
```

Protocol MQTT

O MQTT é um protocolo de transporte de mensagens publish/subscribe Client-Server. É leve, aberto, simples e projetado para ser fácil de implementar. Essas características tornam-no ideal para uso em muitas situações, incluindo ambientes restritos como na comunicação Machine to Machine (M2M) e em contextos da Internet das Coisas (IoT) onde um small code footprint é necessário e/ou a largura de banda da rede é limitada.

O protocolo executa sobre TCP/IP, ou sobre outros protocolos de rede que fornecem conexões ordenadas, sem perdas e bidirecionais. Seus recursos incluem:

- Uso do padrão de mensagem publish/subscribe, que fornece distribuição de mensagens um-para-muitos e desacoplamento de aplicações.
- Um transporte de mensagens que é agnóstico ao conteúdo da carga útil.
- Três qualidades de serviço para entrega de mensagens:
 - "No máximo uma vez", onde as mensagens são entregues de acordo com o melhor esforço do ambiente operacional. Pode ocorrer perda de mensagens. Este nível poderia ser utilizado, por exemplo, com dados de sensores ambientais, onde não importa se uma leitura individual é perdida, pois a próxima será publicada logo em seguida.
 - "Pelo menos uma vez", onde as mensagens têm garantia de chegar, mas duplicatas podem ocorrer.
 - "Exatamente uma vez", em que se garante que as mensagens chegam exatamente uma vez. Este nível pode ser usado, por exemplo, com sistemas de cobrança onde mensagens duplicadas ou perdas poderiam levar à aplicação de cobranças incorretas.
- Um pequeno overhead de transporte e trocas de protocolo minimizadas para reduzir o tráfego de rede.
- Um mecanismo para notificar as partes interessadas quando ocorre uma desconexão anormal.

Features

- Suporta as versões MQTT **3.1.1** e **5.0**.
- Padrão de mensagem **Publish/subscribe** para fornecer distribuição de mensagens um-para-muitos e desacoplamento de aplicações.
- **Acknowledgment** das mensagens enviadas.
- Implementa **QoS** (Quality of Service) para entrega de mensagens (todos os níveis: At most once, At least once e Exactly once)
- **Last Will Testament**.
- **Conexões** seguras.
- **HeartBeat** e **Watchdog**.
- **Autenticação** no servidor.

Componentes

[TsgcWSPClient_MQTT](#): Componente MQTT Client.

Most common uses

- **Conexão**
 - [Conexão MQTT do Cliente](#)
 - [Conectar a Servidores MQTT Mosquitto](#)
 - [Sessões MQTT do Cliente](#)
 - [Client MQTT Version](#)
- **Publish & Subscribe**
 - [MQTT Publish Subscribe](#)
 - [MQTT Topics](#)

- [MQTT Subscribe](#)
- [MQTT Publish Message](#)
- [MQTT Receive Messages](#)
- [MQTT Publish and Wait Response](#)
- **Outros**
 - [MQTT Clear Retained Messages](#)

TsgcWSPClient_MQTT

O componente MQTT fornece uma implementação de cliente MQTT leve e completa, com suporte às versões 3.

Introdução

O componente MQTT fornece uma implementação de cliente MQTT leve e completa, com suporte às versões 3.1.1 e 5.0. O componente suporta conexões em texto puro e seguras tanto sobre TCP padrão quanto sobre WebSockets.

A conexão a um servidor MQTT é simples, você precisa arrastar este componente para o formulário e selecionar um componente [TsgcWebSocketClient](#) utilizando a propriedade Client. Defina host e port em TsgcWebSocketClient e defina Active := True para conectar.

O MQTT v5.0 não é retrocompatível (como o v3.1.1). Obviamente, muitas coisas novas são introduzidas, então as implementações existentes precisam ser revisadas.

De acordo com a especificação, o MQTT v5.0 adiciona um número significativo de novos recursos ao MQTT, mantendo grande parte do núcleo em vigor.

- A funcionalidade da flag Clean Session é dividida em 2 propriedades para permitir um controle mais fino sobre os dados de estado da sessão: o parâmetro CleanStart e o novo SessionExpInterval.
- Desconexão do servidor: permite que o DISCONNECT seja enviado pelo servidor para indicar o motivo do fechamento da conexão.
- Todos os pacotes de resposta (CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK, DISCONNECT) agora contêm um reason code e uma reason string descrevendo por que as operações tiveram sucesso ou falharam.
- Autenticação aprimorada: Fornece um mecanismo para habilitar autenticação no estilo desafio/resposta, incluindo autenticação mútua. Isso permite que a autenticação no estilo SASL seja usada se suportada tanto pelo Cliente quanto pelo Servidor, e inclui a capacidade de um Cliente se reautenticar dentro de uma conexão.
- O padrão Request / Response é formalizado pela adição do ResponseTopic.
- Shared Subscriptions: Adiciona suporte a assinaturas compartilhadas, permitindo consumidores com balanceamento de carga de uma assinatura.
- Os Topic Aliases podem ser enviados tanto pelo cliente quanto pelo servidor para se referir a filtros de tópico por identificadores numéricos mais curtos, a fim de economizar largura de banda.
- Os servidores podem comunicar quais recursos suportam em ConnectionProperties.
- Referência de servidor: Permite que o Servidor especifique um Servidor alternativo a ser usado em CONNACK ou DISCONNECT. Isso pode ser usado como um redirecionamento ou para fazer provisionamento.
- Mais: expiração de mensagem, Receive Maximums e Maximum Packet Sizes, e um intervalo de Will Delay são todos suportados.

Referência

- Propriedades — 10 propriedades disponíveis no componente.
- Métodos — 9 métodos públicos.
- Eventos — 13 eventos que você pode tratar.

TsgcWSPClient_MQTT | Client MQTT Connect

Para se conectar a um MQTT Server, você deve primeiro criar um [TsgcWebSocketClient](#) e um [TsgcWSPClient_MQTT](#). Em seguida, você deve anexar o componente MQTT ao WebSocket Client.

Uso básico

Conecta ao servidor MQTT Mosquitto usando o protocolo websocket. Assina o tópico: "topic1" após conectar.

```
oClient = new TsgcWebSocketClient();
oClient->Host = "test.mosquitto.org";
oClient->Port = 8080;
oMQTT = new TsgcWSPClient_MQTT();
oMQTT->Client = oClient;
oClient->Active = true;

void OnMQTTConnect(TsgcWSConnection *Connection, const bool Session, const int ReasonCode,
    const string ReasonName, const TsgcWSMQTTCONNACKProperties *ConnectProperties);
{
    oMQTT->Subscribe("topic1");
}
```

Client Identifier

O MQTT requer um **Client Identifier** para identificar a conexão do cliente. O componente define um **valor aleatório** automaticamente, mas você pode definir seu próprio Client Identifier se necessário; para isso, basta tratar o evento **OnBeforeConnect** e definir seu valor no parâmetro aClientIdentifier.

```
void OnMQTTBeforeConnect(TsgcWSConnection *Connection, ref bool aCleanSession,
    ref string aClientIdentifier)
{
    aClientIdentifier = "your client id";
}
```

Autenticação

Alguns servidores requerem um usuário e senha para **autorizar conexões MQTT**. Use a propriedade **Authentication** para definir o valor de nome de usuário e senha antes de conectar ao servidor.

```
oMQTT = new TsgcWSPClient_MQTT();
oMQTT->Authentication->Enabled = true;
oMQTT->Authentication->UserName = "your user";
oMQTT->Authentication->Password = "your password";
```

TsgcWSPClient_MQTT | Connect MQTT Mosquitto

Use as seguintes configurações de exemplo para conectar a um servidor MQTT Mosquitto.

MOSQUITTO MQTT WebSockets

```
oClient = new TsgcWebSocketClient();
oClient->Host = "test.mosquitto.org";
oClient->Port = 8080;
oMQTT = new TsgcWSPClient_MQTT();
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 = tls1_2;
oMQTT = new TsgcWSPClient_MQTT();
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 = new TsgcWSPClient_MQTT();
oMQTT->Client = oClient;
oClient->Active = true;
```

MOSQUITTO MQTT TCP Simples TLS

```
oClient = new TsgcWebSocketClient();
oClient->Host = "test.mosquitto.org";
oClient->Port = 8083;
oClient->Specifications->RFC6455 = false;
oClient->TLS = true;
oClient->TLSOptions->Version = tls1_2;
oMQTT = new TsgcWSPClient_MQTT();
oMQTT->Client = oClient;
oClient->Active = true;
```

TsgcWSPClient_MQTT | Client MQTT Sessions

Clean Start

OnMQTTBeforeConnect, há um parâmetro chamado **aCleanSession**. Se o valor deste parâmetro for **True**, significa que o cliente **quer iniciar uma nova sessão**, portanto, se o servidor tiver alguma sessão armazenada, ele deve descartá-la. Assim, quando o evento **OnMQTTConnect** é disparado, o parâmetro **aSession** será **false**. Se o valor deste parâmetro for **False** e houver uma sessão associada a este client identifier, o servidor deve retomar as comunicações com o cliente no estado da sessão existente.

Portanto, se o cliente tiver uma **desconexão inesperada**, e você quiser **recuperar a sessão** de onde foi desconectado, em **OnMQTTBeforeConnect** defina **aCleanSession = True** e **aClientIdentifier = Client ID da Sessão**.

Session

Após uma conexão bem-sucedida, verifique o evento **OnMQTTConnect**, o valor do parâmetro **Session**.

Session = true, significa que a sessão foi retomada.

Session = false, significa que é uma nova sessão.

```
void OnMQTTBeforeConnect(TsgcWSConnection *Connection, ref bool aCleanSession,
    ref string aClientIdentifier)
{
    aCleanSession = false;
    aClientIdentifier = "previous client id";
}

void OnMQTTConnect(TsgcWSConnection *Connection, const bool Session, const int ReasonCode,
    const string ReasonName, const TsgcWSMQTTCONNACKProperties *ConnectProperties);
{
    if (Session == true)
    {
        WriteLine("Session resumed");
    }
    else
    {
        WriteLine("New Session");
    }
}
```

TsgcWSPClient_MQTT | Client MQTT Version

Atualmente, o Cliente MQTT suporta as seguintes especificações:

- **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>

Você pode selecionar qual é a versão que o componente MQTT Client utilizará por meio da propriedade MQTTVersion.

MQTT 3.1.1: TsgcWSPClient_MQTT.Version = mqtt311

MQTT 5.0: sgcWSPClient_MQTT.Version = mqtt5

TsgcWSPClient_MQTT | MQTT Publish Subscribe

O padrão publish/subscribe (também conhecido como pub/sub) oferece uma alternativa à arquitetura cliente-servidor tradicional. No modelo cliente-servidor, um cliente comunica-se diretamente com um endpoint. O modelo pub/sub **desacopla o cliente que envia uma mensagem (o publicador) do cliente ou clientes que recebem as mensagens (os assinantes)**. Os publicadores e assinantes nunca entram em contato diretamente. Na verdade, eles nem sequer sabem que o outro existe. **A conexão entre eles é tratada por um terceiro componente (o broker)**. O trabalho do broker é filtrar todas as mensagens recebidas e distribuí-las corretamente aos assinantes.

Com o **TsgcWSPClient_MQTT** você pode **Publicar mensagens e Assinar Tópicos**.

Subscribe Topic

Assine o Tópico "topic1" após uma conexão bem-sucedida.

```
oClient = new TsgcWebSocketClient();
oClient->Host = "test.mosquitto.org";
oClient->Port = 8080;
oMQTT = new TsgcWSPClient_MQTT();
oMQTT->Client = oClient;
oClient->Active = true;

void OnMQTTConnect(TsgcWSConnection *Connection, const bool Session, const int ReasonCode,
    const string ReasonName, const TsgcWSMQTTCONNACKProperties *ConnectProperties);
{
    oMQTT->Subscribe("topic1");
}
```

Publish Message

Publica uma mensagem para todos os assinantes de "topic1"

```
oClient = new TsgcWebSocketClient();
oClient->Host = "test.mosquitto.org";
oClient->Port = 8080;
oMQTT = new TsgcWSPClient_MQTT();
oMQTT->Client = oClient;
oClient->Active = true;

void OnMQTTConnect(TsgcWSConnection *Connection, const bool Session, const int ReasonCode,
    const string ReasonName, const TsgcWSMQTTCONNACKProperties *ConnectProperties);
{
    oMQTT->Publish("topic1", "Hello Subscribers topic1");
}
```

TsgcWSPClient_MQTT | MQTT Topics

Tópicos

No MQTT, a palavra tópico refere-se a uma string UTF-8 que o broker usa para filtrar mensagens para cada cliente conectado. O tópico consiste em um ou mais níveis de tópico. Cada nível de tópico é separado por uma barra (separador de nível de tópico)

`myHome / groundfloor / livingroom / temperature`

Em comparação com uma message queue, os tópicos MQTT são muito leves. O cliente não precisa criar o tópico desejado antes de publicar ou assinar nele. O broker aceita cada tópico válido sem qualquer inicialização prévia. Observe que cada tópico deve conter pelo menos 1 caractere e que a string do tópico permite espaços vazios. Os tópicos são sensíveis a maiúsculas e minúsculas.

WildCards

Quando um cliente assina um tópico, ele pode assinar o tópico exato de uma mensagem publicada ou pode utilizar wildcards para assinar múltiplos tópicos simultaneamente. Um wildcard só pode ser utilizado para assinar tópicos, não para publicar uma mensagem. Existem dois tipos diferentes de wildcards: `_single-level` e `_multi-level`.

Nível Único: +

Como o nome sugere, um wildcard de nível único substitui um nível de tópico. O símbolo de mais representa um wildcard de nível único em um tópico.

`myHome / groundfloor / + / temperature`

Qualquer tópico corresponde a um tópico com curinga de nível único se ele contiver uma string arbitrária no lugar do curinga. Por exemplo, uma assinatura de `_myhome/groundfloor/+/temperature` pode produzir os seguintes resultados:

```
YES => myHome / groundfloor / livingroom / temperature
YES => myHome / groundfloor / kitchen / temperature
NO => myHome / groundfloor / livingroom / brightness
NÃO => myHome / firstfloor / livingroom / temperature
NÃO => myHome / groundfloor / kitchen / fridge / temperature
```

Multi Level:

O wildcard de múltiplos níveis cobre muitos níveis de tópico. O símbolo de cerquilha representa o wildcard de múltiplos níveis no tópico. Para que o broker determine quais tópicos correspondem, o wildcard de múltiplos níveis deve ser colocado como o último caractere do tópico e precedido por uma barra normal.

`myHome / groundfloor / #`

```
YES => myHome / groundfloor / livingroom / temperature
YES => myHome / groundfloor / kitchen / temperature
SIM => myHome / groundfloor / kitchen / brightness
NÃO => myHome / firstfloor / kitchen / temperature
```

Quando um cliente assina um tópico com um curinga multinível, ele recebe todas as mensagens de um tópico que começa com o padrão antes do caractere curinga, não importa quão longo ou profundo seja o tópico. Se você es-

pecificar apenas o curinga multinível como tópico (`_#`), você recebe todas as mensagens enviadas ao broker MQTT.

TsgcWSPClient_MQTT | MQTT Subscribe

Você pode se inscrever em um Topic utilizando o método Subscribe de TsgcWSPClient_MQTT. Este método tem os seguintes parâmetros:

Topic: é o nome do tópico a ser assinado.

QoS: um dos 3 níveis de QoS (nem todos os brokers suportam os 3 níveis). Se não especificado, usa mtqsAtMostOnce. Leia mais sobre [Níveis de QoS](#).

SubscribeProperties: se MQTT 5.0, são propriedades adicionais sobre as inscrições.

Subscribe QoS = At Least Once

```
MQTT->Subscribe("topic1", mtqsAtLeastOnce);
```

Subscribe MQTT 5.0

```
oProperties = new TsgcWSMQTTSubscribe_Properties();  
oProperties->SubscriptionIdentifier = 1234;  
oProperties->UserProperties->Add("name=value");  
  
MQTT->Subscribe("topic1", mtqsAtMostOnce, oProperties);
```


TsgcWSPClient_MQTT | MQTT Publish Message

Você pode publicar mensagens a todos os assinantes de um Topic utilizando o método **Publish**, que possui os seguintes parâmetros:

Topic: é o nome do tópico onde a mensagem será publicada.

Text: é o texto da mensagem.

QoS: um dos 3 níveis de QoS (nem todos os brokers suportam os 3 níveis). Se não especificado, usa `mqttAtMostOnce`. Leia mais sobre [Níveis de QoS](#).

Retain: se `true`, esta mensagem será retida. E toda vez que um novo cliente assinar este tópico, esta mensagem será enviada a esse cliente.

PublishProperties: se MQTT 5.0, estas são as propriedades da mensagem.

Publicar uma mensagem simples

```
MQTT->Publish("topic1", "Hello Subscribers topic1");
```

Publish QoS = At Least Once

```
MQTT->Publish("topic1", "Hello Subscribers topic1", mqttAtLeastOnce);
```

Publish Retained message

```
MQTT->Publish("topic1", "Hello Subscribers topic1", mqttAtMostOnce, true);
```

TsgcWSPClient_MQTT | MQTT Receive Messages

As mensagens enviadas pelo servidor são recebidas no evento **OnMQTTPublish**. Este evento tem os seguintes parâmetros:

Topic: é o nome do tópico associado a esta mensagem.

Text: é o texto da mensagem.

PublishProperties: se MQTT 5.0, estas são as propriedades da mensagem publicada.

Ler Mensagens publicadas

```
void OnMQTTPublish(TsgcWSConnection *Connection, string aTopic, string aText,
    TsgcWSMQTTPublishProperties *PublishProperties)
{
    WriteLine("Topic: " + aTopic + ". Message: " + aText);
}
```

TsgcWSPClient_MQTT | MQTT Receive Messages (Extended)

O evento **OnMQTTPublishEx** fornece o payload da mensagem publicada em múltiplos formatos por meio de um objeto **TsgcWSMQTTPublishData**. Este evento tem os seguintes parâmetros:

Topic: é o nome do tópico associado a esta mensagem.

Data: contém o payload da mensagem publicada. Ela tem as seguintes propriedades:

- **Value:** o payload como uma string.
- **Bytes:** o payload bruto como TBytes.
- **Stream:** o payload bruto como um TMemoryStream.

PublishProperties: se MQTT 5.0, estas são as propriedades da mensagem publicada.

Read published Messages (Extended)

```
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: " + IntToStr(aData->Bytes.Length));
    // read as stream
    WriteLine("Stream Size: " + IntToStr(aData->Stream->Size));
}
```

TsgcWSPClient_MQTT | Publish e Wait Response

O cliente MQTT permite o uso de alguns tipos de níveis de QoS; qualquer um desses níveis funciona em um nível diferente para garantir que as mensagens foram processadas conforme o esperado.

Existem os seguintes níveis de QoS:

- **mtqsAtMostOnce:** (por padrão) a mensagem é entregue de acordo com o melhor esforço da rede TCP/IP subjacente. Uma resposta não é esperada e nenhuma semântica de retry é definida no protocolo. A mensagem chega ao servidor uma vez ou não chega.
- **mtqsAtLeastOnce:** o recebimento de uma mensagem pelo servidor é confirmado por uma mensagem ACKNOWLEDGMENT. Se houver uma falha identificada do link de comunicação ou do dispositivo de envio, ou a mensagem de confirmação não for recebida após um período de tempo especificado, o remetente reenvia a mensagem. A mensagem chega ao servidor pelo menos uma vez. Uma mensagem com QoS nível 1 tem um parâmetro ID na mensagem.
- **mtqsExactlyOnce:** onde as mensagens têm garantia de chegar exatamente uma vez. Este nível pode ser usado, por exemplo, com sistemas de faturamento, onde mensagens duplicadas ou perdidas poderiam levar a cobranças incorretas. Se houver uma falha identificada no link de comunicação ou no dispositivo de envio, ou se a mensagem de confirmação não for recebida após um período de tempo especificado, o remetente reenvia a mensagem.

Você pode tratar os eventos OnPubAck ou OnPubComp para saber se a mensagem foi processada pelo servidor, ou pode usar o método **PublishAndWait** para saber se a mensagem foi processada pelo servidor.

O uso de **PublishAndWait** é o mesmo que o método Publish normal, mas agora você tem um novo parâmetro chamado Timeout, onde o método retornará false se, após um certo período de tempo, não houver resposta do servidor. Por padrão, este valor é de 10 segundos.

```
if mqtt->PublishAndWait("topic", "text")
{
    ShowMessage("Message processed")
}
else
{
    ShowMessage("Message error");
}
```

TsgcWSPClient_MQTT | MQTT Clear Retained Messages

Por padrão, cada tópico MQTT pode ter uma mensagem retida. O mecanismo padrão do MQTT para limpar mensagens retidas é enviar uma mensagem retida com um payload vazio a um tópico. Isso removerá a mensagem retida.

```
MQTT->Publish("topic1", "", mqttAtMostOnce, true);
```

Protocolo AMQP 0.9.1

O **Advanced Message Queuing Protocol** (AMQP) é um protocolo aberto e padrão de camada de aplicação para middleware orientado a mensagens. As características definidoras do AMQP são orientação a mensagens, enfileiramento, roteamento (incluindo point-to-point e publish-and-subscribe), confiabilidade e segurança.

O AMQP é um protocolo binário de camada de aplicação, projetado para suportar eficientemente uma ampla variedade de aplicações de mensagens e padrões de comunicação. Ele fornece comunicação controlada por fluxo e orientada a mensagens com garantias de entrega de mensagens, como at-most-once (onde cada mensagem é entregue uma vez ou nunca), at-least-once (onde cada mensagem certamente é entregue, mas pode sê-lo várias vezes) e exactly-once (onde a mensagem sempre chegará e o fará apenas uma vez), além de autenticação e/ou criptografia baseadas em SASL e/ou TLS. Ele presume um protocolo de camada de transporte confiável subjacente, como o Transmission Control Protocol (TCP).

Features

AMQP pode ser utilizado em qualquer situação em que haja necessidade de entrega de mensagens segura e de alta qualidade entre cliente e broker.

O AMQP fornece os seguintes recursos:

- Monitorar e compartilhar atualizações.
- Garantir uma resposta rápida do servidor às requisições e a transmissão de tarefas demoradas para processamento posterior.
- Distribui mensagens a múltiplos destinatários.
- Conexão de clientes offline para recuperação adicional de dados.
- Aumenta a confiabilidade e a operação suave das aplicações.
- Confiabilidade da entrega de mensagens.
- Entrega de mensagens em alta velocidade.
- Aceitação de Mensagem.

Componentes

TsgcWSPClient_AMQP: é o componente cliente que implementa o protocolo **AMQP 0.9.1**.

Most common uses

- **Connection**
 - [Client AMQP Connect](#)
 - [Client AMQP Disconnect](#)
- **Commands**
 - [AMQP Channels](#)
 - [AMQP Exchanges](#)
 - [AMQP Queues](#)
 - [AMQP Publish Messages](#)
 - [AMQP Consume Messages](#) (Assíncrono)
 - [AMQP Get Messages](#) (Síncrono)
 - [AMQP QoS](#)
 - [AMQP Transactions](#)

TsgcWSPClient_AMQP

AMQP 0.

Introdução

O cliente **TsgcWSPClient_AMQP** implementa o protocolo completo **AMQP 0.9.1** seguindo a especificação OASIS. O cliente suporta conexões Plain TCP e WebSocket; conexões TLS (seguras) também são suportadas.

Conexão

O protocolo AMQP 0.9.1 define o conceito de canais, que permite compartilhar uma única conexão de socket com vários canais virtuais; o cliente implementa uma thread interna que lê os bytes recebidos e despacha cada mensagem para o canal correto (que já é executado em sua própria thread); então, se você estiver executando uma conexão AMQP com 5 canais, o cliente executará 6 threads (5 threads que tratam os dados de cada canal e 1 thread que trata os dados da conexão).

Antes de conectar a um servidor AMQP, configure as seguintes propriedades do protocolo AMQP

- **AMQPOptions.Locale:** é o locale de mensagem a utilizar; é um valor negociado, então pode mudar em comparação com os locales suportados pelo servidor. O valor padrão é "en_US".
- **AMQPOptions.MaxChannels:** é o número máximo de canais que podem ser abertos, é um valor negociado, portanto pode mudar quando comparado com a configuração do servidor. O valor padrão é 65535.
- **AMQPOptions.MaxFrameSize:** é o tamanho máximo em bytes do frame AMQP, é um valor negociado, portanto pode mudar em comparação com a configuração do servidor. O valor padrão é 2147483647.
- **AMQPOptions.VirtualHost:** é o nome do virtual host. O valor padrão é "/"

O HeartBeat AMQP também pode ser configurado antes de conectar ao servidor; você pode habilitar ou desabilitar o uso de heartbeats.

- **HeartBeat.Enabled:** defina como true se o cliente suportar HeartBeats.
- **HeartBeat.Interval:** o intervalo desejado em segundos.

Uma vez que o cliente AMQP tenha sido configurado, anexe-o a um [TsgcWebSocketClient](#) e agora você pode configurar as propriedades de conexão do servidor para conectar ao Servidor AMQP.

Defina o valor da propriedade **Specifications.RFC6455** como false se estiver usando uma conexão TCP Plain em vez de uma conexão WebSocket.

```
TsgcWSPClient_AMQP *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;

TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
oAMQP->Client = oClient;
oClient->Specifications->RFC6455 = false;
oClient->Host = "www.esegece.com";
oClient->Port = 5672;
oClient->Active = true;
```

Canais

Depois que o cliente AMQP se conectou, ele pode abrir o primeiro canal.

```
oAMQP->OpenChannel("channel_name");
```

Exchanges

Quando um Channel é aberto, o cliente pode declarar novos exchanges, verificar se eles existem... utilize o método **DeclareExchange** para declarar um novo exchange.

```
oAMQP->DeclareExchange("channel_name", "exchange_name");
```

Queues

Quando um Channel é aberto, o cliente pode declarar novas filas, verificar se elas existem... utilize o método **DeclareQueue** para declarar uma nova Fila. As filas não são fornecidas por padrão pelo servidor (ao contrário dos exchanges), então é sempre necessário declarar uma nova fila (a menos que uma fila já tenha sido criada por outro cliente).

```
oAMQP->DeclareQueue("channel_name", "queue_name");
```

Binding Queues

Uma vez configurados os Exchanges e as Queues, talvez você precise vincular queues a exchanges, dessa forma as exchanges podem saber quais mensagens serão despachadas para as queues.

Os Servidores AMQP vinculam automaticamente as filas ao exchange "direct" usando o nome da fila como routing key. Isto permite que você envie uma mensagem a uma fila específica sem a necessidade de declarar um binding (basta chamar o método **PublishMessage** e passar o argumento **Exchange** como valor vazio e o nome da fila no argumento **RoutingKey**).

```
oAMQP->BindQueue("channel_name", "queue_name", "exchange_name", "routing_key");
```

Enviar mensagens

Chame o método **PublishMessage** para publicar uma nova mensagem AMQP. O método permite que você publique uma mensagem **String** ou **TStream**.

```
oAMQP->PublishMessage("channel_name", "exchange_name", "routing_key", "Hello from sgcWebSockets!!!");
```

Receive Messages

O AMQP permite que você receba as mensagens em 2 modos:

- **Requisição pelo Cliente:** utilizando o método **GetMessage**. Se não houver mensagens na fila, o evento **OnAMQPBasicGetEmpty** será chamado.
- **Enviado pelo Servidor:** utilizando o método **Consume**.

Requisição pelo cliente

```
oAMQP->GetMessage("channel_name", "queue_name");

void OnAMQPGetOk(TObject *Sender, const string aChannel,
    const TsgcAMQPFramePayload_Method_BasicGetOk *aGetOk, const TsgcAMQPMessageContent *aContent)
{
```

```
    DoLog("#AMQP_basic_GetOk: " + aChannel + " " + IntToStr(aGetOk->MessageCount) + " " + aContent->Body->AsStr
  }
```

Enviado Pelo Servidor

```
oAMQP->Consume("channel_name", "queue_name");

void OnAMQPGetOk(TObject *Sender, const string aChannel,
  const TsgcAMQPFramePayload_Method_BasicGetOk *aGetOk, const TsgcAMQPMessageContent *aContent)
{
  DoLog("#AMQP_basic_GetOk: " + aChannel + " " + IntToStr(aGetOk->MessageCount) + " " + aContent->Body->AsStr
}
```

Referência

- Properties — 6 propriedades disponíveis no componente.
- Métodos — 41 métodos públicos.
- Eventos — 25 eventos que você pode tratar.

Connection | Client AMQP Connect

Para se conectar a um Servidor AMQP, você deve criar primeiro um [TsgcWebSocketClient](#) e um [TsgcWSPClient_AMQP](#). Em seguida, você deve anexar o Componente AMQP ao cliente WebSocket.

Uso básico

Conecte-se a um servidor AMQP sem autenticação. Defina os valores da propriedade AMQPOptions, o virtual host e, em seguida, defina no TsgcWebSocketClient o Host e a Port do servidor.

Se você está usando uma conexão TCP Plain, defina a propriedade Specifications.RFC6455 do TsgcWebSocketClient como 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;
```

Autenticação

Se o servidor exigir autenticação, use o evento **OnAMQPAuthentication** para seleccionar o mecanismo de Autenticação (se necessário) e definir o User / 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;

void OnAMQPAuthentication(TObject *Sender, TsgcAMQPAuthentications *aMechanisms, TsgcAMQPAuthentication &Mechanism,
String &User, String &Password)
{
    User = "user_value";
    Password = "password_value";
}
```

Connection | Client AMQP Disconnect

O cliente pode desconectar uma conexão ativa atual utilizando os seguintes métodos:

Enviando um Motivo de Fechamento

O cliente AMQP pode informar ao servidor que a conexão será fechada e fornecer informações sobre o motivo pelo qual está fechando a conexão. Utilize o método `Close` para solicitar o fechamento da conexão ao servidor.

```
oAMQP.Close(541, "Internal Error");
```

Fechando a Conexão do Socket

Basta definir a propriedade `Active` de `TsgcWebSocketClient` como `False`. Você pode ler mais sobre [fechamento de conexões](#).

Commands | AMQP Channels

AMQP é um protocolo multicanal. Os canais fornecem uma maneira de multiplexar uma conexão TCP/IP pesada em várias conexões leves. Isso torna o protocolo mais "amigável a firewall", já que o uso de portas é previsível. Também significa que traffic shaping e outros recursos de QoS de rede podem ser facilmente empregados.

Cada canal executa em sua própria thread, portanto, toda vez que uma nova mensagem é recebida, primeiro o cliente identifica o canal e enfileira a mensagem em uma fila que é processada pela thread do canal.

O ciclo de vida do canal é este:

1. O cliente abre um novo canal (Open).
2. O servidor confirma que o novo canal está pronto (Open-Ok).
3. O cliente e o servidor utilizam o canal conforme desejado.
4. Um peer (cliente ou servidor) fecha o canal (Close).
5. O outro peer faz o handshake do fechamento do canal (Close-Ok).

Abrir Channel

Para criar um novo canal, basta chamar o método **OpenChannel** e passar o nome do canal como argumento. O evento **OnAMQPChannelOpen** é gerado como confirmação enviada pelo servidor de que o canal foi aberto.

```
AMQP->OpenChannel("channel_name");

private void OnAMQPChannelOpen(TObject *Sender, const string aChannel)
{
    DoLog("#AMQP_channel_open: " + aChannel);
}
```

Uma chamada síncrona também pode ser feita chamando o método **OpenChannelEx**, este método retorna true se o canal foi aberto e false se nenhuma confirmação do servidor chegou.

```
if AMQP->OpenChannelEx("channel_name")
{
    DoLog("#AMQP_channel_open channel_name");
}
else
{
    DoLog("#AMQP_channel_open_error");
}
```

Fechar Canal

Para fechar um canal existente, chame o método **CloseChannel** e passe o nome do canal como argumento. O evento **OnAMQPChannelClose** será chamado quando o cliente receber uma confirmação de que o canal foi fechado.

Uma chamada Síncrona pode ser feita chamando o método **CloseChannelEx**, este método retorna true se o canal foi fechado e false se nenhuma confirmação do servidor chegou.

Channel Flow

O controle de fluxo é um procedimento de emergência usado para interromper o fluxo de mensagens de um peer. Funciona da mesma forma entre cliente e servidor e é implementado pelos comandos **EnableChannel** / **DisableChannel**. O controle de fluxo é o único mecanismo que pode parar um publicador que produz em excesso.

Para Desabilitar o Flow de um canal, chame o método **DisableChannel**; o evento **OnAMQPChannelFlow** será chamado quando o cliente receber uma confirmação de que o flow do canal foi desabilitado.

O mesmo se aplica ao habilitar o fluxo de um canal, chame o método **EnableChannel**, o evento **OnAMQPChannelFlow** será chamado quando o cliente receber uma confirmação de que o fluxo do canal foi habilitado.

Requisições síncronas estão disponíveis através das funções **EnableChannelEx** e **DisableChannelEx**.

Commands | AMQP Exchanges

A classe `exchange` permite que uma aplicação gerencie exchanges no servidor. Esta classe permite que a aplicação programe seu próprio roteamento (em vez de depender de alguma interface de configuração). Observação: A maioria das aplicações não precisa desse nível de sofisticação, e middleware legado dificilmente conseguirá suportar essa semântica.

O ciclo de vida do exchange é:

1. O cliente pede ao servidor para garantir que o exchange existe (Declare). O cliente pode refinar isto em "crie o exchange se ele não existir" ou "avise-me, mas não o crie, se ele não existir".
2. O cliente publica mensagens no exchange.
3. O cliente pode optar por excluir o exchange (Delete).

Declare Exchange

Este método cria um novo exchange ou verifica se um Exchange já existe. O método tem os seguintes argumentos:

- **ChannelName:** é o nome do canal (deve estar aberto antes de chamar este método).
- **ExchangeName:** é o nome do exchange, não deve ter mais de 255 caracteres e não pode começar com "amq." (exceto se o parâmetro `passive` for `true`).
- **ExchangeType:** é o tipo de exchange; todos os servidores AMQP suportam os exchanges "direct" e "fanout". Consulte a documentação do servidor para saber quais tipos de exchange são suportados.
- **Passive:** se `passive` for `true`, o servidor apenas verifica se o exchange já está declarado. Se `passive` for `false` e o exchange não existir, o servidor criará um novo.
- **Durable:** se `true`, o exchange será recriado quando o servidor iniciar. Se `false`, o exchange será excluído quando o servidor parar.
- **AutoDelete:** se `true`, o exchange será excluído quando todas as filas tiverem sido desvinculadas.
- **Internal:** sempre `false`.
- **NoWait:** se `true`, o servidor não envia uma confirmação ao cliente.
- **Arguments:** string que contém argumentos personalizados, os valores devem ser passados como uma string json, exemplo: `{"x-dead-letter-exchange":"my-dlx"}`.

Para declarar um novo Exchange, basta chamar o método **DeclareExchange** e passar o nome do canal, o nome do exchange e o tipo do exchange como argumentos. O evento **OnAMQPExchangeDeclare** é gerado como confirmação enviada pelo servidor de que o exchange foi declarado.

```
AMQP->DeclareExchange("channel_name", "exchange_name", "direct");

private void OnAMQPExchangeDeclare(TObject *Sender, const string aChannel, const string aExchange)
{
    DoLog("#AMQP_exchange_declare: [" + aChannel + "] " + aExchange);
}
```

Uma chamada síncrona também pode ser feita chamando o método **DeclareExchangeEx**; este método retorna `true` se o Exchange foi declarado e `false` se nenhuma confirmação do servidor chegou.

```
if AMQP->DeclareExchangeEx("channel_name", "exchange_name", "direct")
{
    DoLog("#AMQP_exchange_declare: [" + aChannel + "] " + aExchange);
}
else
{
    DoLog("#AMQP_exchange_declare_error");
}
```

Excluir Exchange

Este método é utilizado para excluir um Exchange existente. O método tem os seguintes argumentos:

- **ChannelName:** é o nome do canal (deve estar aberto antes de chamar este método).
- **ExchangeName:** é o nome do exchange, não deve ter mais de 255 caracteres e não pode começar com "amq." (exceto se o parâmetro `passive` for `true`).
- **IfUnused:** o servidor só exclui o exchange se não houver filas vinculadas a ele.
- **NoWait:** se `true`, o servidor não envia uma confirmação ao cliente.

Para Excluir um Exchange existente, chame o método **DeleteExchange** e passe o nome do canal e o nome do exchange como argumentos. O evento **OnAMQPExchangeDelete** é gerado como uma confirmação enviada pelo servidor de que o exchange foi excluído.

Uma chamada síncrona também pode ser feita chamando o método **DeleteExchangeEx**; este método retorna `true` se o Exchange foi excluído e `false` se nenhuma confirmação do servidor chegou.

Comandos | Filas AMQP

A classe `queue` permite que uma aplicação gerencie as message queues no servidor. Este é um passo básico em quase todas as aplicações que consomem mensagens, ao menos para verificar se uma message queue esperada está de fato presente.

O ciclo de vida de uma fila de mensagens durável é bastante simples:

1. O cliente afirma que a fila de mensagens existe (Declare, com o argumento "passive").
2. O servidor confirma que a fila de mensagens existe (Declare-Ok).
3. O cliente lê as mensagens da fila de mensagens.

O ciclo de vida de uma fila de mensagens temporária é mais interessante:

1. O cliente cria a message queue (Declare, geralmente sem nome de message queue para que o servidor atribua um nome). O servidor confirma (Declare-Ok).
2. O cliente inicia um consumidor na fila de mensagens. A funcionalidade precisa de um consumidor é definida pela classe `Basic`.
3. O cliente cancela o consumidor, seja explicitamente ou fechando o canal e/ou a conexão.
4. Quando o último consumidor desaparece da fila de mensagens, e após um time-out razoável, o servidor exclui a fila de mensagens.

O AMQP implementa o mecanismo de entrega para assinaturas de tópicos como filas de mensagens. Isso habilita estruturas interessantes onde uma assinatura pode ter sua carga balanceada (load balanced) entre um pool de assinantes cooperantes applications.

O ciclo de vida de uma assinatura envolve uma etapa extra de bind:

1. O cliente cria a fila de mensagens (Declare), e o servidor confirma (Declare-Ok).
2. O cliente vincula a message queue a um topic exchange (Bind) e o servidor confirma (Bind-Ok).
3. O cliente usa a fila de mensagens como nos exemplos anteriores.

Declare Queue

Este método cria uma nova fila ou verifica se uma Fila já existe. O método tem os seguintes argumentos:

- **ChannelName:** é o nome do canal (deve estar aberto antes de chamar este método).
- **QueueName:** é o nome da fila, não deve ter mais de 255 caracteres e não deve começar com "amq." (exceto se o parâmetro `passive` for true).
- **Passive:** se `passive` for true, o servidor apenas verifica se a fila já está declarada. Se `passive` for false e a fila não existir, o servidor criará uma nova.
- **Durable:** se true, a fila será recriada quando o servidor iniciar. Se false, a fila será excluída quando o servidor parar.
- **Exclusive:** se true, significa que a queue é acessada apenas pela conexão atual.
- **AutoDelete:** se true, a fila será excluída quando todos os consumidores não usarem mais a fila.
- **NoWait:** se true, o servidor não envia uma confirmação ao cliente.
- **Arguments:** string que contém argumentos personalizados, os valores devem ser passados como uma string json, exemplo: `{"x-dead-letter-exchange":"my-dlx"}`.

Para Declarar uma nova Queue, basta chamar o método **DeclareQueue** e passar o nome do canal e o nome da queue como argumentos. O evento **OnAMQPQueueDeclare** é gerado como uma confirmação enviada pelo servidor de que o exchange foi declarado.

```
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));
}
```

Uma chamada síncrona também pode ser feita chamando o método **DeclareQueueEx**; este método retorna true se a Queue foi Declarada e false se nenhuma confirmação do servidor chegou.

```
if AMQP->DeclareQueueEx("channel_name", "queue_name")
{
    DoLog("#AMQP_queue_declare: [" + aChannel + "] " + aQueue);
}
else
{
    DoLog("#AMQP_queue_declare_error");
}
```

Delete Queue

Este método é utilizado para excluir uma Fila existente. O método tem os seguintes argumentos:

- **ChannelName:** é o nome do canal (deve estar aberto antes de chamar este método).
- **QueueName:** é o nome da fila, não deve ter mais de 255 caracteres e não deve começar com "amq." (exceto se o parâmetro passive for true).
- **IfUnused:** o servidor só exclui a fila se não houver nenhum consumer anexado a ela.
- **IfEmpty:** o servidor só exclui a fila se não houver mensagens.
- **NoWait:** se true, o servidor não envia uma confirmação ao cliente.

Para Excluir uma Queue existente, chame o método **DeleteQueue** e passe o nome do canal e o nome da queue como argumentos. O evento **OnAMQPQueueDelete** é gerado como uma confirmação enviada pelo servidor de que a queue foi excluída.

Uma chamada Síncrona também pode ser feita chamando o método **DeleteQueueEx**; este método retorna true se a Fila foi Excluída e false se nenhuma confirmação do servidor chegou.

Bind Queue

Este método é utilizado para vincular uma Queue a um Exchange. Os Exchanges utilizam os vínculos para saber quais queues serão usadas para rotear as mensagens.

Todos os AMQP Servers fazem binding automaticamente de todas as filas ao exchange padrão (é um exchange "direct" sem nome) utilizando o Queue Name como a routing key do binding. Isso permite que você envie uma mensagem a uma fila específica sem declarar um binding. Basta chamar o método **PublishMessage**, passar um valor vazio como Exchange Name e definir o **RoutingKey** com o valor do **Queue Name**.

O método tem os seguintes argumentos:

- **ChannelName:** é o nome do canal (deve estar aberto antes de chamar este método).
- **QueueName:** é o nome da fila, não deve ter mais de 255 caracteres e não deve começar com "amq." (exceto se o parâmetro passive for true).
- **ExchangeName:** é o nome do exchange, não deve ter mais de 255 caracteres e não pode começar com "amq." (exceto se o parâmetro passive for true).
- **RoutingKey:** é a routing key do binding.
- **NoWait:** se true, o servidor não envia uma confirmação ao cliente.
- **Arguments:** string que contém argumentos personalizados, os valores devem ser passados como uma string json, exemplo: {"x-dead-letter-exchange":"my-dlx"}.

Para vincular uma Queue a um Exchange, chame o método **BindQueue** e passe o nome do canal, o nome da queue, o exchange e a routing key como argumentos. O evento **OnAMQPQueueBind** é gerado como uma confirmação enviada pelo servidor de que a queue foi vinculada.

```
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);
}
```

Uma chamada síncrona também pode ser feita chamando o método **BindQueueEx**; este método retorna true se a Queue foi vinculada (Bound) e false se nenhuma confirmação do servidor chegou.

```
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

Este método exclui um binding de queue existente.

O método tem os seguintes argumentos:

- **ChannelName:** é o nome do canal (deve estar aberto antes de chamar este método).
- **QueueName:** é o nome da fila, não deve ter mais de 255 caracteres e não deve começar com "amq." (exceto se o parâmetro passive for true).
- **ExchangeName:** é o nome do exchange, não deve ter mais de 255 caracteres e não pode começar com "amq." (exceto se o parâmetro passive for true).
- **RoutingKey:** é a routing key do binding.

Para desvincular (UnBind) uma fila, basta chamar o método **UnBindQueue** e passar o nome do canal, o nome da fila, o exchange e a routing key como argumentos. O evento **OnAMQPQueueUnBind** é gerado como confirmação enviada pelo servidor de que a fila foi desvinculada.

Uma chamada Síncrona também pode ser feita chamando o método **UnBindQueueEx**, este método retorna true se a Fila foi desvinculada e false se nenhuma confirmação do servidor chegou.

Purge Queue

Este método purga todas as mensagens de uma fila. Todas as mensagens que foram enviadas, mas estão aguardando confirmação, não são afetadas.

O método tem os seguintes argumentos:

- **ChannelName:** é o nome do canal (deve estar aberto antes de chamar este método).
- **QueueName:** é o nome da fila, não deve ter mais de 255 caracteres e não deve começar com "amq." (exceto se o parâmetro passive for true).
- **NoWait:** se true, o servidor não envia uma confirmação ao cliente.

Para Purgar uma Queue, basta chamar o método **PurgeQueue** e passar o nome do canal e o nome da queue como argumentos. O evento **OnAMQPQueuePurge** é gerado como uma confirmação enviada pelo servidor de que a queue foi Purgada.

Uma chamada síncrona também pode ser feita chamando o método **PurgeQueueEx**; este método retorna true se a Queue foi Purgada e false se nenhuma confirmação do servidor chegou.

Comandos | Publicar Mensagens AMQP

Publicar Mensagens

O método `PublishMessages` é utilizado para enviar uma mensagem ao servidor AMQP.

Os Servidores AMQP vinculam automaticamente as filas ao exchange "direct" usando o nome da fila como routing key. Isto permite que você envie uma mensagem a uma fila específica sem a necessidade de declarar um binding (basta chamar o método `PublishMessage` e passar o argumento `Exchange` como valor vazio e o nome da fila no argumento `RoutingKey`).

O método tem os seguintes argumentos:

- **ChannelName:** é o nome do canal (deve estar aberto antes de chamar este método).
- **ExchangeName:** é o nome do exchange, não deve ter mais de 255 caracteres nem começar com "amq." (exceto se o parâmetro `passive` for true).
- **RoutingKey:** é o nome da routing key do binding.
- **Mandatory:** se true e a mensagem não puder ser roteada para nenhuma queue, a mensagem é retornada pelo servidor e o evento `OnAMQPBasicReturn` é disparado.
- **Immediate:** se true e a mensagem não puder ser roteada para nenhuma fila, a mensagem é retornada pelo servidor e o evento `OnAMQPBasicReturn` é disparado.

```
AMQP->PublishMessage("channel_name", "exchange_name", "routing_key", "Hello from sgcWebSockets!!!");

private void OnAMQPBasicReturn(TObject *Sender, const string aChannel,
    const TsgcAMQPFramePayload_Method_BasicReturn *aReturn,
    const TsgcAMQPMessageContent *aContent)
{
    DoLog("#AMQP_basic_return: " + aChannel + " " + IntToStr(aReturn->ReplyCode) + " " + aReturn->ReplyText + " " +
}
```

Confirmações de Publicação

A rede pode falhar ao publicar uma mensagem, a única maneira de garantir que uma mensagem não seja perdida é utilizando transações, então para cada mensagem(ns) **selecione a transação, envie a mensagem e faça commit**. A confirmação de uma transação bem-sucedida é recebida quando o evento `OnAMQPTransactionOk` é disparado.

AMQP Consume Messages

Os consumidores consomem de filas. Para consumir mensagens, deve haver uma fila. Quando um novo consumidor é adicionado, supondo que já existam mensagens prontas na fila, as entregas começarão imediatamente. A fila de destino pode estar vazia no momento do registro do consumidor. Nesse caso, as primeiras entregas ocorrerão quando novas mensagens forem enfileiradas.

Consumir mensagens é uma tarefa **assíncrona**, o que significa que toda vez que uma nova mensagem pode ser entregue à fila do consumidor, ela é enviada pelo servidor ao cliente automaticamente. Você pode ler um método alternativo em [Receber Mensagem de Forma Síncrona](#).

Consume

O método **Consume** cria um novo consumidor na fila, e toda vez que houver uma nova mensagem, ela será entregue automaticamente ao cliente consumidor.

O método tem os seguintes argumentos:

- **ChannelName:** é o nome do canal (deve estar aberto antes de chamar este método).
- **QueueName:** é o nome da queue, não deve ter mais de 255 caracteres e não começar com "amq." (exceto se o parâmetro `passive` for true).
- **ConsumerTag:** é o nome do consumidor e deve ser único. Se não for definido, o servidor cria um novo.
- **NoLocal:** se true, significa que o consumidor nunca consome mensagens publicadas no mesmo canal.
- **NoAck:** se true, significa que o servidor não espera uma confirmação para cada mensagem entregue.
- **Exclusive:** se true, impede que outros consumidores consumam mensagens desta fila.
- **NoWait:** se true, o servidor não enviará uma confirmação ao cliente.
- **Arguments:** string que contém argumentos personalizados, os valores devem ser passados como uma string json, exemplo: {"x-dead-letter-exchange":"my-dlx"}.

As mensagens são entregues no evento **OnAMQPBasicDeliver**.

```
AMQP->Consume("channel_name", "queue_name", "consumer_tag");

void OnAMQPBasicDeliver(TObject *Sender, const string aChannel,
    const TsgcAMQPFramePayload_Method_BasicDeliver *aDeliver,
    const TsgcAMQPMessageContent *aContent)
{
    DoLog("#AMQP_basic_deliver: " + aChannel + " " + aDeliver->ConsumerTag + " " +
        " " + aContent->Body->AsString);
}
```

Uma chamada Síncrona pode ser feita apenas chamando o método **ConsumeEx**; este método retorna true se o Consumer foi criado e false se nenhuma confirmação do servidor chegou.

Cancel Consume

Este método é usado para Cancelar uma fila de consumidor existente.

O método tem os seguintes argumentos:

- **ChannelName:** é o nome do canal (deve estar aberto antes de chamar este método).
- **ConsumerTag:** é o nome do consumidor.
- **NoWait:** se true, o servidor não enviará uma confirmação ao cliente.

```
AMQP->CancelConsume("channel_name", "consumer_tag");

private void OnAMQPBasicCancelConsume(TObject *Sender, const string aChannel, const string aConsumerTag)
```

```
{  
  DoLog("#AMQP_basic_cancel_consume: " + aChannel + " " + aConsumerTag);  
}
```

Uma chamada síncrona pode ser feita simplesmente chamando o método **CancelConsumeEx**; este método retorna true se o Consumer foi cancelado e false se nenhuma confirmação do servidor chegou.

Comandos | Obter Mensagens AMQP

Obter mensagens é uma tarefa **Síncrona**, o que significa que é o cliente que pergunta ao servidor se há mensagens na fila. Você pode ler um método alternativo para [Receber Mensagem de Forma Assíncrona](#).

Get Message

O método **GetMessage** envia uma requisição ao servidor AMQP perguntando se há mensagens disponíveis em uma fila. Se houver mensagens, elas serão despachadas no evento **OnAMQPBasicGetOk** e, se a fila estiver vazia, o evento **OnAMQPBasicGetEmpty** será chamado.

O método tem os seguintes argumentos:

- **ChannelName:** é o nome do canal (deve estar aberto antes de chamar este método).
- **QueueName:** é o nome da fila, não deve ter mais de 255 caracteres e não deve começar com "amq." (exceto se o parâmetro `passive` for true).
- **NoWait:** se true, o servidor não enviará uma confirmação ao cliente.

```
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);
}
```

Uma chamada Síncrona pode ser feita apenas chamando o método **GetMessageEx**; este método retorna true se a fila tiver mensagens disponíveis, caso contrário, o resultado será false.

Commands | AMQP QoS

O AMQP permite que você defina um nível de QoS para limitar o número de mensagens que o servidor envia ao cliente antes de aguardar a confirmação das mensagens.

Set QoS

O método **SetQoS** é utilizado para limitar o número de mensagens que o servidor envia ao cliente AMQP. O método tem os seguintes argumentos:

- **ChannelName:** é o nome do canal (deve estar aberto antes de chamar este método).
- **PrefetchSize:** é o tamanho da janela em bytes; o servidor não envia mensagens ao cliente se o tamanho total de todas as mensagens não confirmadas já enviadas mais a próxima mensagem a ser enviada for maior que o argumento **PrefetchSize**. Se o valor for zero, significa sem limite.
- **PrefetchCount:** é o número máximo de mensagens não confirmadas já enviadas e não confirmadas; se o número for maior, o servidor para de enviar mensagens ao cliente.
- **Global:** se true, o QoS se aplica a todos os consumidores existentes e novos da conexão. Se false, o QoS se aplica a todos os consumidores existentes e novos do canal.

A resposta do servidor é recebida no 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 + " " + IntToStr(aQoS->PrefetchSize) + " "
        + IntToStr(aQoS->PrefetchCount) + " " + BoolToStr(aQoS->Global));
}
```

Uma chamada Síncrona pode ser feita apenas chamando o método **SetQoSEx**; este método retorna true se a requisição foi processada, caso contrário, o resultado será false.

Commands | AMQP Transactions

O AMQP suporta dois tipos de transações:

1. Transações automáticas, nas quais cada mensagem publicada e confirmação é processada como uma transação autônoma.
2. Transações locais do servidor, em que o servidor armazenará em buffer as mensagens publicadas e as confirmações e fará o commit sob demanda do cliente.

A classe Transaction ("tx") dá às aplicações acesso ao segundo tipo, a saber, transações do servidor. A semântica desta classe é:

1. A aplicação solicita transações do servidor em cada canal onde ela deseja essas transações (Select).
2. A aplicação realiza o trabalho (Publish, Ack).
3. A aplicação confirma ou desfaz o trabalho (Commit, Roll-back).
4. A aplicação faz o trabalho, ad infinitum.

As transações cobrem conteúdos publicados e confirmações, não entregas. Assim, um rollback não reenfileira nem reentrega nenhuma mensagem, e um cliente tem o direito de confirmar essas mensagens em uma transação seguinte.

Os métodos de Transaction permitem que as operações de publish e ack sejam agrupadas em unidades de trabalho atômicas. A intenção é que todas as requisições de publish e ack emitidas dentro de uma transação sejam concluídas com sucesso ou nenhuma delas seja.

Iniciar Transação

O método **StartTransaction** inicia uma nova transação no servidor; o cliente utiliza este método pelo menos uma vez em um canal antes de utilizar os métodos Commit ou Rollback. O evento **OnAMQPTransactionOk** é gerado quando o servidor confirma o uso de transações.

```
AMQP->StartTransaction("channel_name");
```

Uma chamada síncrona pode ser feita simplesmente chamando o método **StartTransactionEx**; este método retorna true se a requisição foi processada, caso contrário o resultado será false.

Commit Transaction

Este método faz commit de todas as publicações de mensagens e confirmações realizadas na transação atual. Uma nova transação começa imediatamente após um commit. O evento **OnAMQPTransactionOk** é gerado quando o servidor reconhece o uso de transações.

```
AMQP->CommitTransaction("channel_name");
```

Uma chamada síncrona pode ser feita simplesmente chamando o método **CommitTransactionEx**; este método retorna true se a requisição foi processada, caso contrário o resultado será false.

Rollback Transaction

Este método abandona todas as publicações de mensagens e confirmações realizadas na transação atual. Uma nova transação inicia imediatamente após um rollback. Observe que mensagens não confirmadas não serão automaticamente reentregues pelo rollback; se isso for necessário, uma chamada explícita de recover deve ser emitida. O evento **OnAMQPTransactionOk** é gerado quando o servidor confirma o uso de transações.

```
AMQP->RollbackTransaction("channel_name");
```

Uma chamada Síncrona pode ser feita apenas chamando o método **RollbackTransactionEx**, este método retorna true se a requisição foi processada, caso contrário o resultado será false.

Protocolo Kafka

Apache Kafka é uma plataforma distribuída de event-streaming de código aberto usada para publicar, armazenar e consumir streams de registros. Os produtores escrevem mensagens em tópicos, o broker as armazena em logs particionados do tipo append-only, e os consumidores as leem de volta em ordem, individualmente ou como parte de um grupo de consumidores coordenado.

O Kafka utiliza um protocolo binário de camada de aplicação sobre TCP. Cada tópico é dividido em uma ou mais partições, cada registro em uma partição é identificado por um offset que aumenta monotonicamente e o broker retém os registros para que os consumidores possam ler em seu próprio ritmo e reproduzir o histórico quando necessário. Isso torna o Kafka bem adequado para mensageria de alta vazão, event sourcing, agregação de logs e processamento de streams.

Features

O cliente Kafka `sgcWebSockets` se comunica com o broker diretamente pelo protocolo nativo do Kafka, sem necessidade de proxy REST ou biblioteca externa.

- Publica (produz) registros em qualquer tópico e partição, com chave, valor e confirmações configuráveis.
- Consome registros com um grupo de consumidores gerenciado ou sem um grupo (lê todas as partições diretamente).
- Coordenação de grupo de consumidores: descoberta de coordenador, join/sync, atribuição de partições e commit/fetch de offset.
- Gerenciamento de offset: consulta os offsets earliest, latest e committed.
- Administração de tópicos: criar e excluir tópicos, solicitar metadados de cluster e listar consumer groups.
- Lê e escreve o formato de record batch v2 utilizado pelos brokers Kafka atuais.

Versões Suportadas

O cliente usa o formato de record batch v2 introduzido no **Apache Kafka 0.11** e foi testado contra o **Apache Kafka 3.x**. A porta padrão do broker é 9092.

Componentes

TsgcWSPClient_Kafka: é o componente cliente que implementa o protocolo **Apache Kafka**.

Most common uses

- **Connection**
 - [Client Kafka Connect](#)
 - [Client Kafka Disconnect](#)
- **Commands**
 - [Kafka Produce Messages](#)
 - [Kafka Consume Messages](#)
 - [Kafka Consumer Groups](#)
 - [Kafka Topics](#)
 - [Kafka Offsets](#)

TsgcWSPClient_Kafka

Cliente Apache Kafka.

Introdução

O cliente **TsgcWSPClient_Kafka** implementa o protocolo **Apache Kafka**, comunicando-se diretamente com o broker pelo protocolo binário nativo do Kafka em uma conexão TCP simples. Ele pode produzir e consumir registros, gerenciar consumer groups, administrar tópicos e consultar offsets, sem proxy REST ou biblioteca externa.

Conexão

O cliente Kafka é um componente de protocolo que roda sobre um [TsgcWebSocketClient](#). Como o Kafka não é um protocolo WebSocket, defina **Specifications.RFC6455** como false para que o cliente se conecte como um socket TCP puro. Em seguida, atribua o cliente ao componente Kafka e defina o **Host** e a **Port** do broker (padrão 9092). O comportamento do produtor e do consumidor é configurado através da propriedade **KafkaOptions**:

- **KafkaOptions.ClientId**: identifica sua aplicação para o broker.
- **KafkaOptions.Producer.Acks**: confirmações necessárias, kafkaAcksNone, kafkaAcksLeader ou kafkaAcksAll.
- **KafkaOptions.Producer.TimeoutMs**: por quanto tempo o broker aguarda as confirmações necessárias.
- **KafkaOptions.Consumer.GroupId**: o nome do grupo de consumidores. Deixe vazio para consumir sem um grupo.
- **KafkaOptions.Consumer.OffsetReset**: kafkaOffsetEarliest ou kafkaOffsetLatest, ponto em que um grupo totalmente novo começa a ler.

```
TsgcWSPClient_Kafka *oKafka = new TsgcWSPClient_Kafka();
oKafka->KafkaOptions->ClientId = "my-cbuilder-app";
oKafka->OnKafkaConnect = OnKafkaConnect;
oKafka->OnKafkaMessage = OnKafkaMessage;
oKafka->OnKafkaProduce = OnKafkaProduce;

TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
oKafka->Client = oClient;
oClient->Specifications->RFC6455 = false;
oClient->Host = "127.0.0.1";
oClient->Port = 9092;
oClient->Active = true;
```

Produzir mensagens

Chame o método **Produce** para publicar um registro. O resultado de cada publicação é reportado por meio do evento **OnKafkaProduce**, incluindo o offset em que o registro foi armazenado.

```
oKafka->KafkaOptions->Producer->Acks = kafkaAcksLeader;
oKafka->Produce("my-topic", "Hello Kafka", "key-1");
```

Consume Messages

Defina um **GroupId** e um **OffsetReset**, assine um ou mais tópicos e chame **Poll** para buscar os registros que estiverem disponíveis. Poll retorna uma lista TsgcKafkaMessages e também gera o evento OnKafkaMessage para cada registro. Chame Poll repetidamente (por exemplo, a partir de um timer) para receber continuamente.

```
oKafka->KafkaOptions->Consumer->GroupId = "my-group";
oKafka->KafkaOptions->Consumer->OffsetReset = kafkaOffsetEarliest;
String topics[1] = { "my-topic" };
oKafka->Subscribe(topics, 0);

TsgcKafkaMessages *oMessages = oKafka->Poll(1000);
```

```
try {  
    for (int i = 0; i < oMessages->Count; i++)  
        DoLog(oMessages->Items[i]->GetValueString());  
} __finally {  
    oMessages->Free();  
}
```

Referência

- Propriedades — 3 propriedades disponíveis no componente.
- Métodos — 18 métodos públicos.
- Eventos — 6 eventos que você pode tratar.

Connection | Client Kafka Connect

Para conectar a um broker Apache Kafka, você deve primeiro criar um [TsgcWebSocketClient](#) e um [TsgcWSPClient_Kafka](#). Em seguida, anexe o Componente Kafka ao Cliente WebSocket.

O Kafka usa uma conexão TCP simples, portanto você deve definir a propriedade `Specifications.RFC6455` do `TsgcWebSocketClient` como `false`. Defina o `Host` e a `Port` do broker (a porta Kafka padrão é 9092), opcionalmente defina `KafkaOptions.ClientId` e então defina `Active` como `true` para iniciar a conexão.

Quando a conexão com o broker é estabelecida, o evento **OnKafkaConnect** é gerado.

Uso básico

```
oKafka = new TsgcWSPClient_Kafka();
oKafka->KafkaOptions->ClientId = "sgc-client";

oClient = new TsgcWebSocketClient();
oKafka->Client = oClient;
oClient->Specifications->RFC6455 = false;
oClient->Host = "www.esegece.com";
oClient->Port = 9092;
oClient->Active = true;

void OnKafkaConnect(TObject *Sender)
{
    ShowMessage("Connected to Kafka broker");
}
```

Connection | Client Kafka Disconnect

Para desconectar do broker Apache Kafka, defina a propriedade Active do WebSocket Client como false. Isso fecha a conexão TCP subjacente com o broker.

Quando a conexão é fechada, o evento **OnKafkaDisconnect** é gerado.

Uso básico

```
oClient->Active = false;  
  
void OnKafkaDisconnect(TObject *Sender)  
{  
    ShowMessage("Disconnected from Kafka broker");  
}
```

Commands | Kafka Produce Messages

Utilize o método **Produce(topic, value, key, partition)** para publicar um registro em um tópico Kafka. Os argumentos key e partition são opcionais; quando nenhuma partição é definida, o broker seleciona uma (por exemplo, fazendo hash da chave). Chame **ProduceBytes** para publicar uma carga útil binária.

Configure a garantia de entrega com `KafkaOptions.Producer.Acks`: `kafkaAcksNone` (envia e esquece), `kafkaAcksLeader` (aguarda o líder da partição) ou `kafkaAcksAll` (aguarda todas as réplicas in-sync). Você também pode definir `Producer.TimeoutMs` e `Producer.Compression` (`kafkaCompressionNone` ou `kafkaCompressionGzip`).

Depois que um registro é enviado, o evento **OnKafkaProduce** reporta o topic, a partition, o offset e o código de erro do registro produzido.

Uso básico

```
oKafka->KafkaOptions->Producer->Acks = kafkaAcksAll;
oKafka->KafkaOptions->Producer->TimeoutMs = 5000;
oKafka->KafkaOptions->Producer->Compression = kafkaCompressionGzip;

oKafka->Produce("my-topic", "hello world", "key1");

void OnKafkaProduce(TObject *Sender, const String Topic, int Partition, __int64 Offset, int ErrorCode)
{
    if (ErrorCode == 0)
        ShowMessage("Produced to " + Topic + " at offset " + IntToStr(Offset));
    else
        ShowMessage("Produce error " + IntToStr(ErrorCode));
}
```

Commands | Kafka Consume Messages

Para consumir registros, defina `KafkaOptions.Consumer.GroupId` e `KafkaOptions.Consumer.OffsetReset` (`kafkaOffsetEarliest` ou `kafkaOffsetLatest`), depois chame **Subscribe([topic])** com um ou mais tópicos.

Após se inscrever, chame **Poll(timeoutMs)** para buscar registros. `Poll` retorna uma lista `TsgcKafkaMessages` e gera o evento **OnKafkaMessage** uma vez por registro. Normalmente, você chama `Poll` repetidamente, por exemplo a partir de um timer, para continuar consumindo novos registros conforme eles chegam.

Cada registro na lista é um `TsgcKafkaMessage`. Itere a lista utilizando `Count` e leia cada registro com `GetKeyString`, `GetValueString`, `Topic`, `Partition` e `Offset`.

Subscribe and Poll

```
oKafka->KafkaOptions->Consumer->GroupId = "my-group";
oKafka->KafkaOptions->Consumer->OffsetReset = kafkaOffsetEarliest;
oKafka->Subscribe(ARRAYOFCONST(("my-topic")));

// call repeatedly, e.g. from a TTimer::OnTimer
TsgcKafkaMessages *oMessages = oKafka->Poll(1000);
for (int i = 0; i < oMessages->Count; i++)
    ShowMessage(oMessages->Items[i]->GetKeyString() + " = " + oMessages->Items[i]->GetValueString());
```

OnKafkaMessage Event

Em vez de iterar a lista retornada por `Poll`, você pode tratar cada registro no evento `OnKafkaMessage`.

```
void OnKafkaMessage(TObject *Sender, TsgcKafkaMessage *Message)
{
    ShowMessage(Message->Topic + ": " + Message->GetKeyString() + " = " + Message->GetValueString());
}
```

Commands | Grupos de Consumidores Kafka

Um consumer group permite que múltiplos consumidores compartilhem o trabalho de leitura de um tópico. O broker rastreia os offsets confirmados por GroupId e atribui as partições do tópico aos membros do grupo, de modo que cada partição seja lida por apenas um membro por vez. Defina o grupo com `KafkaOptions.Consumer.GroupId`. Para um novo grupo sem offsets confirmados, `KafkaOptions.Consumer.OffsetReset` decide onde começar: `kafka.OffsetEarliest` lê desde o início da partição, `kafka.OffsetLatest` lê apenas registros novos. Uma vez que os offsets são confirmados, o consumo é retomado a partir da última posição confirmada.

Chame **CommitSync** para confirmar os offsets dos registros retornados pelo último Poll, de modo que o grupo retome a partir deles na próxima sessão. Se você deixar GroupId vazio, o cliente consome todas as partições diretamente sem entrar em um grupo e sem confirmar offsets ao broker.

Uso básico

```
oKafka->KafkaOptions->Consumer->GroupId = "orders-group";
oKafka->KafkaOptions->Consumer->OffsetReset = kafkaOffsetEarliest;
oKafka->Subscribe(ARRAYOFCONST(("orders")));

TsgcKafkaMessages *oMessages = oKafka->Poll(1000);
if (oMessages->Count > 0)
    oKafka->CommitSync();
```


Commands | Kafka Topics

O cliente Kafka pode administrar tópicos no broker. Utilize **CreateTopic(name, partitions, replication)** para criar um novo tópico com o número de partições e o fator de replicação informados, e **DeleteTopic(name)** para remover um tópico.

Utilize **GetMetadata([topics])** para recuperar informações do cluster e dos tópicos, como os brokers, os tópicos e seu layout de partições. Passe um array vazio para solicitar os metadados de todo o cluster.

Uso básico

```
// create a topic with 3 partitions and replication factor 1
oKafka->CreateTopic("my-topic", 3, 1);

// request metadata for the topic
oKafka->GetMetadata(ARRAYOFCONST(("my-topic")));

// delete the topic
oKafka->DeleteTopic("my-topic");
```

Commands | Kafka Offsets

O cliente Kafka pode consultar e definir offsets para um determinado tópico e partição. **GetEarliestOffset(topic, partition)** retorna o offset disponível mais antigo, **GetLatestOffset(topic, partition)** retorna o próximo offset a ser escrito, e **GetCommittedOffset(topic, partition)** retorna o último offset confirmado pelo consumer group. Os três retornam um `Int64`.

Utilize **CommitOffset(topic, partition, offset)** para confirmar um offset específico para o consumer group, de modo que o consumo seja retomado a partir dessa posição na próxima sessão.

Uso básico

```
__int64 vEarliest = oKafka->GetEarliestOffset("my-topic", 0);
__int64 vLatest = oKafka->GetLatestOffset("my-topic", 0);
__int64 vCommitted = oKafka->GetCommittedOffset("my-topic", 0);

// commit a specific offset for the consumer group
oKafka->CommitOffset("my-topic", 0, vLatest);
```

Protocol AMQP 1.0.0

O AMQP (Advanced Message Queuing Protocol) 1.0.0 é um protocolo de mensageria projetado para comunicação confiável e assíncrona entre sistemas distribuídos. Ele facilita a troca de mensagens entre aplicações ou componentes de forma desacoplada, permitindo que se comuniquem sem dependências diretas. Veja a seguir uma análise técnica de alguns aspectos-chave do AMQP 1.0.0:

- **Comunicação orientada a mensagens:** O AMQP 1.0.0 é centrado no conceito de mensagens. As mensagens podem carregar dados, instruções ou comandos e são as unidades fundamentais de comunicação.
- **Message Brokers:** O protocolo opera em um modelo de mensageria com broker. Os brokers, que podem ser servidores ou entidades intermediárias, gerenciam o roteamento e a entrega de mensagens entre produtores e consumidores.
- **Queues e Exchanges:** As Queues são entidades de armazenamento dentro do broker onde as mensagens são armazenadas temporariamente. Os Exchanges definem as regras para rotear mensagens dos produtores às filas com base em critérios como conteúdo da mensagem ou routing keys.
- **Addresses and Links:** Os Addresses identificam os destinos de mensagens dentro da infraestrutura de mensagens. Os Links são canais de comunicação entre um remetente (produtor) e um receptor (consumidor) associados a um endereço específico.
- **Sessions and Connections:** As sessões representam um canal lógico para comunicação, permitindo múltiplos fluxos de mensagens dentro de uma única conexão. As conexões gerenciam o link de comunicação geral entre as aplicações cliente e o message broker.
- **Security:** O AMQP 1.0.0 suporta vários mecanismos de segurança, incluindo autenticação e autorização, para garantir comunicação segura entre clientes e brokers.
- **Agnóstico de transporte:** O protocolo é projetado para ser agnóstico de transporte, o que significa que pode operar sobre diferentes transportes de rede, como TCP, TLS ou WebSockets, proporcionando flexibilidade na implantação.
- **Flow Control:** O AMQP 1.0.0 inclui mecanismos de controle de fluxo, permitindo que os consumidores indiquem sua capacidade de tratar mensagens de entrada a uma determinada taxa. Isso ajuda a evitar sobrecarregar os consumidores com um grande número de mensagens.
- **Tratamento de Erros:** O protocolo especifica mecanismos para tratar erros, incluindo confirmação e rejeição de mensagens, garantindo robustez e confiabilidade na entrega de mensagens.
- **SASL Authentication:** A Simple Authentication and Security Layer (SASL) é usada para autenticar e proteger conexões entre clientes e brokers.

No geral, o AMQP 1.0.0 fornece uma forma padronizada e interoperável para diferentes componentes de software e sistemas se comunicarem de maneira fracamente acoplada, tornando-o adequado para diversas aplicações distribuídas e de nível corporativo.

Componentes

TsgcWSPClient_AMQP1: é o componente cliente que implementa o protocolo **AMQP 1.0.0**.

Most common uses

- **Conexão**
 - [Client AMQP1 Connect](#)
 - [Client AMQP1 Disconnect](#)
 - [Client AMQP1 Idle Timeout Connection](#)
 - [Client AMQP1 Connection State](#)
 - [Client AMQP1 Authentication](#)
 - [Client AMQP1 Azure Service Bus](#)
- **Commands**
 - [AMQP1 Sessions](#)
 - [AMQP1 Links](#)
 - [AMQP1 Sender Links](#)
 - [AMQP1 Receiver Links](#)
 - [AMQP1 Send Message](#)
 - [AMQP1 Read Message](#)

TsgcWSClient_AMQP1

O AMQP 1.

Introdução

O cliente **TsgcWSClient_AMQP1** implementa o protocolo **AMQP 1.0.0** seguindo a especificação OASIS. O cliente suporta conexões TCP Puro e WebSocket; conexões TLS (seguras) também são suportadas.

Configuração

O cliente AMQP 1.0.0 tem a propriedade `AMQPOptions`, onde você pode configurar a conexão.

- **ChannelMax:** O valor `channel-max` é o número de canal mais alto que pode ser utilizado na conexão. Este valor mais um é o número máximo de sessões que podem estar simultaneamente ativas na conexão
- **ContainerId:** (opcional) é o nome do container de origem, identifica de forma única a conexão no servidor.
- **CreditSize:** tamanho padrão do fluxo de crédito.
- **IdleTimeout:** O timeout é acionado por um peer local quando nenhum frame é recebido após um valor de limiar ser excedido. O idle timeout é medido em milissegundos e começa a partir do momento em que o último frame é recebido.
- **MaxFrameSize:** o tamanho máximo de frame aceito.
- **MaxLinksPerSession:** o número máximo de links por sessão.
- **WindowSize:** o tamanho de janela padrão.

A AMQP Authentication deve ser configurada na propriedade `Authentication`.

- **AuthType:** tipo de autenticação
 - **amqp1authNone:** não configurado.
 - **amqp1authSASLAnonymous:** autenticação anônima
 - **amqp1authSASLPlain:** autenticação user/password. Este tipo de autenticação requer o preenchimento das seguintes propriedades:
 - Username
 - Password
 - **amqp1authSASLExternal:** autenticação externa

Conexão

A conexão começa com o cliente (geralmente uma aplicação ou serviço de mensagens) iniciando uma conexão TCP com o servidor (o message broker). O cliente se conecta à porta do servidor, normalmente 5672 para conexões não-TLS e 5671 para conexões protegidas por TLS. Uma vez que a conexão TCP é estabelecida, o cliente e o servidor negociam a versão do protocolo AMQP que usarão. O AMQP 1.0.0 suporta várias versões e, durante a negociação, ambas as partes concordam em usar a versão 1.0.0.

Após a negociação do protocolo, o cliente pode precisar autenticar-se no servidor, dependendo da configuração do servidor. Os mecanismos de autenticação podem incluir mecanismos SASL (Simple Authentication and Security Layer) como PLAIN, EXTERNAL ou outros suportados pelo servidor.

Example: conecta ao servidor AMQP escutando na porta segura 5671 e usando credenciais SASL

```
// Creating AMQP client
oAMQP = new TsgcWSClient_AMQP1(this);
// Setting AMQP authentication options
```

```
oAMQP->AMQPOptions->Authentication->AuthType = amqp1authSASLPlain;
oAMQP->AMQPOptions->Authentication->Username = L"sgc";
oAMQP->AMQPOptions->Authentication->Password = L"sgc";
// Creating WebSocket client
oClient = new TsgcWebSocketClient(this);
// Setting WebSocket specifications
oClient->Specifications->RFC6455 = false;
// Setting WebSocket client properties
oClient->Host = L"www.esegece.com";
oClient->Port = 5671;
oClient->TLS = true;
// Assigning WebSocket client to AMQP client
oAMQP->Client = oClient;
// Activating WebSocket client
oClient->Active = true;
```

Sessions

Uma vez autenticado, o cliente abre uma sessão AMQP. Uma sessão é um contexto lógico para comunicação entre o cliente e o servidor. As sessões são utilizadas para agrupar operações de mensageria relacionadas. Utilize o método **CreateSession** para criar uma nova sessão; o método permite que você defina o nome da sessão ou deixe vazio, e o componente atribuirá um automaticamente.

Se a sessão foi criada com sucesso, o evento **OnAMQPSessionOpen** será disparado com os detalhes da sessão.

```
oAMQP->CreateSession("MySession");
oAMQP->OnAMQPSessionOpen = AMQP1AMQPSessionOpen;
void __fastcall TMyClass::AMQP1AMQPSessionOpen(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Fr
{
    ShowMessage("#session-open: " + aSession->Id);
}
```

Links

Dentro de uma sessão, o cliente cria links para se comunicar com entidades específicas como queues, topics ou outros recursos fornecidos pelo servidor. Os links são canais de comunicação bidirecionais utilizados para enviar e receber mensagens.

O componente pode trabalhar como nó remetente e receptor. Permite criar qualquer número de links para cada sessão, até o limite definido na propriedade **MaxLinksPerSession**.

Sender Links

Para criar um novo sender link, use o método **CreateSenderLink** e passe o nome da sessão e, opcionalmente, o nome do sender link. Se o link for criado com sucesso, o evento **OnAMQPLinkOpen** é gerado.

```
oAMQP->CreateSenderLink("MySession", "MySenderLink");
oAMQP->OnAMQPLinkOpen = AMQP1AMQPLinkOpen;
void __fastcall TMyForm::AMQP1AMQPLinkOpen(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *
{
    ShowMessage("#link-open: " + aLink->Name);
}
```

Receiver Links

Para criar um novo receiver link, utilize o método **CreateReceiverLink** e passe o nome da sessão e, opcionalmente, o nome do receiver link. Se o link for criado com sucesso, o evento **OnAMQPLinkOpen** é disparado.

```
oAMQP->CreateSenderLink("MySession", "MySenderLink");
oAMQP->OnAMQPLinkOpen = AMQP1AMQPLinkOpen;
void __fastcall TMyForm::AMQP1AMQPLinkOpen(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *
{
    ShowMessage("#link-open: " + aLink->Name);
}
```

Enviando mensagens

Com a sessão estabelecida e os links criados, o cliente pode começar a realizar operações de mensagem, como enviar mensagens a um destino. Utilize o método **SendMessage** para enviar uma mensagem utilizando um sender link.

```
oAMQP->SendMessage("MySession", "MySenderLink", "My first AMQP Message");
```

Recebendo mensagens

Por padrão, os Receiver Links são criados em **modo Automatic**, o que significa que toda vez que uma nova mensagem chega, ela será entregue ao cliente.

Se os Receiver Links foram criados em **modo manual**, use o Método Sync **GetMessage** para buscar e aguardar até que uma nova mensagem chegue.

Nos modos Automatic e Manual, toda vez que uma nova mensagem chega, o evento **OnAMQPMessage** é disparado.

```
void __fastcall OnAMQPMessageEvent(TObject *Sender, TsgcAMQP1Session *const aSession,
    TsgcAMQP1ReceiverLink *const aLink, TsgcAMQP1Message *const aMessage,
    TsgcAMQP1MessageDeliveryState &DeliveryState)
{
    ShowMessage(aMessage->ApplicationData->AMQPValue->Value);
}
```

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Métodos — 12 métodos públicos.
- Eventos — 14 eventos que você pode manipular.

Connection | Client AMQP1 Connect

Para conectar a um servidor AMQP, você deve criar primeiro um [TsgcWebSocketClient](#) e um `TsgcWSPClient_AMQP1`. Em seguida, você deve anexar o componente AMQP1 ao WebSocket Client.

Após uma conexão bem-sucedida, o evento `OnAMQPConnect` é disparado.

Uso básico

Conecta a um servidor AMQP 1.0.0 sem autenticação. Defina os valores da propriedade `AMQPOptions`, o virtual host e então defina no `TsgcWebSocketClient` o Host e a Port do servidor.

Se você está usando uma conexão TCP Plain, defina a propriedade `Specifications.RFC6455` do `TsgcWebSocketClient` como `false`.

```
// Creating AMQP client
oAMQP = new TsgcWSPClient_AMQP1(this);
// Creating WebSocket client
oClient = new TsgcWebSocketClient(this);
// Setting WebSocket specifications
oClient->Specifications->RFC6455 = false;
// Setting WebSocket client properties
oClient->Host = L"amqp_host_address";
oClient->Port = 5672;
// Assigning WebSocket client to AMQP client
oAMQP->Client = oClient;
// Activating WebSocket client
oClient->Active = true;
```

Autenticação

Se o servidor requer autenticação, utilize as propriedades `AMQP:Authentication` para definir os valores de `Username/Password` e defina `AuthType` como o valor `"amqp1authSASLPlain"`.

```
// Creating AMQP client
oAMQP = new TsgcWSPClient_AMQP1(this);
// Setting AMQP authentication options
oAMQP->AMQPOptions->Authentication->AuthType = amqp1authSASLPlain;
oAMQP->AMQPOptions->Authentication->Username = L"sgc";
oAMQP->AMQPOptions->Authentication->Password = L"sgc";
// Creating WebSocket client
oClient = new TsgcWebSocketClient(this);
// Setting WebSocket specifications
oClient->Specifications->RFC6455 = false;
// Setting WebSocket client properties
oClient->Host = L"www.esegece.com";
oClient->Port = 5671;
oClient->TLS = true;
// Assigning WebSocket client to AMQP client
oAMQP->Client = oClient;
// Activating WebSocket client
oClient->Active = true;
```


Connection | Cliente AMQP1 Disconnect

O cliente pode desconectar uma conexão ativa atual utilizando os seguintes métodos:

Enviando um Motivo de Fechamento

O cliente AMQP pode informar ao servidor que a conexão será fechada e fornecer informações sobre o motivo pelo qual está fechando a conexão. Utilize o método `Close` para solicitar o fechamento da conexão ao servidor.

```
oAMQP.Close('invalid-frame', "The received frame has an invalid format.");
```

Await Close

Por padrão, o método **Close** é **Assíncrono**, então, após chamar o método, o código continua. Se você quiser **aguardar até que o método Close seja concluído** e a confirmação enviada pelo servidor seja recebida, defina a propriedade **Await** como `True` no parâmetro `Options`.

```
void Close(const System::UnicodeString aCondition, const System::UnicodeString aDescription)
{
    TsgcAMQP1MethodOptions_Close *oOptions = new TsgcAMQP1MethodOptions_Close();
    try
    {
        oOptions->ErrorCondition = aCondition;
        oOptions->ErrorDescription = aDescription;
        oOptions->Await = true;
        AMQP1->Close(oOptions);
    }
    __finally
    {
        delete oOptions;
    }
}
```

Fechando a Conexão do Socket

Basta definir a propriedade `Active` de `TsgcWebSocketClient` como `False`. Você pode ler mais sobre [fechamento de conexões](#).

Connection | Idle Timeout

As conexões estão sujeitas a um threshold de **idle timeout**. O timeout é acionado pelo cliente quando nenhum quadro é recebido do servidor após um valor de threshold ser excedido. O idle timeout é medido em milissegundos e começa a partir do momento em que o último quadro é recebido. Se o threshold for excedido, o componente envia um Close Frame ao servidor. Se o servidor não responder após 10 segundos, o cliente fechará o socket TCP.

O Valor do Idle Timeout pode ser configurado na propriedade:

AMQPOptions.IdleTimeout

O valor definido nesta propriedade será enviado ao servidor ao abrir a conexão AMQP. Se o valor for maior que zero e menor que a metade do valor MaxInt, um timer interno será habilitado para verificar se o idle timeout não foi excedido.

Exemplo: definir um valor de IdleTimeout de 60 segundos

```
AMQPOptions.IdleTimeout = 60000
```

Connection | Estado da Conexão

O AMQP 1.0.0 define os seguintes estados de conexão:

- **amqp1csUnknown:** estado inicial.
- **amqp1csStart:** Neste estado, uma conexão existe, mas nada foi enviado ou recebido. Este é o estado em que uma implementação estaria imediatamente após realizar um socket connect ou socket accept.
- **amqp1csHeaderReceived:** neste estado, o cabeçalho da conexão foi recebido do peer, mas um cabeçalho de conexão ainda não foi enviado.
- **amqp1csHeaderSent:** Neste estado, o cabeçalho de conexão foi enviado ao peer, mas nenhum cabeçalho de conexão foi recebido.
- **amqp1csHeaderExchanged:** Neste estado, o connection header foi enviado ao peer e um connection header foi recebido do peer.
- **amqp1csOpenPipe:** Neste estado, tanto o cabeçalho da conexão quanto o frame open foram enviados, mas nada foi recebido.
- **amqp1csOpenClosePipe:** Neste estado, o cabeçalho da conexão, o frame open, qualquer tráfego de conexão pipelined e o frame close foram enviados, mas nada foi recebido.
- **amqp1csOpenReceived:** Neste estado, os cabeçalhos de conexão foram trocados. Um frame open foi recebido do peer, mas um frame open não foi enviado.
- **amqp1csOpenSent:** Neste estado os headers da conexão foram trocados. Um frame open foi enviado ao peer, mas nenhum frame open foi recebido ainda.
- **amqp1csClosePipe:** Neste estado, os cabeçalhos de conexão foram trocados. Um frame open, qualquer tráfego de conexão em pipeline e o frame close foram enviados, mas nenhum frame open foi ainda recebido do peer.
- **amqp1csOpened:** Neste estado, o connection header e o open frame foram ambos enviados e recebidos.
- **amqp1csCloseReceived:** Neste estado, um frame close foi recebido indicando que o peer iniciou um close AMQP. Nenhum frame adicional é esperado para chegar na conexão; no entanto, frames ainda podem ser enviados. Se desejado, uma implementação PODE fazer um TCP half-close neste ponto para encerrar o lado de leitura da conexão.
- **amqp1csCloseSent:** Neste estado, um frame de close foi enviado ao peer. É ilegal escrever qualquer outra coisa na conexão; no entanto, ainda poderia haver frames de entrada. Se desejado, uma implementação PODE fazer um half-close TCP neste ponto para encerrar o lado de escrita da conexão.
- **amqp1csDiscarding:** O estado DISCARDING é uma variante do estado CLOSE SENT em que o fechamento é acionado por um erro. Nesse caso, quaisquer frames de entrada na conexão DEVEM ser descartados silenciosamente até que o frame de fechamento do peer seja recebido.
- **amqp1csEnd:** Neste estado, é ilegal para qualquer um dos endpoints escrever qualquer coisa adicional na conexão. A conexão pode ser fechada e descartada com segurança.

O Cliente AMQP tem a propriedade **ConnectionState** onde você pode verificar em qual estado de conexão o componente cliente está.

Connection | AMQP1 Authentication

O componente possui os seguintes métodos de autenticação:

- **amqp1authNone:** não há método de autenticação a ser utilizado ao se conectar ao servidor.
- **amqp1authSASLAnonymous:** conecta como anônimo.
- **amqp1authSASLPlain:** o padrão, usa uma autenticação de usuário/senha.
- **amqp1authSASLExternal:** não suportado atualmente.

Autenticação SASL

A autenticação mais comum é usar o tipo **amqp1authSASLPlain**. Este tipo de autenticação pode ser habilitado no componente AMQP1, acessando a propriedade `AMQPOptions.Authentication`.

- **AuthType:** selecione `amqp1authSASLPlain`
- **Username:** o usuário a ser utilizado para Autenticação SASL.
- **Password:** o valor secreto a ser usado para a Autenticação SASL.

O resultado da SASL Authentication pode ser obtido por meio do evento **OnAMQPSASLAuthentication**.

```
void __fastcall OnAMQP1SASLAuthentication(System::TObject* Sender, TsgcAMQP1SaslCode aCode,
    const System::UnicodeString aDescription, bool &Handled)
{
    ShowMessage("#sasl-authentication: " + aDescription);
}
```

Connection | Azure MessageBus

O AMQP (Advanced Message Queuing Protocol) é um sistema de mensagens robusto projetado para facilitar a comunicação entre diversos containers em vários nós. Ele padroniza tanto o protocolo de transmissão de mensagens quanto o framework estrutural das próprias mensagens, garantindo uma comunicação consistente e confiável. Para se aprofundar nos fundamentos do AMQP, consulte nosso guia [Getting Started with AMQP](#).

O componente AMQP dentro da biblioteca eSeGeCe possibilita a integração contínua com os principais brokers de mensagens em nuvem, incluindo Amazon MQ e Azure Service Bus. Este guia foca no uso do componente AMQP para conectar ao Azure Service Bus, demonstrando como construir uma aplicação multi-tenant capaz de enviar e receber mensagens de forma eficiente.

O componente oferece uma implementação abrangente com suporte a recursos-chave como queues, tópicos e assinaturas, tornando-o uma escolha ideal para aplicações modernas de IoT e corporativas.

Configuração do Azure

Para começar, crie um **recurso Service Bus** dentro do Azure Portal. Uma vez que o recurso esteja estabelecido, certifique-se de anotar o **nome de domínio do recurso**, pois ele será essencial para a integração e configuração.

Depois que o **namespace** tiver sido criado com sucesso, você pode gerenciá-lo e monitorá-lo diretamente a partir da **visão geral do namespace** no Azure Portal. Essa interface centralizada fornece acesso a ferramentas e configurações de gerenciamento essenciais, possibilitando uma administração contínua do seu recurso do Service Bus.

Ao utilizar **SAS Authentication**, o **nome de usuário** é o **nome da SAS Policy** e a **senha** é a **chave primária ou secundária**.

```
// Create TCP client
TsgcWebSocketClient* oClient = new TsgcWebSocketClient(nullptr);
oClient->Specifications->RFC6455 = false;
oClient->Host = "esegece.servicebus.windows.net";
oClient->Port = 5671;
oClient->TLS = true;
// Create AMQP1 protocol client
TsgcWSClient_AMQP1* oAMQP1 = new TsgcWSClient_AMQP1(nullptr);
oAMQP1->Specifications->RFC6455 = false;
oAMQP1->AMQPOptions->Authentication->AuthType = amqp1authSASLPlain;
oAMQP1->AMQPOptions->Authentication->Username = "RootManageSharedAccessKey";
oAMQP1->AMQPOptions->Authentication->Password = "BhJ78+w8kMXhS/eE/nBy0cRzodx9tipbi+ASbAXIaH8=";
oAMQP1->Client = oClient;
// Connect to the server
oClient->Active = true;
```

Azure CBS Authentication

O Azure Service Bus implementa Claims-Based Security (CBS) sobre AMQP para autorizar remetentes e destinatários após o handshake SASL inicial. O cliente abre um link de gerenciamento para o nó **\$cbs** e envia uma requisição **put-token** contendo um token Shared Access Signature (SAS) ou um JSON Web Token (JWT) emitido pelo Microsoft Entra ID. Assim que o broker valida o token, a autorização é armazenada em cache durante seu tempo de vida e a aplicação pode prosseguir criando links de remetente e destinatário contra filas, tópicos ou assinaturas.

O cliente AMQP1 automatiza esse fluxo através de dois métodos auxiliares:

- **CreateAzureCbsSasToken** estabelece um par de links CBS de envio/recebimento, gera um token SAS para a entidade de destino e o publica em **\$cbs**. Utilize-o ao autenticar com uma política de acesso compartilhado.
- **CreateAzureCbsJWT** segue a mesma troca CBS, mas obtém um token de acesso do Microsoft Entra ID (Azure AD) usando o grant client-credentials antes de enviar o JWT para **\$cbs**.

Ambos os métodos requerem uma conexão AMQP ativa e aceitam os seguintes parâmetros:

- **aName:** Identificador para o par de links CBS criado internamente.
- **aNameSpace** e **aEntityName:** O namespace do Service Bus (sem o sufixo *.servicebus.windows.net*) e o caminho da queue, topic ou subscription utilizado para construir o audience do token.
- **aKeyName** / **aKeyValue:** Nome da shared access policy e chave para tokens SAS. O componente assina o token e o envia utilizando o token type *servicebus.windows.net:sastoken*.
- **aTenant**, **aApplicationId**, **aSecret:** directory ID do Microsoft Entra (Azure AD), application (client) ID e client secret utilizados para solicitar o JWT com o fluxo client credentials.
- **aListeningPort** (JWT): Porta HTTP local para o redirect OAuth 2.0 (o padrão é 8080 quando não fornecida).
- **aExpiration** e **aTimeout:** Tempo de vida do token emitido (em segundos) e o tempo máximo de espera (em milissegundos) para a negociação CBS.
- **aRaiseIfError:** Quando definido como *True*, o método gera uma exceção se a aquisição do token ou a resposta CBS falhar.

Os exemplos a seguir ilustram como autenticar com o CBS antes de enviar mensagens.

```
// Create TCP client
TsgcWebSocketClient* oClient = new TsgcWebSocketClient(nullptr);
oClient->Specifications->RFC6455 = false;
oClient->Host = "esegece.servicebus.windows.net";
oClient->Port = 5671;
oClient->TLS = true;
// Create AMQP1 protocol client
TsgcWSSClient_AMQP1* oAMQP1 = new TsgcWSSClient_AMQP1(nullptr);
oAMQP1->Specifications->RFC6455 = false;
oAMQP1->AMQPOptions->Authentication->AuthType = amqp1authSASLAnonymous;
oAMQP1->Client = oClient;
// Connect and publish SAS token through CBS
oClient->Active = true;
// wait the client is connected
oAMQP1->CreateAzureCbsSasToken("cbs", "esegece", "queue1",
    "RootManageSharedAccessKey",
    "BhJ78+w8kMXhS/eE/nBy0cRzodx9tipbi+ASbAXIaH8=",
    3600, 10000, true);
```

O próximo exemplo foca exclusivamente na autenticação Microsoft Entra ID (Azure AD) utilizando JWTs. Ele mostra como solicitar um token com o fluxo de credenciais de cliente e publicá-lo em **\$cbs** antes de criar links para enviar ou receber mensagens.

```
// Create TCP client
TsgcWebSocketClient* oClient = new TsgcWebSocketClient(nullptr);
oClient->Specifications->RFC6455 = false;
oClient->Host = "esegece.servicebus.windows.net";
oClient->Port = 5671;
oClient->TLS = true;
// Create AMQP1 protocol client
TsgcWSSClient_AMQP1* oAMQP1 = new TsgcWSSClient_AMQP1(nullptr);
oAMQP1->Specifications->RFC6455 = false;
oAMQP1->AMQPOptions->Authentication->AuthType = amqp1authSASLAnonymous;
oAMQP1->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);
```

Commands | AMQP1 Sessions

No contexto da especificação AMQP (Advanced Message Queuing Protocol) 1.0.0, uma sessão representa um contexto lógico para comunicação entre um cliente e um message broker. Veja a seguir um detalhamento do que envolve uma sessão AMQP 1.0.0:

- **Logical Context:** Uma sessão estabelece um contexto lógico para operações de mensagens entre um cliente AMQP (produtor ou consumidor) e um broker AMQP. Ela fornece uma forma de agrupar operações de mensagens relacionadas dentro de uma única conexão.
- **Communication Channel:** As sessões servem como canais de comunicação pelos quais as mensagens são enviadas e recebidas. Elas encapsulam a troca de mensagens, confirmações e mecanismos de controle de fluxo.
- **Transactional Boundaries:** As sessões definem limites transacionais para operações de mensagem. Elas permitem o agrupamento de múltiplos envios ou recebimentos de mensagens em uma única unidade atômica, garantindo que todas as operações dentro da sessão sejam processadas com sucesso ou que nenhuma seja processada.
- **Flow Control:** As sessões suportam mecanismos de controle de fluxo para regular a taxa na qual as mensagens são trocadas entre o cliente e o broker. O controle de fluxo ajuda a evitar a sobrecarga dos recursos de qualquer uma das partes, garantindo entrega de mensagens eficiente e confiável.
- **Lifetime Management:** As sessões têm um ciclo de vida que começa quando são criadas e termina quando são fechadas. Os clientes podem estabelecer múltiplas sessões dentro de uma única conexão para paralelizar o processamento de mensagens ou isolar streams de mensagens.
- **Alocação de Recursos:** As sessões podem ser associadas a recursos específicos, como filas, tópicos ou assinaturas dentro do broker. As mensagens enviadas ou recebidas dentro de uma sessão estão vinculadas a esses recursos, permitindo o roteamento e a entrega direcionados de mensagens.

Em resumo, uma sessão AMQP 1.0.0 fornece um contexto lógico para a troca de mensagens entre um cliente e um broker AMQP, facilitando a integridade transacional, o controle de fluxo e o gerenciamento de recursos dentro do sistema de mensagens. Ela define os limites dentro dos quais as operações de mensagens são realizadas e ajuda a garantir a troca eficiente e confiável de mensagens.

Abrir sessão

O método **CreateSession** cria uma nova sessão com o nome informado (ou, se vazio, cria com um nome aleatório); se a sessão já existir, uma exceção é gerada. O cliente permite que você crie múltiplas sessões utilizando a mesma conexão AMQP.

Uma vez que a sessão seja criada com sucesso, o evento **OnAMQPSessionOpen** é disparado.

```
oAMQP->CreateSession("MySession");
oAMQP->OnAMQPSessionOpen = OnAMQPSessionOpenEvent;
void __fastcall TMyClass::OnAMQPSessionOpenEvent(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Frame *aFrame)
{
    ShowMessage("#session-open: " + aSession->Id);
}
```

O método **CreateSession** retorna a classe **TsgcAMQP1Session** que contém as informações da sessão.

Await Open Session

Por padrão, o método **CreateSession** é **Assíncrono**, então, após chamar o método, o código continua. Se você quiser **aguardar até que o método CreateSession seja concluído** e a confirmação enviada pelo servidor seja recebida, defina a propriedade **Await** como True no parâmetro Options.

```
void OpenSession(const System::UnicodeString aSession)
{
    TsgcAMQP1MethodOptions_SessionOpen *oOptions = new TsgcAMQP1MethodOptions_SessionOpen();
    try
    {
        oOptions->Await = true;
        AMQP1->CreateSession(aSession, oOptions);
    }
    __finally
    {
        delete oOptions;
    }
}
```

Close Session

Para fechar uma sessão existente, utilize o método **CloseSession** passando o nome da sessão a ser fechada.

Uma vez que a sessão é fechada com sucesso, o evento **OnAQMPSessionClose** é disparado.

```
oAMQP->CloseSession("MySession");
oAMQP->OnAMQPSessionClose = OnAMQPSessionCloseEvent;
// OnAMQPSessionCloseEvent implementation
void __fastcall TMyClass::OnAMQPSessionCloseEvent(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Fr
{
    ShowMessage("#session-close: " + aSession->Id + " [" + IntToStr(aSession->Channel) + "] reason: " + aEnd->Err
}
```

Await Close Session

Por padrão, o método **CloseSession** é **Assíncrono**, portanto, após chamar o método, o código continua. Se você quiser **aguardar até que o método CloseSession seja concluído** e a confirmação enviada pelo servidor seja recebida, defina a propriedade **Await** como True no parâmetro Options.

```
void CloseSession(const System::UnicodeString aSession)
{
    TsgcAMQP1MethodOptions_SessionClose *oOptions = new TsgcAMQP1MethodOptions_SessionClose();
    try
    {
        oOptions->Await = true;
        AMQP1->CloseSession(aSession, oOptions);
    }
    __finally
    {
        delete oOptions;
    }
}
```


Comandos | AMQP1 Links

Na especificação AMQP (Advanced Message Queuing Protocol) 1.0.0, um link representa um canal de comunicação unidirecional entre um cliente AMQP e um message broker. Vamos nos aprofundar no que os links do AMQP 1.0.0 envolvem:

- **Canal de Comunicação:** Um link serve como um caminho através do qual as mensagens fluem entre um sender e um receiver AMQP. Ele permite a transmissão de mensagens em uma direção, seja do sender para o receiver ou vice-versa.
- **Fluxo Unidirecional:** Cada link é unidirecional, o que significa que as mensagens só podem trafegar em uma direção ao longo do link. Se for necessária comunicação bidirecional, dois links devem ser estabelecidos—um para cada direção.
- **Transferência de Mensagens:** As mensagens são transferidas pelos links de acordo com as regras do protocolo AMQP. Essas mensagens podem incluir payloads, propriedades de mensagem e metadados adicionais necessários para a comunicação.
- **Resource Binding:** Os links são associados a recursos específicos dentro do broker AMQP, como queues, topics ou exchanges. As mensagens enviadas ou recebidas por meio de um link são direcionadas a esses recursos ou originadas deles.
- **Controle de Fluxo:** Os links suportam mecanismos de controle de fluxo para regular a taxa na qual as mensagens são enviadas ou recebidas. O controle de fluxo garante que nem o remetente nem o receptor sejam sobrecarregados pelo volume de mensagens trocadas.
- **Lifetime Management:** Os links têm um ciclo de vida que começa quando são estabelecidos e termina quando são fechados. Eles podem ser criados dinamicamente conforme necessário e fechados quando não são mais requeridos.
- **Endereçamento:** Os links são identificados por endereços únicos que especificam os endpoints de origem e destino da comunicação. Esses endereços permitem que clientes e brokers identifiquem e estabeleçam conexões com os endpoints apropriados.
- **Limites Transacionais:** Os links definem limites transacionais para operações de mensagens. Eles permitem o agrupamento de múltiplos envios ou recebimentos de mensagens em uma única unidade atômica, garantindo consistência e confiabilidade na entrega de mensagens.

Em resumo, os links AMQP 1.0.0 fornecem um meio de comunicação unidirecional entre clientes e brokers AMQP, facilitando a transferência de mensagens enquanto suportam controle de fluxo, binding de recursos, endereçamento e integridade transacional dentro do sistema de mensagens. Eles formam os blocos de construção fundamentais da troca de mensagens no protocolo AMQP.

Existem 2 tipos de Links:

- **Sender Links:** esses links são utilizados para enviar mensagens.
- **Receiver Links:** esses links são usados para receber mensagens.

Toda vez que um novo link é criado ou excluído, os seguintes eventos são disparados:

- **OnAMQPLinkOpen:** este evento é acionado quando um novo link é criado. Utilize a propriedade `aLink.Mode` para verificar se o link está em modo receiver ou sender.
- **OnAMQPLinkClose:** este evento é disparado quando um link é fechado.

Commands | AMQP1 Sender Links

No protocolo AMQP 1.0.0, um **Sender Link** é um **canal de comunicação** estabelecido entre um cliente AMQP e um servidor AMQP com o propósito de **enviar mensagens**. Ele opera dentro do contexto de uma sessão AMQP, que representa um canal lógico para comunicação entre o cliente e o servidor.

Create Sender Link

Para **Criar um novo Sender Link**, chame o método **CreateSenderLink** que contém os seguintes parâmetros:

- **Session:** o nome da sessão à qual o sender link será anexado.
- **Name:** (opcional) o nome do sender link; se não for definido, um nome aleatório será atribuído automaticamente.
- **Target:** (opcional) você pode especificar o destino onde as mensagens devem ser recebidas no host remoto definindo o parâmetro "target". No entanto, em certos cenários, especificar o target pode não ser necessário. Nesses casos, fornecer uma string vazia será suficiente.
- **SndSettleMode:** (mixed por padrão) o AMQP oferece a capacidade de discutir garantias de entrega através do mecanismo de Message Settlement. Ao estabelecer um link, tanto o remetente quanto o destinatário discutem e concordam com um modo de settlement (um para cada função). Os remetentes operam dentro de um destes modos:
 - **amqp1ssmSettled:** A mensagem é considerada entregue e confirmada com sucesso assim que é enviada.
 - **amqp1ssmUnsettled:** A mensagem não é considerada settled até ser explicitamente aceita ou rejeitada pelo destinatário. Isso permite maior controle sobre o processamento e tratamento das mensagens.
 - **amqp1ssmMixed:** Uma combinação dos modos settled e unsettled pode ser utilizada dentro de uma única sessão AMQP. Utilize o parâmetro **MessageOptions** do método **SendMessage** para configurar se a mensagem é Settled ou não.

Quando o Sender Link tiver sido criado com sucesso, o evento **OnAMQPLinkOpen** será disparado.

```
oAMQP1->CreateSenderLink("MySession", "MySenderLink");
oAMQP1->OnAMQPLinkOpen = AMQP1AMQPLinkOpen;
void __fastcall TMyForm::AMQP1AMQPLinkOpen(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *const aLink)
{
    ShowMessage("#link-open: " + aLink->Name);
}
```

Await Create Sender Link

Por padrão, o método **CreateSenderLink** é **Assíncrono**, portanto, após chamar o método, o código continua. Se você quiser **aguardar até que o método CreateSenderLink seja concluído** e a confirmação enviada pelo servidor seja recebida, defina a propriedade **Await** como True no parâmetro Options.

```
void CreateSenderLink(const System::UnicodeString aSession, const System::UnicodeString aSender)
{
    TsgcAMQP1MethodOptions_CreateSenderLink *oOptions = new TsgcAMQP1MethodOptions_CreateSenderLink();
    try
    {
        oOptions->Await = true;
        AMQP1->CreateSenderLink(aSession, aSender, L"", oOptions);
    }
    __finally
    {
        delete oOptions;
    }
}
```

Enviando mensagens

Para enviar uma nova mensagem, chame o método `SendMessage`, que contém os seguintes parâmetros:

- **Session:** nome da sessão.
- **Link:** nome do link remetente.
- **Text:** o texto da mensagem de string.

```
oAMQP1->SendMessage("MySession", "MySenderLink", "My first AMQP Message");
```

Sending Messages Mixed Mode

Quando o Sender Link é criado em modo Mixed (o padrão), ao enviar uma mensagem, o usuário pode definir se quer que a mensagem seja **settled** ou **not**. Utilize o parâmetro **MessageOptions** para definir se a mensagem é settled ou não.

```
TsgcAMQP1MessageOptions* oMessageOptions = new TsgcAMQP1MessageOptions();
try
{
    oMessageOptions->Settled = true;
    oAMQP1->SendMessage("MySession", "MySenderLink", "MyMessage", "message-id", oMessageOptions);
}
__finally
{
    delete oMessageOptions;
}
```

Fechar Sender Link

Para fechar um Sender Link existente, chame o método **CloseLink** que contém os seguintes parâmetros:

- Session: nome da sessão que contém o link.
- Link: nome do sender link.
- Error: (opcional) aqui você pode definir o motivo pelo qual o link está sendo fechado.

Quando o Sender Link tiver sido fechado com sucesso, o evento **OnAMQPLinkClose** será disparado.

```
oAMQP->CloseLink("MySession", "MySenderLink");
void __fastcall TForm1::OnAMQPLinkCloseEvent(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *cc
{
    ShowMessage("#link-close: " + aLink->Name);
}
```

Aguardar Fechamento do Sender Link

Por padrão, o método **CloseLink** é **Assíncrono**, então após chamar o método, o código continua. Se você quiser **aguardar até que o método CloseLink seja concluído** e a confirmação enviada pelo servidor seja recebida, defina a propriedade **Await** como True no parâmetro Options.

```
void CloseSenderLink(const System::UnicodeString aSession, const System::UnicodeString aSenderLink)
{
    TsgcAMQP1MethodOptions_CloseLink *oOptions = new TsgcAMQP1MethodOptions_CloseLink();
    try
    {
        oOptions->Await = true;
        AMQP1->CloseLink(aSession, aSenderLink, oOptions);
    }
    __finally
```

```
{  
  delete oOptions;  
}
```

Comandos | AMQP1 Receiver Links

No protocolo AMQP 1.0.0, um **Receiver Link** é um **canal de comunicação** estabelecido entre um cliente AMQP e um servidor AMQP com a finalidade de **receber mensagens**. Ele opera dentro do contexto de uma sessão AMQP, que representa um canal lógico para comunicação entre o cliente e o servidor.

Create Receiver Link

Para **criar um novo Receiver Link**, chame o método **CreateReceiverLink** que contém os seguintes parâmetros:

- **Session:** o nome da sessão à qual o sender link será anexado.
- **Name:** (opcional) o nome do sender link; se não for definido, um nome aleatório será atribuído automaticamente.
- **Source:** (opcional) a source pode ser configurada para indicar a localização do node no host remoto que deve atuar como remetente. Em algumas situações, especificar este endereço pode não ser necessário. Nesses casos, simplesmente fornecer uma string vazia como valor para os parâmetros será suficiente.
- **ReadMode:** (amqp1srmAuto por padrão) Os receiver links podem funcionar em um de dois modos para receber mensagens:
 - **amqp1srmAuto:** Modo Automático, neste modo o receptor trabalha ativamente para garantir que as mensagens sejam recebidas prontamente assim que se tornam disponíveis. Ele escuta e recebe mensagens automaticamente, sem qualquer instrução explícita cada vez que uma nova mensagem chega.
 - **amqp1srmManual:** Modo Baseado em Fetch, neste modo o receptor só recuperará ou buscará (fetch) uma nova mensagem quando for especificamente instruído a fazê-lo. Ao contrário do modo automático, o receptor não escutará ativamente por novas mensagens, mas aguardará instruções manuais para buscar a próxima mensagem.
- **RcvSettleMode:** (amqp1srmFirst por padrão) Os Receiver Links operam em um destes modos:
 - **amqp1srmFirst:** Quando as mensagens chegam, elas serão processadas e confirmadas imediatamente. Se a mensagem ainda não tiver sido confirmada no momento em que foi enviada, o remetente será informado de que a mensagem foi recebida.
 - **amqp1srmSecond:** As mensagens que chegam só serão confirmadas após o remetente tê-las confirmado primeiro. Além disso, o remetente receberá uma notificação quando uma mensagem tiver sido recebida, desde que a mensagem não já tenha sido confirmada quando foi enviada.

Quando o Receiver Link tiver sido criado com sucesso, o evento **OnAMQPLinkOpen** será disparado.

```
oAMQP1->CreateReceiverLink("MySession", "MyReceiverLink");
oAMQP1->OnAMQPLinkOpen = AMQP1AMQPLinkOpen;
void __fastcall TMyForm::AMQP1AMQPLinkOpen(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *const aLink)
{
    ShowMessage("#link-open: " + aLink->Name);
}
```

Aguardar Criação do Receiver Link

Por padrão, o método **CreateReceiverLink** é **Assíncrono**, portanto, após chamar o método, o código continua. Se você quiser **aguardar até que o método CreateReceiverLink seja concluído** e a confirmação enviada pelo servidor seja recebida, defina a propriedade **Await** como True no parâmetro Options.

```
void CreateReceiverLink(const System::UnicodeString aSession, const System::UnicodeString aReceiver)
{
    TsgcAMQP1MethodOptions_CreateReceiverLink *oOptions = new TsgcAMQP1MethodOptions_CreateReceiverLink();
    try
    {
        oOptions->Await = true;
        AMQP1->CreateReceiverLink(aSession, aReceiver, L"", oOptions);
    }
}
```

```

    __finally
    {
        delete oOptions;
    }
}

```

Sync Messages

Quando o Receiver Link funciona no modo Manual ReadMode, chame o método **GetMessage** para obter novas mensagens. Este método é síncrono, o que significa que aguarda até que um timeout seja excedido (por padrão 10 segundos). Quando o método é chamado, o componente aumenta o crédito em uma unidade e aguarda até que uma nova mensagem chegue ou o timeout tenha sido excedido. Se nenhuma mensagem chegar, o crédito é definido como zero novamente.

O método **GetMessage** possui os seguintes parâmetros:

- **Session:** nome da sessão que contém o link.
- **Link:** nome do receiver link.
- **Timeout:** (por padrão 1000 = 10 segundos) o tempo máximo que a função aguardará para obter uma nova mensagem.

Close Receiver Link

Para Fechar um Receiver Link existente, chame o método **CloseLink** que contém os seguintes parâmetros:

- **Session:** nome da sessão que contém o link.
- **Link:** nome do receiver link.
- **Error:** (opcional) aqui você pode definir o motivo pelo qual o link é fechado.

Quando o Receiver Link tiver sido fechado com sucesso, o evento **OnAMQPLinkClose** será disparado.

```

oAMQP1->CloseLink("MySession", "MyReceiverLink");
void __fastcall TForm1::OnAMQPLinkCloseEvent(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *const aLink, TsgcAMQP1MethodOptions *const aOptions)
{
    ShowMessage("#link-close: " + aLink->Name);
}

```

Await Close Receiver Link

Por padrão, o método **CloseLink** é **Assíncrono**, então após chamar o método, o código continua. Se você quiser **aguardar até que o método CloseLink seja concluído** e a confirmação enviada pelo servidor seja recebida, defina a propriedade **Await** como True no parâmetro Options.

```

void CloseReceiverLink(const System::UnicodeString aSession, const System::UnicodeString aReceiverLink)
{
    TsgcAMQP1MethodOptions_CloseLink *oOptions = new TsgcAMQP1MethodOptions_CloseLink();
    try
    {
        oOptions->Await = true;
        AMQP1->CloseLink(aSession, aReceiverLink, oOptions);
    }
    __finally
    {
        delete oOptions;
    }
}

```

AMQP1 | Send Message

Leia primeiro [AMQP1 Sender Links](#) para saber como criar um Sender Link.

Enviar mensagem

Utilize o método **SendMessage** passando o nome da Session e do SenderLink para enviar uma mensagem de texto ao AMQP1 Server. O método tem os seguintes parâmetros:

- **Session:** nome da sessão.
- **Link:** nome do link remetente.
- **Text:** texto da mensagem.
- **MessageId:** (opcional) o id da mensagem, pode ser usado ao utilizar o modo unsettled, para saber se o servidor processou a mensagem.
- **Options:** (opcional) permite personalizar algumas opções ao enviar a mensagem.
 - **Settled:** ao utilizar um sender link em modo misto, ao enviar uma mensagem a propriedade Settled pode ser personalizada.
 - **Await:** se a mensagem não estiver liquidada (unsettled) e o valor for true, o código aguardará até que a mensagem seja processada pelo servidor ou o tempo limite seja excedido.
 - **Timeout:** valor em milissegundos se await for true (por padrão 10000).
 - **RaiseTimeoutException:** se o tempo limite for excedido, uma exceção é gerada (por padrão true).

```
oAMQP1->SendMessage("MySession", "MySenderLink", "My first AMQP Message");
```

Await Send Message

Por padrão, o método **SendMessage** é assíncrono ao enviar uma mensagem unsettled; definindo a propriedade **Await** como true, o cliente aguardará até receber uma confirmação do servidor de que a mensagem foi processada.

```
void SendMessageAwait(const System::UnicodeString aSession, const System::UnicodeString aSenderLink, const System::UnicodeString aText, const System::UnicodeString messageId)
{
    TsgcAMQP1MethodOptions_SendMessageAck *oOptions = new TsgcAMQP1MethodOptions_SendMessageAck();
    try
    {
        oOptions->Settled = false;
        oOptions->Await = true;
        AMQP1->SendMessage(aSession, aSenderLink, aText, L"message-id", oOptions);
    }
    __finally
    {
        delete oOptions;
    }
}
```

Eventos

Ao enviar uma mensagem, há 2 eventos que podem ser usados para saber quando a mensagem é enviada e se a mensagem foi processada pelo servidor (ao enviar como unsettled).

- **OnAMQPMessageSent:** este evento é chamado após a mensagem ser enviada ao servidor. Ao chamar o método `SendMessage`, a mensagem é armazenada em uma fila interna e processada por uma thread secundária, então, após a mensagem ser enviada, este evento é chamado.
- **OnAMQPMessageSentAck:** este evento é chamado quando o cliente recebe uma confirmação de que a mensagem foi processada pelo Servidor AMQP1.

```
void __fastcall OnAMQPMessageSentAck(System::TObject* Sender, TsgcAMQP1Session* const aSession,
    TsgcAMQP1SenderLink* const aLink, const System::UnicodeString aMessageId,
    TsgcAMQP1FrameDeliveryStates aDeliveryState, TsgcAMQP1FrameDisposition aDisposition)
{
    System::UnicodeString vMessageId = aMessageId;
    switch (aDeliveryState)
    {
        case amqp1fdtsAccepted:
            ShowMessage("#msg-accepted: " + vMessageId);
            break;
        case amqp1fdtsRejected:
            ShowMessage("#msg-rejected: " + vMessageId + " " +
                TsgcAMQP1FrameRejected(aDisposition.State)->Error->Condition + " " +
                TsgcAMQP1FrameRejected(aDisposition.State)->Error->Description);
            break;
        case amqp1fdtsReleased:
            ShowMessage("#msg-released: " + vMessageId);
            break;
        case amqp1fdtsModified:
            ShowMessage("#msg-modified: " + vMessageId + " " +
                TsgcAMQP1FrameModified(aDisposition.State)->MessageAnnotations);
            break;
        case amqp1fdtsReceived:
            ShowMessage("#msg-received: " + vMessageId);
            break;
    }
}
```


AMQP1 | Read Message

Toda vez que uma nova mensagem é recebida, o evento **OnAMQPMessage** é disparado.

A instância de `TsgcAMQP1Message` contém a mensagem recebida. Você pode acessar a mensagem de texto utilizando a propriedade **aMessage.ApplicationData.AMQPValue.Value**.

Para especificar o Delivery Outcome, utilize o parâmetro **DeliveryState**. Por padrão, todas as mensagens têm o estado accepted, mas você pode definir um dos seguintes:

- **amqp1mdtsAccepted:** A mensagem foi processada com sucesso.
- **amqp1mdtsRejected:** A mensagem não foi processada com sucesso. Defina o erro utilizando a propriedade `DeliveryState.Rejected`.
- **amqp1mdtsReleased:** A mensagem não foi e não será processada.
- **amqp1mdtsModified:** Igual a `amqp1mdtsReleased`, mas você pode adicionar dados adicionais utilizando a propriedade `DeliveryState.Modified`.

```
void __fastcall OnAMQPMessage(System::TObject* Sender, TsgcAMQP1Session* const aSession,
    TsgcAMQP1ReceiverLink* const aLink, TsgcAMQP1Message* const aMessage,
    TsgcAMQP1MessageDeliveryState &DeliveryState)
{
    if (aMessage->ApplicationData->AMQPValue->Value == "xxx")
    {
        DeliveryState.State = amqp1mdtsRejected;
        DeliveryState.Rejected->Error->Condition = "amqp-error-processing";
        DeliveryState.Rejected->Error->Description = "Value received was not the expected.";
    }
    else
    {
        DeliveryState.State = amqp1mdtsAccepted;
    }
}
```

Protocol STOMP

STOMP é o Simple (ou Streaming) Text Orientated Messaging Protocol. STOMP fornece um formato físico interoperável para que clientes STOMP possam comunicar-se com qualquer message broker STOMP, proporcionando interoperabilidade de mensagens fácil e ampla entre muitas linguagens, plataformas e brokers.

Nossos componentes cliente STOMP suportam as seguintes versões STOMP: 1.0, 1.1 e 1.2.

Componentes

TsgcWSPClient_STOMP: cliente genérico do protocolo STOMP, permite que você se conecte a qualquer STOMP Server.

TsgcWSPClient_STOMP_RabbitMQ: cliente STOMP para o Broker RabbitMQ.

TsgcWSPClient_STOMP_ActiveMQ: cliente STOMP para o Broker ActiveMQ.

TsgcWSPClient_STOMP

Este é o componente do protocolo cliente STOMP.

Introdução

Este é o Componente Cliente Protocolo STOMP. Você precisa colocar este componente no formulário e selecionar um Componente [TsgcWebSocketClient](#) utilizando a propriedade Client.

Referência

- Propriedades — 8 propriedades disponíveis no componente.
- Métodos — 11 métodos públicos.
- Events — 6 eventos que você pode tratar.

TsgcWSPClient_STOMP_RabbitMQ

Este é o Componente Cliente de Protocolo STOMP para o Broker RabbitMQ.

Introdução

Este é o Componente do Protocolo STOMP para Cliente para o Broker RabbitMQ. Você precisa soltar este componente no formulário e selecionar um Componente [TsgcWebSocketClient](#) utilizando a propriedade Client.

Destinos

A especificação STOMP não prescreve quais tipos de destinos um broker deve suportar; em vez disso, o valor do cabeçalho destination nos frames SEND e MESSAGE é específico do broker. O adaptador STOMP do RabbitMQ suporta uma série de tipos de destino diferentes:

- **Topic:** SEND e SUBSCRIBE em tópicos transientes e duráveis.
- **Queue:** SEND e SUBSCRIBE para filas gerenciadas pelo gateway STOMP.
- **QueueOutside:** SEND e SUBSCRIBE em filas criadas fora do gateway STOMP.
- **TemporaryQueue:** cria queues temporárias (apenas em cabeçalhos reply-to).
- **Exchange:** SEND para routing keys arbitrárias e SUBSCRIBE para padrões de binding arbitrários.

Referência

- Propriedades — 8 propriedades disponíveis no componente.
- Métodos — 25 métodos públicos.
- Eventos — 6 eventos que você pode tratar.

TsgcWSPClient_STOMP_ActiveMQ

Este é o componente Client Protocol STOMP para o Broker ActiveMQ.

Introdução

Este é o componente do protocolo cliente STOMP para o Broker ActiveMQ. Você precisa colocar este componente no formulário e selecionar um componente [TsgcWebSocketClient](#) utilizando a propriedade Client.

Destinos

A especificação STOMP não prescreve quais tipos de destinos um broker deve suportar; em vez disso, o valor do cabeçalho destination nos frames SEND e MESSAGE é específico do broker. O adaptador Active STOMP suporta diversos tipos de destino diferentes:

- **Topic:** SEND e SUBSCRIBE em tópicos transientes e duráveis.
- **Queue:** SEND e SUBSCRIBE para filas gerenciadas pelo gateway STOMP.

Publish Options

Observe que o STOMP é projetado para ser o mais simples possível, de modo que qualquer linguagem/plataforma de scripting possa enviar mensagens a qualquer outra com esforço mínimo. O STOMP permite cabeçalhos plugáveis em cada requisição, como ao enviar e receber mensagens. O ActiveMQ tem várias extensões ao protocolo Stomp, de modo que a semântica JMS possa ser suportada por clientes Stomp. Um produtor JMS OpenWire pode enviar mensagens a um consumer Stomp, e um produtor Stomp pode enviar mensagens a um consumer JMS OpenWire. E configurações Stomp para Stomp podem usar o controle mais rico de mensagens do JMS.

O STOMP suporta as seguintes propriedades JMS padrão em mensagens SENT:

- **CorrelationId:** Bons consumidores adicionarão este header a quaisquer respostas que enviarem.
- **Expires:** Tempo de expiração da mensagem.
- **JMSXGroupID:** Especifica os Message Groups.
- **JMSXGroupSeq:** Cabeçalho opcional que especifica o número de sequência nos Message Groups.
- **Persistent:** Se a mensagem é persistente ou não.
- **Priority:** Prioridade da mensagem.
- **ReplyTo:** Destino para onde você deve enviar as respostas.
- **MsgType:** Tipo da mensagem.

Referência

- Propriedades — 9 propriedades disponíveis no componente.
- Métodos — 16 métodos públicos.
- Eventos — 6 eventos que você pode tratar.

Protocolo AppRTC

WebRTC (Web Real-Time Communication) é uma definição de API que está sendo elaborada pelo World Wide Web Consortium (W3C) para habilitar aplicações browser-to-browser para chamadas de voz, video chat e compartilhamento de arquivos P2P sem plugins. O RTC em WebRTC significa Real-Time Communications, uma tecnologia que habilita streaming de áudio/vídeo e compartilhamento de dados entre clientes de navegador (peers). Como um conjunto de padrões, o WebRTC fornece a qualquer navegador a capacidade de compartilhar dados de aplicação e realizar teleconferência peer to peer, sem a necessidade de instalar plug-ins ou software de terceiros.

Os componentes WebRTC são acessados com APIs JavaScript. Atualmente, em desenvolvimento estão a Network Stream API, que representa um stream de dados de áudio ou vídeo, e a PeerConnection API, que permite que dois ou mais usuários se comuniquem navegador-a-navegador. Também em desenvolvimento está uma DataChannel API que permite a comunicação de outros tipos de dados para jogos em tempo real, chat de texto, transferência de arquivos e assim por diante.

[appr.tc](#) é uma aplicação de demonstração WebRTC desenvolvida pela Google e Mozilla, ela permite que ambos os navegadores "conversem" entre si utilizando a API WebRTC.

Componentes

[TsgcWSPServer_AppRTC](#): Componente VCL Server Protocol AppRTC.

TsgcWSPServer_AppRTC

Este é o Componente de Protocolo Servidor AppRTC.

Introdução

Este é o componente Server Protocol AppRTC. Você precisa colocar este componente no formulário e selecionar um componente [TsgcWebSocketServer](#) utilizando a propriedade Server.

Parâmetros

- **IceServers:** aqui você pode configurar servidores turn/stun para conexões WebRTC.
- **RoomLink:** URL base para acessar a sala. Exemplo: <https://mydemo.com/r/>
- **WebSocketURL:** URL para o servidor WebSocket. Exemplo: <wss://mydemo.com>

O protocolo WebRTC requer um servidor STUN/TURN; as demos usam servidores STUN/TURN públicos para fins de teste. Para colocar em um sistema de produção, é necessário um servidor STUN/TURN dedicado.

Usuários registrados podem baixar binários compilados do **servidor Coturn para Windows**. Leia mais sobre [COTURN STUN/TURN](#).

Configuração de IceServers

Se você estiver executando seu servidor STUN/TURN no seguinte Endereço IP: 51.122.4.88 e estiver escutando na porta 3478. O usuário para conectar é "apprtc" e a credencial é "secret". Configure os IceServers da seguinte forma:

```
{
  "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"
}
```

Referência

- Properties — 5 propriedades disponíveis no componente.
- Métodos — 0 métodos públicos.
- Eventos — 11 eventos que você pode tratar.

Protocolo WebRTC

WebRTC (Web Real-Time Communication) é uma definição de API que está sendo elaborada pelo World Wide Web Consortium (W3C) para habilitar aplicações browser-to-browser para chamadas de voz, video chat e compartilhamento de arquivos P2P sem plugins. O RTC em WebRTC significa Real-Time Communications, uma tecnologia que habilita streaming de áudio/vídeo e compartilhamento de dados entre clientes de navegador (peers). Como um conjunto de padrões, o WebRTC fornece a qualquer navegador a capacidade de compartilhar dados de aplicação e realizar teleconferência peer to peer, sem a necessidade de instalar plug-ins ou software de terceiros.

Os componentes WebRTC são acessados com APIs JavaScript. Atualmente, em desenvolvimento estão a Network Stream API, que representa um stream de dados de áudio ou vídeo, e a PeerConnection API, que permite que dois ou mais usuários se comuniquem navegador-a-navegador. Também em desenvolvimento está uma DataChannel API que permite a comunicação de outros tipos de dados para jogos em tempo real, chat de texto, transferência de arquivos e assim por diante.

Componentes

[TsgcWSPServer_WebRTC](#): Componente VCL de protocolo de servidor WebRTC.

Parâmetros

- **IceServers**: aqui você pode configurar servidores turn/stun para conexões WebRTC. Por padrão utiliza os seguintes servidores STUN públicos

```
{"iceServers": [{"url": "stun:stun.l.google.com:19302"}]}
```

Browser Test

Se você quiser testar este protocolo com seu navegador web favorito, digite esta url (você precisa definir seu host e porta personalizados)

```
http://host:port/webrtc.esegece.com.html
```


TsgcWSPServer_WebRTC

Este é o Componente Server Protocol WebRTC.

Introdução

Este é o Componente WebRTC do Protocolo Servidor. Você precisa soltar este componente no formulário e selecionar um Componente [TsgcWebSocketServer](#) utilizando a propriedade Server.

O protocolo WebRTC requer um servidor STUN/TURN; as demos usam servidores STUN/TURN públicos para fins de teste. Para colocar em um sistema de produção, é necessário um servidor STUN/TURN dedicado.

Usuários registrados podem baixar binários compilados do **servidor Coturn para Windows**. Leia mais sobre [COTURN STUN/TURN](#).

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Methods — 2 métodos públicos.
- Eventos — 11 eventos que você pode tratar.

Protocolo WebRTC Javascript

Aqui você pode encontrar os métodos disponíveis; você precisa substituir as variáveis `{%host%}` e `{%port%}` conforme necessário, exemplo: se você configurou seu servidor `sgcWebSocket` para escutar a porta 80 no site `www.example.com`, você precisa configurar:

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
<script src="http://www.example.com:80/webrtc.esegce.com.js"></script>
```

Open Connection

Quando uma conexão `WebSocket` é aberta, o navegador solicita acesso à câmera e ao microfone locais, você precisa permitir o acesso.

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/webrtc.esegce.com.js"></script>
<script>
  var socket = new sgcws_webrtc('ws://{%host%}:{%port%}');
</script>
```

Open WebRTC Channel

Quando um navegador tem acesso à câmera e ao microfone locais, o evento `'sgcmediastart'` é disparado e então você pode tentar conectar a outro cliente utilizando o procedimento `webrtc_connect`

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/webrtc.esegce.com.js"></script>
<script>
  var socket = new sgcws_webrtc('ws://{%host%}:{%port%}');
  socket.on('sgcmediastart', function(event)
  {
    socket.webrtc_connect('custom channel');
  }
</script>
```

Fechar canal WebRTC

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/webrtc.esegce.com.js"></script>
<script>
  socket.webrtc_disconnect('custom channel');
</script>
```

Protocolo WAMP

O WAMP é um subprotocolo WebSocket aberto que fornece dois padrões de mensageria assíncrona: RPC e PubSub.

Tecnicamente, o WAMP é um subprotocolo WebSocket oficialmente registrado (executa sobre WebSocket) que usa JSON como formato de serialização de mensagens.

O que é RPC?

A Chamada de Procedimento Remoto (RPC) é um padrão de mensagens envolvendo peers de dois papéis: cliente e servidor.

Um servidor fornece métodos ou procedimentos para chamar em endpoints bem conhecidos.

Um cliente chama métodos ou procedimentos remotos fornecendo o endpoint do método ou procedimento e quaisquer argumentos para a chamada.

O servidor executará o método ou procedimento usando os argumentos fornecidos à chamada e retornará o resultado da chamada ao cliente.

O que é PubSub?

Publish & Subscribe (PubSub) é um padrão de mensagens que envolve peers de três papéis: publicador, assinante e broker.

Um publicador envia (publica) um evento fornecendo um tópico (também chamado de canal) como o endereço abstrato, e não um peer específico.

Um subscriber recebe eventos primeiro fornecendo os tópicos (também conhecidos como canais) em que está interessado. Subsequentemente, o subscriber receberá quaisquer eventos publicados naquele tópico.

O broker fica entre os publicadores e os assinantes e media as mensagens que os publicadores enviam aos assinantes. Um broker mantém listas de assinantes por tópico para poder despachar eventos recém-publicados aos assinantes apropriados.

Um broker também pode despachar eventos por conta própria, por exemplo quando o broker também atua como um servidor RPC e um método executado no servidor deve acionar um evento PubSub.

Em resumo, o PubSub desacopla publicadores e receptores por meio de um intermediário, o broker.

Componentes

[TsgcWSPServer_WAMP](#): Componente VCL do Protocolo WAMP do Servidor.

[TsgcWSPClient_WAMP](#): Componente VCL Client Protocol WAMP.

[Componente Javascript](#): Referência do Cliente Javascript.

Usos mais comuns

- **RPC**
 - [RPC Simples](#)
 - [RPC Progress Results](#)
- **PubSub**
 - [Assinantes](#)
 - [Publishers](#)

Browser Test

Se você quiser testar este protocolo com seu navegador web favorito, por favor digite esta URL (você precisa definir seu host e porta personalizados)

`http://host:port/wamp.esegece.com.html`

TsgcWSPServer_WAMP

Este é o Componente de Protocolo WAMP do Servidor.

Introdução

Este é o componente Server Protocol WAMP. Você precisa soltar este componente no formulário e selecionar um componente [TsgcWebSocketServer](#) utilizando a propriedade Server.

Referência

- Propriedades — 4 propriedades disponíveis no componente.
- Métodos — 4 métodos públicos.
- Eventos — 14 eventos que você pode tratar.

TsgcWSPClient_WAMP

Este é o Componente WAMP do Protocolo Cliente.

Introdução

Este é o componente Client Protocol WAMP. Você precisa soltar este componente no formulário e selecionar um Componente [TsgcWebSocketClient](#) utilizando a propriedade Client.

Referência

- Propriedades — 3 propriedades disponíveis no componente.
- Métodos — 7 métodos públicos.
- Eventos — 13 eventos que você pode tratar.

Protocol WAMP Javascript

Aqui você pode encontrar os métodos disponíveis; você precisa substituir as variáveis `{%host%}` e `{%port%}` conforme necessário, exemplo: se você configurou seu servidor `sgcWebSocket` para escutar a porta 80 no site `www.example.com`, você precisa configurar:

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
<script src="http://www.example.com:80/wamp.esegece.com.js"></script>
```

Open Connection

```
<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>
```

Send New Prefix

```
<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>
```

Request 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>
```

Assinar uma 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>
```

UnSubscribe de uma 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>
```

Publicar mensagem

```
<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>
```

Show Alert with Message Received

```
<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>
```

Show Alert OnCallResult or 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>
```

Show Alert 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>
```

Show Alert OnConnect, OnDisconnect and OnError Events

```
<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>
```

Close Connection

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  socket.close();
</script>
```

Obter Status da Conexão

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  socket.state();
</script>
```

WAMP | Subscribers

Um subscriber recebe eventos primeiro fornecendo os tópicos (também conhecidos como canais) em que está interessado. Subsequentemente, o subscriber receberá quaisquer eventos publicados naquele tópico. Para receber eventos de um tópico, o assinante primeiro precisa se inscrever nesse tópico.

WAMP Client

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

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

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

WAMP Server

```
void OnSubscriptionEvent(TsgcWSConnection *Connection, string Subscription)
{
    ShowMessage("Subscribed: " + Subscription);
}

oServer = new TsgcWebSocketServer();
oServer->Port = 80;
oServerWAMP = new TsgcWSPServer_WAMP();
oServerWAMP->OnSubscription = OnSubscriptionEvent;
oServerWAMP->Server = oServer;
oServerWAMP->Active = true;
```

WAMP | Publishers

Um publicador envia (publica) um evento fornecendo um tópico (também conhecido como canal) como endereço abstrato, não um peer específico. Basta chamar o método Publish e passar como argumentos o nome do tópico e a mensagem que você deseja enviar. Esta mensagem será entregue a todos os assinantes deste tópico. Como nota, não há necessidade de assinar um tópico para publicar mensagens nesse tópico.

Não há necessidade de configurar nada no lado do servidor, porque as mensagens são automaticamente transmitidas aos clientes quando uma mensagem de publicação é recebida.

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 simples

O uso mais comum do componente WAMP é um cliente requisitar um método do servidor, e o servidor enviar uma resposta ao cliente. O cliente pode enviar apenas o nome do método e/ou pode passar alguns parâmetros exigidos pelo servidor para calcular o resultado. O servidor processa as requisições e, se bem-sucedido, envia uma resposta ao cliente com o resultado. Se houver algum erro, o servidor envia uma resposta de erro ao cliente. Como você pode ver, há apenas Uma requisição e Uma resposta (bem-sucedida ou não).

Exemplo: o servidor tem um método chamado **GetTime**, portanto toda vez que um cliente requisita este método, o servidor retorna o horário do servidor.

WAMP Server

```
void OnServerCall(TsgcWSConnection *Connection, const string CallId, const string ProcUri, const string Arguments)
{
    if (ProcUri == "GetTime")
    {
        oServerWAMP->CallResult(CallId, FormatDateTime("yyymmdd 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);
{
    ShowMessage(Result);
}

void OnCallErrorClient(TsgcWSConnection *Connection, string Error)
{
    ShowMessage(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 | Resultados Progressivos de RPC

Às vezes, as Chamadas de Procedimento Remoto exigem mais de um resultado para finalizar as requisições; por padrão, o protocolo WAMP 1.0 não permite resultados parciais em uma chamada, este é um recurso exclusivo da biblioteca `sgcWebSockets`.

O fluxo é muito semelhante a um RPC simples, mas aqui há 1 ou mais resultados parciais antes de `CallResult` ser chamado para finalizar o processo.

Basicamente, um cliente solicita um procedimento ao servidor, e o servidor pode enviar um resultado ou um erro. Se ele enviar um resultado, este pode ser o resultado final ou ele deve enviar mais resultados posteriormente. Se for o resultado final, chamará o método **CallResult** e o processo será finalizado. Se houver mais resultados a enviar, chamará o método **CallProgressResult**.

Exemplo: o cliente requisita ao servidor um método para receber a cada segundo o horário do servidor e parar após 20 mensagens.

WAMP Server

```
void OnServerCall(TsgcWSConnection *Connection, const string CallId, const string ProcUri, const string Arguments)
{
    if (ProcUri == "GetProgressiveTime")
    {
        int vNum = StrToInt(Arguments);
        for (int i = 1; i = vNum; i++)
        {
            if (i == 20)
            {
                oServerWAMP->CallResult(CallId, FormatDateTime("yyymmdd hh:nn:ss", Now));
            }
            else
            {
                oServerWAMP->CallProgressiveResult(CallId, FormatDateTime("yyymmdd 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);
{
    ShowMessage(Result);
}

void OnCallProgressResultClient(TsgcWSConnection *Connection, string CallId, string Result);
{
    ShowMessage(Result);
}

void OnCallErrorClient(TsgcWSConnection *Connection, string Error)
{
    ShowMessage(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");
```

Protocolo WAMP 2

O WAMP fornece Roteamento Unificado de Aplicações em um protocolo WebSocket aberto que funciona com diferentes linguagens.

Usando WAMP, você pode construir sistemas distribuídos a partir de componentes de aplicação que são fracamente acoplados e se comunicam em tempo (quase) real.

Em sua essência, o WAMP oferece dois padrões de comunicação para que os componentes de aplicação conversem entre si:

- Publish & Subscribe (PubSub)
- Chamadas de Procedimento Remoto (RPC)

O WAMP é fácil de usar, simples de implementar e baseado em padrões web modernos: WebSocket, JSON e URIs.

Componentes

[TsgcWSPClient_WAMP2](#): Componente VCL do Protocolo WAMP2 do Cliente.

TsgcWSPClient_WAMP2

Este é o Componente Cliente do Protocolo WAMP2.

Introdução

Este é o Componente de Protocolo Cliente WAMP2. Você precisa soltar este componente no formulário e selecionar um Componente [TsgcWebSocketClient](#) usando a propriedade Client.

Métodos de Session

- **ABORT:** Tanto o Router quanto o Client podem abortar a abertura de uma sessão WAMP enviando uma mensagem ABORT.

Reason DEVE ser uma URI.

Details DEVE ser um dicionário que permite fornecer informações de fechamento adicionais e opcionais (veja abaixo).

Nenhuma resposta a uma mensagem ABORT é esperada.

- **GOODBYE:** Uma sessão WAMP inicia seu ciclo de vida com o Router enviando uma mensagem WELCOME ao Cliente e termina quando o transporte subjacente desaparece ou quando a sessão WAMP é encerrada explicitamente por uma mensagem GOODBYE enviada por um Peer e uma mensagem GOODBYE enviada pelo outro Peer em resposta.

Reason DEVE ser uma URI.

Details DEVE ser um dicionário que permite fornecer informações de fechamento adicionais e opcionais.

Métodos Publish/Subscribe

- **PUBLISH:** Quando um Publisher solicita publicar um evento em algum tópico, ele envia uma mensagem PUBLISH a um Broker:

Request é um ID aleatório e efêmero escolhido pelo Publisher e utilizado para correlacionar a resposta do Broker com a requisição.

Options é um dicionário que permite fornecer detalhes adicionais da requisição de publicação de forma extensível. Isto é descrito em mais detalhes abaixo.

Topic é o tópico para o qual é publicado.

Arguments é uma lista de elementos de payload de evento em nível de aplicação. A lista pode ter comprimento zero.

ArgumentsKw é um dicionário opcional contendo o payload de evento em nível de aplicação, fornecido como argumentos nomeados. O dicionário pode estar vazio.

Se o Broker for capaz de atender e permitir a publicação, o Broker enviará o evento a todos os Assinantes atuais do tópico do evento publicado.

Por padrão, as publicações não são confirmadas, e o Broker não responderá, independentemente de a publicação ter sido bem-sucedida ou não.

- **SUBSCRIBE:** Um Subscriber comunica seu interesse em um tópico a um Broker enviando uma mensagem SUBSCRIBE:

Request DEVE ser um ID aleatório e efêmero escolhido pelo Subscriber e utilizado para correlacionar a resposta do Broker com a requisição.

Options DEVE ser um dicionário que permite fornecer detalhes adicionais da requisição de assinatura de forma extensível.

Topic é o tópico ao qual o Subscriber deseja se inscrever e DEVE ser uma URI.

- **UNSUBSCRIBE:** Quando um Subscriber não está mais interessado em receber eventos de uma subscrição, ele envia uma mensagem UNSUBSCRIBE

Request DEVE ser um ID aleatório e efêmero escolhido pelo Subscriber e utilizado para correlacionar a resposta do Broker com a requisição.

SUBSCRIBED.Subscription DEVE ser o ID da assinatura da qual cancelar a inscrição, originalmente entregue pelo Broker ao Subscriber.

RPC Methods

- **CALL:** Quando um Caller deseja chamar um procedimento remoto, ele envia uma mensagem CALL a um Dealer:

Request é um ID aleatório e efêmero escolhido pelo Caller e usado para correlacionar a resposta do Dealer com a requisição.

Options é um dicionário que permite que você forneça detalhes adicionais da requisição de chamada de forma extensível. Isso é descrito mais adiante abaixo.

Procedure é a URI do procedimento a ser chamado.

Arguments é uma lista de argumentos posicionais da chamada (cada um de tipo arbitrário). A lista pode ter comprimento zero.

ArgumentsKw é um dicionário de argumentos de chamada nomeados (cada um de tipo arbitrário). O dicionário pode estar vazio.

- **REGISTERCALL:** Um Callee anuncia a disponibilidade de um endpoint que implementa um procedimento junto a um Dealer enviando uma mensagem REGISTER:

Request é um ID aleatório e efêmero escolhido pelo Callee e usado para correlacionar a resposta do Dealer com a requisição.

Options é um dicionário que permite fornecer detalhes adicionais da requisição de registro de forma extensível. Isto é descrito mais adiante.

Procedure é o procedimento que o Callee deseja registrar

- **UNREGISTERCALL:** Quando um Callee não está mais disposto a fornecer uma implementação do procedimento registrado, ele envia uma mensagem UNREGISTER ao Dealer:

Request é um ID aleatório e efêmero escolhido pelo Callee e usado para correlacionar a resposta do Dealer com a requisição.

REGISTERED.Registration é o ID do registro a ser revogado, originalmente entregue pelo Dealer ao Callee.

- **INVOCATION:** Se o Dealer for capaz de atender (mediar) a chamada e a permitir, ele envia uma mensagem INVOCATION ao respectivo Callee que implementa o procedimento:

Request é um ID aleatório e efêmero escolhido pelo Dealer e utilizado para correlacionar a resposta do Callee com a requisição.

REGISTERED.Registration é o ID de registro sob o qual o procedimento foi registrado no Dealer.

Details é um dicionário que permite que você forneça detalhes adicionais da requisição de invocação de forma extensível. Isso é descrito mais adiante abaixo.

CALL.Arguments é a lista original de argumentos posicionais da chamada conforme fornecidos pelo Caller.

CALL.ArgumentsKw é o dicionário original de argumentos de chamada por palavra-chave conforme fornecido pelo Caller.

- **YIELD:** Se o Callee for capaz de processar e finalizar a execução da chamada com sucesso, ele responde enviando uma mensagem YIELD ao Dealer:

INVOCATION.Request é o ID da requisição de invocação original.

Options é um dicionário que permite fornecer opções adicionais.

Arguments é uma lista de elementos de resultado posicionais (cada um de tipo arbitrário). A lista pode ter comprimento zero.

ArgumentsKw é um dicionário de elementos de resultado nomeados (cada um de tipo arbitrário). O dicionário pode estar vazio.

Referência

- Propriedades — 3 propriedades disponíveis no componente.
- Métodos — 11 métodos públicos.
- Eventos — 20 eventos que você pode tratar.

Protocol Default

Este é o subprotocolo padrão implementado usando mensagens "JSONRPC 2.0". Toda vez que você envia uma mensagem usando este protocolo, um objeto JSON é criado com as seguintes propriedades:

jsonrpc: Uma String especificando a versão do protocolo JSON-RPC. DEVE ser exatamente "2.0".

method: Uma String contendo o nome do método a ser invocado. Nomes de métodos que começam com a palavra `rpc` seguida por um caractere de ponto (U+002E ou ASCII 46) são reservados para métodos e extensões internos de `rpc` e NÃO DEVEM ser usados para qualquer outra coisa.

params: Um valor Estruturado que contém os valores de parâmetro a serem utilizados durante a invocação do método. Este membro PODE ser omitido.

id: Um identificador estabelecido pelo Cliente que DEVE conter um valor String, Number ou NULL se incluído. Se não for incluído, presume-se que seja uma notificação. O valor normalmente NÃO DEVE ser Null [1] e Numbers NÃO DEVEM conter partes fracionárias [2]

Exemplo de objeto JSON:

```
{"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "id": 1}
```

Features

- Padrão de mensagem **Publish/subscribe** para fornecer distribuição de mensagens um-para-muitos e desacoplamento de aplicações. Suporta caracteres Wildcard, para que você possa assinar uma hierarquia de canais. Exemplo: se você quiser assinar todos os canais que começam com 'news', então chame `Subscribe('news*')`.
- Um transporte de mensagens que é **agnóstico** ao conteúdo do payload
- **Acknowledgment** das mensagens enviadas.
- Suporta **mensagens transacionais** através de transações locais do servidor. Quando o cliente confirma a transação, o servidor processa todas as mensagens enfileiradas. Se o cliente reverte a transação, então todas as mensagens são excluídas.
- Implementa **QoS** (Quality of Service) para entrega de mensagens.

Componentes

[TsgcWSPClient_sgc](#): Componente VCL de protocolo de servidor padrão.

[TsgcWSPClient_sgc](#): Componente VCL do Protocolo Padrão do Cliente.

[Componente Javascript](#): Referência do cliente Javascript.

Browser Test

Se você quiser testar este protocolo com seu navegador web favorito, por favor digite esta URL (você precisa definir seu host e porta personalizados)

`http://host:port/esegece.com.html`

TsgcWSPServer_sgc

Este é o Componente Padrão de Protocolo de Servidor.

Introdução

Este é o Componente do Protocolo Default para Servidor. Você precisa soltar este componente no formulário e selecionar um Componente [TsgcWebSocketServer](#) utilizando a propriedade Server.

Referência

- Propriedades — 6 propriedades disponíveis no componente.
- Métodos — 6 métodos públicos.
- Eventos — 16 eventos que você pode tratar.

TsgcWSPClient_sgc

Este é o Componente Cliente do Protocolo Default.

Introdução

Este é o componente padrão do protocolo cliente. Você precisa colocar este componente no formulário e selecionar um componente [TsgcWebSocketClient](#) utilizando a propriedade Client.

Referência

- Properties — 5 propriedades disponíveis no componente.
- Métodos — 11 métodos públicos.
- Events — 15 eventos que você pode tratar.

TsgclWWSPClient_sgc

Este é o Componente Padrão do Protocolo Cliente Intraweb.

Introdução

Este é o Componente Default do Intraweb Client Protocol. Você precisa colocar este componente no form e selecionar um Componente [TsgclWWebSocketClient](#) utilizando a propriedade Client.

Referência

- Propriedades — 2 propriedades disponíveis no componente.
- Methods — 3 métodos públicos.
- Eventos — 6 eventos que você pode tratar.

Protocol Default Javascript

O Protocolo Padrão Javascript sgcWebSockets usa os arquivos **sgcWebSocket.js** e **esegece.com.js**.

Aqui você pode encontrar os métodos disponíveis; você precisa substituir as variáveis `{%host%}` e `{%port%}` conforme necessário, exemplo: se você configurou seu servidor sgcWebSocket para escutar a porta 80 no site `www.example.com`, você precisa configurar:

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
<script src="http://www.example.com:80/esegece.com.js"></script>
```

Open Connection

```
<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>
```

Enviar mensagem

```
<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>
```

Show Alert with Message Received

```
<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>
```

Publicar mensagem no canal de teste

```
<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>
```


Exibir Alerta com Mensagem do Evento Recebida

```
<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('sgcevent', function(event)
  {
    alert('channel: ' + event.channel + ' . message: ' + event.message);
  }
</script>
```

Call RPC

```
<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%}');
  var params = {param:10};
  socket.rpc(GUID(), 'test', JSON.stringify(params));
</script>
```

Manipular resposta de RPC

```
<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('sgcrpcresult', function(event)
  {
    alert('result: ' + event.result);
  }
  socket.on('sgcrpcerror', function(event)
  {
    alert('error: ' + event.code + ' ' + event.message);
  }
</script>
```

Notificar Chamada

```
<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%}');
  var params = {param:10};
  socket.notify('test', JSON.stringify(params));
</script>
```

Enviar Mensagens em uma Transação

```
<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.starttransaction('sgc:test');
  socket.publish('Message1', 'sgc:test');
  socket.publish('Message2', 'sgc:test');
```

```
socket.publish('Message3', 'sgc:test');
socket.commit('sgc:test');
</script>
```

Mostra um Alert no OnSubscribe ou OnUnSubscribe de um canal

```
<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>
```

Show Alert OnConnect, OnDisconnect and OnError Events

```
<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>
```

Close Connection

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  socket.close();
</script>
```

Obter Status da Conexão

```
<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>
```

Set 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>
```

Set Queue Level

```
<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>
```

Protocolo Dataset

Este protocolo herda do Protocol Default e é útil se você quiser transmitir (broadcast) alterações de dataset para os clientes conectados a este protocolo. Ele pode ser utilizado em 2 modos:

1. Replicar banco de dados: as alterações do banco de dados são replicadas para todos os bancos de dados dos clientes, exemplo: um servidor tem um banco de dados com cotações de ações e todos os clientes conectados recebem as alterações de cotação. Há um único banco de dados (no servidor) e cada cliente tem seu próprio banco de dados. Cada vez que uma cotação é atualizada, essa alteração é transmitida para todos os clientes conectados e cada cliente atualiza seu próprio registro no banco de dados. Use UpdateMode: upWhereAll ou upWhereChanged para este tipo de modo.

2. Atualizações de banco de dados: aqui há um único banco de dados compartilhado pelo servidor e pelos clientes, e toda vez que há um cliente que atualiza um registro em um banco de dados, todos os outros clientes que rem ser notificados sobre essa atualização. Utilize UpdateMode: upRefreshAll para este modo.

Most common uses

- **Update Mode**
 - [Como Replicar Tabela](#)
 - [Como Notificar Atualizações](#)

Ele utiliza o Objeto "JSON-RPC 2.0" e, toda vez que há uma alteração no dataset, envia todos os valores de campo (* apenas campos suportados) utilizando o Objeto Dataset.

Para permitir que o componente busque registros no dataset, você precisa especificar quais campos são a Chave, **exemplo:** se no seu dataset o campo ID for a chave, você precisará escrever um código como este

```
void OnAfterOpenDataSet(TDataSet *DataSet)
{
    DataSet->FieldByName("ID")->ProviderFlags =
        DataSet->FieldByName("ID")->ProviderFlags + [pfInKey];
}
```

Componentes

[TsgcWSPServer_Dataset](#): Componente VCL do Protocolo Dataset para Servidor.

[TsgcWSPClient_Dataset](#): Componente VCL Client Protocol Dataset.

[Componente Javascript](#): Referência do Cliente Javascript.

Browser Test

Se você quiser testar este protocolo com seu navegador web favorito, por favor digite esta URL (você precisa definir seu host e porta personalizados)

`http://host:port/dataset.esegece.com.html`

TsgcWSPServer_Dataset

Este é o Componente de Protocolo Dataset do Servidor.

Introdução

Este é o Componente de Protocolo Servidor Dataset. Você precisa soltar este componente no formulário e selecionar um Componente [TsgcWebSocketServer](#) usando a propriedade Server e selecionar um Componente Dataset usando a propriedade Dataset.

Este componente herda de [TsgcWSProtocol_Server_sgc](#) todos os métodos e propriedades.

Referência

- Propriedades — 13 propriedades disponíveis no componente.
- Methods — 9 métodos públicos.
- Eventos — 22 eventos que você pode tratar.

TsgcWSPClient_Dataset

Este é o Componente Cliente Protocolo Dataset.

Introdução

Este é o Componente de Protocolo Dataset do Cliente. Você precisa colocar este componente no formulário e selecionar um Componente [TsgcWebSocketClient](#) usando a propriedade Client e selecionar um Componente Dataset usando a propriedade Dataset.

Este componente herda do [TsgcWSProtocol_Client_sgc](#) todos os métodos e propriedades.

Referência

- Propriedades — 13 propriedades disponíveis no componente.
- Métodos — 14 métodos públicos.
- Eventos — 25 eventos que você pode tratar.

TsgclWWSPClient_Dataset

Este é o Componente Dataset do IntraWeb Client Protocol.

Introdução

Este é o Componente Dataset do Protocolo de Cliente IntraWeb. Você precisa soltar este componente no formulário e selecionar um Componente [TsgclWWWebSocketClient](#) usando a propriedade Client e selecionar um Componente Dataset usando a propriedade Dataset.

Este componente herda de [TsgclWWSPClient_sgc](#) todos os métodos e propriedades.

Referência

- Propriedades — 3 propriedades disponíveis no componente.
- Métodos — 6 métodos públicos.
- Eventos — 9 eventos que você pode tratar.

Protocolo Dataset Javascript

O Dataset Protocol Javascript do sgcWebSockets utiliza os arquivos **sgcWebSocket.js** e **dataset.esegece.com.js**.

Aqui você pode encontrar os métodos disponíveis; você precisa substituir as variáveis `{%host%}` e `{%port%}` conforme necessário, exemplo: se você configurou seu servidor sgcWebSocket para escutar a porta 80 no site `www.example.com`, você precisa configurar:

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
<script src="http://www.example.com:80/dataset.esegece.com.js"></script>
```

Open Connection

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
  var socket = new sgcws_dataset('ws://{%host%}:{%port%}');
</script>
```

Enviar mensagem

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
  var socket = new sgcws_dataset('ws://{%host%}:{%port%}');
  socket.send('Hello sgcWebSockets!');
</script>
```

Show Alert with Message Received

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
  var socket = new sgcws('ws://{%host%}:{%port%}');
  socket.on('sgcdataset', function(event)
  {
    alert(event.dataset);
  }
</script>
```

Exibir Alerta com Dataset Recebido

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
  var socket = new sgcws_dataset('ws://{%host%}:{%port%}');
  socket.on('sgcmessage', function(event)
  {
    alert(event.message);
  }
</script>
```



```
}
</script>
```

Mostra um Alert no OnSubscribe ou OnUnSubscribe de um canal

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
  var socket = new sgcws_dataset('ws://{%host%}:{%port%}');
  socket.on('sgcsubscribe', function(event)
  {
    alert('subscribed: ' + event.channel);
  }
  socket.on('sgcunsubscribe', function(event)
  {
    alert('unsubscribed: ' + event.channel);
  }
</script>
```

Show Alert OnConnect, OnDisconnect and OnError Events

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
  var socket = new sgcws_dataset('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>
```

Assinar Todas as Alterações do Dataset

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
  socket.subscribe_all();
</script>
```

UnSubscribe All Dataset Changes

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
  socket.unsubscribe_all();
</script>
```

Tratar Alterações de Dataset

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
var socket = new sgcws_dataset('ws://{%host%}:{%port%}');

socket.on('sgcdataset', function(evt){

if ((evt.channel == "sgc@dataset@new") || (evt.channel == "sgc@dataset@update")) {

... here you need to implement your own code insert/update records ...
}
else if (evt.channel == "sgc@dataset@delete") {

... here you need to implement your own code to delete records ...

}
});
</script>
```

Close Connection

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
    socket.close();
</script>
```

Obter Status da Conexão

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
    socket.state();
</script>
```

Protocolo Dataset | Replicar Tabela

Este modo tenta resolver um cenário comum em que uma **tabela é replicada** para todos os clientes conectados, **exemplo**, se você tem um servidor com uma **tabela de cotações de ações**, você quer **transmitir as mudanças das ações** para todos os **clientes**, mas não quer que um cliente possa se conectar ao seu banco de dados. Então, toda vez que houver uma **mudança** em qualquer cotação de ações, as **informações do registro serão transmitidas** para todos os clientes conectados. Cada **cliente lerá o registro e atualizará** sua própria tabela.

Você pode verificar na pasta Demos, o demo SQLite/MultipleDatabase.

Configure Dataset Server

Crie um novo Dataset Protocol Server e configure utilizando as seguintes propriedades

- **ApplyUpdates:** defina como **True**; toda vez que houver uma alteração, ela será transmitida (broadcast) aos clientes
- **AutoSynchronize:** definido como **True**, toda vez que um novo cliente se conecta ao servidor, o servidor enviará todos os registros (metadados e dados), de modo que o cliente obterá as informações mais recentes do servidor.
- **UpdateMode:** defina como **upWhereAll** ou **upWhereChanged**. A diferença é que o primeiro envia todos os campos de um registro e o segundo apenas os campos alterados em uma atualização.

```
TsgcWebSocketServer *oServer = new TsgcWebSocketServer();
TsgcWSPServer_Dataset *oProtocolDataset = new TsgcWSPServer_Dataset();
oProtocolDataset->Server = oServer;
oProtocolDataset->Dataset = <...your dataset...>;
oProtocolDataset->ApplyUpdates = true;
oProtocolDataset->AutoSynchronize = true;
oProtocolDataset->NotifyUpdates = true;
oProtocolDataset->UpdateMode = upWhereAll;
oServer->Port = 80;
oServer->Active = true;
```

Configure Dataset Client

Crie um novo Dataset Protocol Client e configure usando as seguintes propriedades

- **ApplyUpdates:** defina como **True**; toda vez que houver uma alteração, ela será enviada ao servidor.
- **AutoSubscribe:** definido como **True**, toda vez que um novo cliente se conecta ao servidor, o cliente assina automaticamente as atualizações, exclusões e novos registros.
- **UpdateMode:** defina como **upWhereAll** ou **upWhereChanged**. A diferença é que o primeiro envia todos os campos de um registro e o segundo apenas os campos alterados em uma atualização.

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSPClient_Dataset *oProtocolDataset = new TsgcWSPClient_Dataset();
oProtocolDataset->Client = oClient;
oProtocolDataset->Dataset = <...your dataset...>;
oProtocolDataset->ApplyUpdates = true;
oProtocolDataset->AutoSubscribe = true;
oProtocolDataset->NotifyUpdates = true;
oProtocolDataset->UpdateMode = upWhereAll;
oClient->Host = "127.0.0.1";
oClient->Port = 80;
oClient->Active = true;
```

Protocol Dataset | Notify Updates

Este modo tenta resolver um cenário em que **servidor** e **clientes** compartilham um **único banco de dados** (servidor e clientes estão conectados ao mesmo banco de dados físico) e os clientes querem ser notificados cada vez que outro cliente tiver feito qualquer mudança em um dataset.

Você pode conferir na pasta Demos, o exemplo SQLite/SingleDatabase.

Configure Dataset Server

Crie um novo Dataset Protocol Server e configure utilizando as seguintes propriedades

- **ApplyUpdates:** defina como **True**; toda vez que houver uma alteração, ela será transmitida (broadcast) aos clientes
- **AutoSynchronize:** defina como **False**, aqui não é necessário definir como true, porque o cliente está conectado ao mesmo banco de dados que o servidor.
- **UpdateMode:** defina como **upRefreshAll**.

```
TsgcWebSocketServer *oServer = new TsgcWebSocketServer();
TsgcWSPServer_Dataset *oProtocolDataset = new TsgcWSPServer_Dataset();
oProtocolDataset->Server = oServer;
oProtocolDataset->Dataset = <...your dataset...>;
oProtocolDataset->ApplyUpdates = true;
oProtocolDataset->AutoSynchronize = false;
oProtocolDataset->NotifyUpdates = true;
oProtocolDataset->UpdateMode = upRefreshAll;
oServer->Port = 80;
oServer->Active = true;
```

Configure Dataset Client

Crie um novo Dataset Protocol Client e configure usando as seguintes propriedades

- **ApplyUpdates:** defina como **True**; toda vez que houver uma alteração, ela será enviada ao servidor.
- **AutoSubscribe:** definido como **True**, toda vez que um novo cliente se conecta ao servidor, o cliente assina automaticamente as atualizações, exclusões e novos registros.
- **UpdateMode:** defina como **upRefreshAll**.

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSPClient_Dataset *oProtocolDataset = new TsgcWSPClient_Dataset();
oProtocolDataset->Client = oClient;
oProtocolDataset->Dataset = <...your dataset...>;
oProtocolDataset->ApplyUpdates = true;
oProtocolDataset->AutoSubscribe = true;
oProtocolDataset->NotifyUpdates = true;
oProtocolDataset->UpdateMode = upRefreshAll;
oClient->Host = "127.0.0.1";
oClient->Port = 80;
oClient->Active = true;
```

Protocol Files

Este protocolo permite enviar arquivos usando transporte binário WebSocket. Ele pode lidar com arquivos grandes com baixo uso de memória.

Features

- Padrão de mensagem **Publish/subscribe** para fornecer distribuição de mensagens um-para-muitos e desacoplamento de aplicações.
- **Acknowledgment** das mensagens enviadas.
- Implementa **QoS** (Quality of Service) para a entrega de arquivos.
- Opcionalmente pode solicitar **Authorization** para os arquivos recebidos.
- **Baixo** uso de memória.

Componentes

[TsgcWSPServer_Files](#): Componente VCL Server Protocol Files.

[TsgcWSPClient_Files](#): Componente VCL do protocolo cliente Files.

Classes

[TsgcWSMessageFile](#): o objeto que encapsula as informações do pacote de arquivo.

Most common uses

- **Enviar arquivos**
 - [Como Enviar Arquivos para o Servidor](#)
 - [Como Enviar Arquivos para Clientes](#)
- **Arquivos grandes**
 - [Como Enviar Arquivos Grandes](#)

TsgcWSPServer_Files

Este é o Componente do Protocolo Files para Servidor.

Introdução

Este é o Componente do Protocolo Files para Servidor. Você precisa soltar este componente no formulário e selecionar um Componente [TsgcWebSocketServer](#) utilizando a propriedade Server.

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Métodos — 3 métodos públicos.
- Events — 20 eventos que você pode tratar.

TsgcWSPClient_Files

Este é o Componente de Protocolo Files do Cliente.

Introdução

Este é o Componente do Protocolo Files para Cliente. Você precisa soltar este componente no formulário e selecionar um Componente [TsgcWebSocketClient](#) utilizando a propriedade Client.

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Métodos — 4 métodos públicos.
- Eventos — 19 eventos que você pode tratar.

TsgcWSMessageFile

Este objeto é passado como parâmetro toda vez que um evento de protocolo de arquivo é gerado.

Properties

- **BufferSize:** tamanho padrão do pacote.
- **Channel:** se especificado, este arquivo será enviado apenas a clientes assinados em um canal específico.
- **Method:** método interno.
- **FileId:** identificador de um arquivo; é único para todos os arquivos recebidos/enviados.
- **Data:** dados personalizados do usuário. Aqui o usuário pode definir qualquer texto.
- **FileName:** nome do arquivo.
- **FilePosition:** posição do arquivo em bytes.
- **FileSize:** Tamanho total do arquivo em bytes.
- **Id:** identificador de um pacote; é único para cada pacote.
- **QoS:** qualidade de serviço da mensagem.
- **Streaming:** para uso interno.
- **Text:** para uso interno.

Protocol Files | Como enviar arquivos ao servidor

Para enviar um arquivo ao servidor, basta chamar o método **SendFile** do protocolo Files e passar o **FileName** completo como argumento.

O arquivo recebido pelo servidor será salvo por padrão no mesmo diretório onde o executável do servidor está localizado ou no Path definido na propriedade 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");
```

Protocol Files | Como enviar arquivos aos clientes

Para enviar um arquivo a um cliente, basta chamar o método **SendFile** do Protocolo Files e passar o **Guid** da Conexão e o **FileName** completo como argumento. O Guid da conexão do cliente pode ser capturado no evento OnConnect do Protocolo Files do Servidor.

O arquivo recebido pelo cliente será salvo por padrão no mesmo diretório onde o executável do cliente está localizado ou no Path definido na propriedade 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");
```

Arquivos de Protocolo | Como Enviar Arquivos Grandes

Quando você quiser enviar arquivos grandes ao Servidor ou Cliente, por exemplo um Arquivo de alguns Gigabytes, você pode ter problemas de memória ao tentar carregar o arquivo completo. O Protocol Files permite enviar os arquivos em pacotes menores que, quando recebidos pelo outro peer, são remontados em um único arquivo. Basta utilizar o parâmetro **Size** do método **SendFile** para definir o Tamanho em Bytes de cada pacote individual.

```
// ... 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, qosLevel0, "");
```

Protocol Presence

O protocolo Presence permite que você saiba quem está inscrito em um canal, o que facilita criar aplicações de chat e saber quem está online, por exemplo: usuários de jogos, salas de chat, usuários visualizando o mesmo documento...

Features

- Por padrão, o usuário é **identificado por um nome**, mas isso pode ser **personalizado** passando mais dados: email, empresa, twitter...
- Eventos para **Autorizar** se um **Channel** pode ser criado, se um **membro** é permitido...
- Cada vez que um novo **membro entra** em um canal, todos os membros são **notificados**.
- **Publica mensagens** para todos os assinantes do canal.
- **Baixo** uso de memória.

Componentes

[TsgcWSPServer_Presence](#): Componente VCL do Protocolo Presence do Servidor.

[TsgcWSPClient_Presence](#): Componente VCL do Protocolo Presence para Cliente.

Classes

[TsgcWSPresenceMessage](#): o objeto que encapsula as informações do pacote de presença.

TsgcWSPServer_Presence

Este é o componente do Protocolo Presence do Servidor.

Introdução

Este é o Componente do Server Presence Protocol. Você precisa colocar este componente no form e seleccionar um Componente [TsgcWebSocketServer](#) utilizando a propriedade Server.

Referência

- Propriedades — 6 propriedades disponíveis no componente.
- Methods — 1 método público.
- Events — 17 eventos que você pode tratar.

TsgcWSPresenceMessage

Este objeto encapsula todas as mensagens internas trocadas pelo protocolo presence do servidor e do cliente.

TsgcWSPresenceMember

ID: identificador interno

Name: nome do membro, fornecido pelo cliente.

Info: informações adicionais do membro, fornecidas pelo cliente.

Data: TObject que pode ser usado para fins do servidor.

TsgcWSPresenceMemberList

Member[i]: membro de uma lista por índice

Count: número de membros da lista

TsgcWSPresenceChannel

Name: nome do canal, fornecido pelo cliente.

TsgcWSPresenceMsg

Text: mensagem de texto, fornecida pelo cliente ao chamar o método Publish

TsgcWSPresenceError

Code: valor inteiro que identifica o erro

Text: descrição do erro.

TsgcWSPClient_Presence

Este é o Componente de Protocolo de Presença do Cliente.

Introdução

Este é o componente Client Presence Protocol. Você precisa arrastar este componente para o formulário e selecionar um componente [TsgcWebSocketClient](#) utilizando a propriedade Client.

Referência

- Properties — 7 propriedades disponíveis no componente.
- Métodos — 5 métodos públicos.
- Events — 16 eventos que você pode tratar.

Protocolo Presence Javascript

O Presence Protocol Javascript do sgcWebSockets utiliza os arquivos **sgcWebSocket.js** e **presence.esegece.com.js**.

Aqui você pode encontrar os métodos disponíveis, você deve substituir as variáveis {%host%} e {%port%} conforme necessário, por exemplo: se você configurou o seu servidor sgcWebSocket para escutar a porta 80 no site www.example.com, você precisa configurar:

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
<script src="http://www.example.com:80/presence.esegece.com.js"></script>
```

Open Connection

```
<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>
```

Novo membro após a conexão

```
<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>
```

Assinar o canal 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>
```

Cancelar assinatura do canal 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>
```


Publicar Mensagem no canal 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>
```

Receber Mensagem

```
<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>
```

Obter todos os membros conectados

```
<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>
```

Exibir Alerta quando Membros assinam/cancelam assinatura

```
<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>
```

Show Alert OnConnect, OnDisconnect and OnError Events

```
<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>
```

Close Connection

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  socket.close();
</script>
```

Protocol E2EE

End-to-End Encryption (E2EE) significa que as mensagens são criptografadas no dispositivo do remetente e podem ser descriptografadas apenas nos dispositivos do destinatário. O servidor roteia os pacotes, mas não consegue ler o conteúdo em texto puro.

Este tópico explica o fluxo técnico para:

- **Mensagens diretas (1:1)** entre dois peers.
- **Mensagens de grupo** utilizando um modelo de sender-key com rotação de chave baseada em associação.

Blocos de construção criptográficos

- **Identidade / acordo de chaves:** Elliptic Curve Diffie-Hellman (ECDH) P-256.
- **Criptografia simétrica:** AES-256-GCM.
- **Derivação de chave:** HKDF-SHA-256 para derivar chaves de criptografia a partir de segredos compartilhados.
- **Autenticação de mensagem:** tag de autenticação AEAD (fornecida pelo AES-GCM).
- **Proteção contra replay:** nonce/IV por mensagem mais contadores de sequência do remetente.

Fluxo técnico de mensagens diretas (1:1)

1. **Public key discovery**
Cada peer publica uma chave de identidade pública. O servidor pode distribuir chaves públicas, mas as chaves privadas nunca deixam o dispositivo do cliente.
2. **Estabelecimento do segredo compartilhado**
O remetente e o destinatário realizam o ECDH (chave privada + chave pública do peer) e obtêm o mesmo segredo compartilhado.
3. **Derivação da chave de sessão**
O HKDF-SHA-256 deriva uma ou mais chaves simétricas (chave de criptografia, chave de cabeçalho opcional) a partir da saída do ECDH.
4. **Criptografia de mensagens**
O texto simples é criptografado com AES-256-GCM usando um nonce/IV exclusivo. A saída inclui o texto cifrado + tag de autenticação.
5. **Transporte**
O servidor encaminha os payloads criptografados e os metadados (por exemplo: id do remetente, id da chave, contador, timestamp) sem acesso ao texto puro.
6. **Descriptografia do destinatário**
O destinatário deriva a mesma chave de sessão, verifica a tag de autenticação e descriptografa. Qualquer adulteração causa falha de autenticação.

Fluxo técnico de mensagens de grupo (Sender Keys)

Para grupos, criptografar cada mensagem separadamente para cada membro é caro. Uma otimização comum é um design de **sender key**:

- Cada remetente mantém um **Sender Key State** por grupo.
- O estado contém uma **chain key** simétrica e um contador de mensagens.
- Para cada mensagem de grupo enviada, o remetente deriva uma chave de mensagem de uso único a partir da chave de cadeia e avança a cadeia (hash ratchet).
- A chave da mensagem criptografa o payload com AES-256-GCM.

Como uma sender key é distribuída

1. Quando um remetente entra em um grupo, ele cria um estado de chave de remetente novo.
2. O estado da sender key é compartilhado com cada membro atual através de sessões criptografadas 1:1 existentes (E2EE par a par).
3. Após a distribuição, os payloads normais de grupo utilizam o caminho rápido de sender-key (uma única criptografia por mensagem).

Alterações de associação e rotação de sender key

Para preservar o sigilo direto e reverso, as chaves de remetente do grupo devem rotacionar em eventos de associação:

- **Usuário entra no grupo:** rotacione as sender keys para que o novo membro não possa descriptografar o histórico mais antigo, a menos que seja explicitamente compartilhado.
- **Usuário sai ou é removido:** rotacione as sender keys imediatamente para que o membro removido não possa descriptografar o novo tráfego.
- **Rotação periódica:** a rotação opcional baseada em tempo ou contagem de mensagens reduz o impacto do comprometimento de uma chave.

Sequência de rotação típica:

1. Cria um novo estado de sender key (novo key id, nova chain key).
2. Distribua-a apenas para os membros atualmente autorizados através de canais E2EE par a par (pairwise).
3. Inicia a criptografia de novas mensagens de grupo com o novo id de chave.
4. Aceitar o id de chave antiga somente durante uma curta janela de transição, depois aposentá-lo.

Propriedades de segurança

- **Confidencialidade:** apenas clientes com chaves válidas podem descriptografar.
- **Integridade e autenticidade:** a verificação AEAD detecta adulteração e textos cifrados forjados.
- **Cegueira do servidor:** os servidores de retransmissão não retêm chaves em texto simples para o conteúdo das mensagens.
- **Proteção pós-associação:** chaves rotacionadas impedem que ex-membros recebam mensagens futuras do grupo.

Componentes

[TsgcWSPServer_E2EE](#): Componente do protocolo servidor E2EE. Ele encaminha mensagens criptografadas entre clientes sem conhecer o conteúdo das mensagens.

[TsgcWSPClient_E2EE](#): Componente do Client Protocol E2EE. Ele gerencia a troca de chaves, a criptografia e a descriptografia nos dispositivos peer.

TsgcWSPServer_E2EE

O componente de protocolo de servidor E2EE roteia mensagens diretas e de grupo criptografadas entre clientes, mantendo os payloads opacos.

Introdução

O componente de protocolo do servidor E2EE roteia mensagens diretas e em grupo criptografadas entre clientes, mantendo os payloads opacos. Solte-o em um formulário, atribua um [TsgcWebSocketHTTPServer](#) (ou outro componente de servidor) à propriedade **Server** e configure as opções E2EE. A implementação reside na unit **sgcWebSocket_Protocol_E2EE_Server**.

Configuração

Para utilizar o componente:

- Defina **Server** como uma instância configurada de **TsgcWebSocketHTTPServer**.
- Configure o tratamento de confirmações em **E2EE_Options.Ack**.
- Escolha como as novas chaves públicas são tratadas com **E2EE_Options.PublicKeys.ReceiveNewPublicKey**.
- Defina **Server.Active := True** para começar a escutar.

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Métodos — 0 métodos públicos.
- Events — 8 eventos que você pode tratar.

TsgcWSPClient_E2EE

O componente de protocolo cliente E2EE adiciona criptografia de ponta a ponta sobre uma conexão WebSocket, incluindo mensagens diretas criptografadas e mensagens de grupo criptografadas.

Introdução

O componente de protocolo cliente E2EE adiciona criptografia de ponta a ponta sobre uma conexão WebSocket, incluindo mensagens diretas criptografadas e mensagens de grupo criptografadas. Solte o componente em um formulário, atribua um [TsgcWebSocketClient](#) à propriedade **Client** e configure as opções E2EE antes de conectar. A implementação reside na unit **sgcWebSocket_Protocol_E2EE_Client**.

Configuração

Para utilizar o componente:

- Defina **Client** como uma instância configurada de **TsgcWebSocketClient**.
- Atribua um identificador de usuário único utilizando **E2EE_Options.UserId**.
- Habilite as confirmações de mensagem com **E2EE_Options.Ack.RcvDirectMessage** e **E2EE_Options.Ack.RcvGroupMessage** quando você quiser callbacks de estado de entrega.
- Defina **Client.Active := True** para abrir a conexão WebSocket.

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Métodos — 9 métodos públicos.
- Eventos — 19 eventos que você pode tratar.

WebSocket APIs

Há várias implementações baseadas em WebSockets: finanças, publicação de mensagens, queues... o `sgcWebSockets` implementa as APIs mais importantes baseadas no protocolo WebSocket. Para usar uma API, basta anexar o componente da API ao cliente e todas as mensagens serão tratadas pelo componente da API (apenas um componente de API pode ser anexado a um cliente).

APIs do Cliente

API	Descrição
Binance	é um exchange de criptomoedas internacional multilíngue.
Binance Futures	permite que você se conecte aos Market Streams WebSocket / REST do Binance
Coinbase	A Coinbase é uma exchange de cripto baseada nos EUA. Negocie Bitcoin (BTC), EUR e GBP. Suporte para WebSocket API e REST API.
SignalR	é uma biblioteca para desenvolvedores ASP.NET que facilita o desenvolvimento em tempo real usando WebSockets como transporte.
SignalRCore	ASP.NET Core SignalR é uma biblioteca de código aberto que simplifica a adição de tempo real a aplicações.
SocketIO	é uma biblioteca JavaScript para aplicações web em tempo real. Ela habilita a comunicação em tempo real entre clientes web e servidores.
Kraken	é uma exchange de criptomoedas baseada nos EUA.
Kraken Futures	permite que você se conecte aos dados de mercado WebSocket / REST do Kraken
Pusher	Pusher é uma plataforma fácil e confiável com mensagens pub/sub flexíveis, listadas na Amazon.
Bitfinex	A Bitfinex é uma das maiores e mais avançadas plataformas de trading de cripto, onde você pode comprar e vender Bitcoin, Ethereum, Ripple, EOS, Bitcoin Cash, Iota, NEO, Litecoin, e mais.
Bitstamp	A Bitstamp é uma das exchanges de cripto mais antigas do mundo, dando suporte desde 2011.
Huobi	é um exchange de criptomoedas internacional multilíngue.
Cex	é uma exchange de criptomoedas e antigo provedor de mineração de Bitcoin na China.
Cex Plus	A CEX.IO Exchange Plus é a plataforma definitiva de negociação de cripto, com ferramentas avançadas de negociação
Bitmex	é uma exchange de criptomoedas e plataforma de negociação de derivativos.
3Commas	É um bot de negociação de criptomoedas.
Kucoin	é uma exchange de criptomoedas que permite que você compre, venda e armazene Bitcoin, ETH, KCS, SHIB, DOGE
Kucoin Futures	permite que você se conecte aos Servidores Kucoin Futures (WebSocket e REST)
OKX	anteriormente conhecida como OKEx, é uma das maiores exchanges de negociação de cripto
XTB	Negociação de FX e CFD, fornecendo acesso a mais de +2000 mercados financeiros
Discord	é uma das ferramentas de comunicação mais populares para jogos online e streaming
Bybit	exchange de criptomoedas e plataforma de negociação
OpenAI API	A OpenAI Realtime API permite interações multimodais de baixa latência, incluindo texto-to-audio, audio-to-text, speech-to-speech e transcrição em tempo real.
MEXC	Exchange centralizada de criptomoedas e plataforma de negociação; este componente suporta apenas REST
MEXC Futures	Exchange de criptomoedas e plataforma de trading centralizada, este componente suporta apenas WebSocket
Bitget	Exchange de criptomoedas e plataforma de negociação que suporta mercados de cripto
Gate.io	Exchange de criptomoedas que suporta negociação Spot e Futures com APIs WebSocket e REST

Deribit	Exchange de derivativos de criptomoedas que oferece negociação de futuros e
Crypto.com	Corretora de criptomoedas que suporta canais de Market e User com APIs Web
HTX	Exchange internacional de criptomoedas (anteriormente Huobi) com REST API p ação.

As APIs WebSocket podem ser registradas em **tempo de execução**, basta chamar o Método **RegisterAPI** e passar o componente da API como parâmetro.

Outras Client APIs

API	Descrição
Telegram	é um serviço de mensagens instantâneas e de voz sobre IP baseado em nuvem. C gens e trocar fotos, vídeos, stickers, áudio e arquivos de qualquer tipo.
Whatsapp	é um serviço de mensagens instantâneas e voz sobre IP, freeware, multiplataform ricana, disponível internacionalmente
RCON	é um protocolo de comunicação baseado em TCP/IP que permite que comandos vidor.
CryptoHopper	É um bot de trading de cripto e gerenciador de portfólio.
CryptoRobotics	É um robô de trading de cripto.

Server APIs

API	Descrição
RTCMultiConnection	RTCMultiConnection é uma biblioteca JavaScript WebRTC para aplicações de de tela, conferência de áudio/vídeo, compartilhamento de arquivos, stre
WebPush	O protocolo Web Push permite que aplicações web enviem notificações aplicação não está aberta ou ativa.
WebAuthn	API de servidor FIDO2/WebAuthn para autenticação sem senha usando passkeys.

API Binance

Binance

A Binance é uma exchange de criptomoedas internacional e multilíngue. Ela oferece algumas APIs para acessar dados da Binance. As seguintes APIs são suportadas:

1. **WebSocket streams:** permite que você faça subscribe a alguns métodos e obtenha dados em tempo real. Os eventos são enviados aos clientes pelo servidor para os assinantes. Usa o WebSocket como protocolo.
2. **UserData stream:** os clientes assinados obtêm detalhes da conta. Requer uma chave de API para autenticar e usa WebSocket como protocolo.
3. **REST API:** Requer uma API Key e Secret para autenticar e utiliza HTTPs como protocolo.
 1. [Market Data](#)
 2. [Account and Trading Data](#)
 3. [Wallet](#)
4. **Futures:** os WebSocket Futures Market Data Streams são suportados através da [API Binance Futures Client](#).

O cliente também suporta **Binance.us**; as seguintes APIs são suportadas:

1. **WebSocket streams:** permite que você faça subscribe a alguns métodos e obtenha dados em tempo real. Os eventos são enviados aos clientes pelo servidor para os assinantes. Usa o WebSocket como protocolo.
2. **UserData stream:** os clientes assinados obtêm detalhes da conta. Requer uma chave de API para autenticar e usa WebSocket como protocolo.
3. **REST API:** os clientes podem solicitar dados de mercado e de conta ao servidor. Requer uma API Key e Secret para autenticar e usa HTTPs como protocolo.

Properties

A API da Binance tem 2 tipos de métodos: públicos e privados. Os métodos públicos podem ser acessados sem autenticação, por exemplo: obter preços de tickers. Alguns são privados e relacionados a dados do usuário; esses métodos exigem o uso das chaves de API da Binance.

- **ApiKey:** você pode solicitar uma nova api key na sua conta binance, basta copiar o valor para esta propriedade.
- **ApiSecret:** O API secret é necessário apenas para a REST_API; a api websocket requer apenas a ApiKey para alguns métodos.
- **TestNet:** se habilitado, conectará à Binance Demo Account (por padrão false).
 - **HTTPLogOptions:** armazena em um arquivo de texto um log das requisições HTTP
 - **Enabled:** se habilitado, armazenará todas as requisições HTTP da WebSocket API.
 - **FileName:** caminho completo do nome do arquivo onde os logs serão armazenados
 - **REST:** armazena em um arquivo de texto um log das requisições da REST API
 - **Enabled:** se habilitado, armazenará todas as Requisições HTTP da REST API.
 - **FileName:** caminho completo do nome do arquivo onde os logs serão armazenados.
- **UserStream:** se habilitado, o cliente receberá notificações sobre Atualizações de Account, Orders ou Balance (por padrão true).
- **BinanceUS:** se habilitado, conectará aos Servidores Binance.us (em vez dos servidores Binance.com, que é o padrão).
- **ListenKeyOnDisconnect:** esta propriedade especifica o que fazer quando o cliente se desconecta dos servidores Binance com um ListenKey ativo.
 - **blkodDeleteListenKey:** Exclui o ListenKey ativo fazendo uma requisição HTTP aos servidores Binance (este é o padrão).
 - **blkodClearListenKey:** Não exclui o ListenKey dos servidores Binance e apenas limpa o valor do campo.
 - **blkodDoNothing:** não faz nada, portanto, na próxima vez que se conectar à Binance tentará usar o mesmo ListenKey.
- **UseCombinedStreams:** se habilitado, combinará os streams da seguinte forma: `{"stream": "<streamName>", "data": <rawPayload>}` (por padrão desabilitado)

Most common uses

- **WebSockets API**
 - [Como Conectar à WebSocket API](#)
 - [Como Assinar um Canal WebSocket](#)
- **REST API**
 - [Como obter dados de mercado](#)
 - [Como usar a Private REST API](#)
 - [Como negociar Spot](#)
 - [Private Requests Time](#)
 - [Withdraw](#)

WebSocket Stream API

O endpoint base é `wss://stream.binance.com:9443`, o cliente pode assinar / cancelar a assinatura de eventos após uma conexão bem-sucedida.

Os seguintes métodos de Assinatura / Cancelamento de assinatura são suportados.

Method	Parâmetros	Descrição
AggregateTrades	Symbol	envia informações de trade agregadas para uma única taker order
Trades	Symbol	envia informações brutas de trade; cada trade tem um comprador e um vendedor únicos
KLine	Symbol, Interval	envia atualizações dos klines/candlestick atuais a cada segundo, minuto, hora...
MiniTicker	Symbol	Estatísticas de mini-ticker de janela deslizante de 24h. Estas NÃO são as estatísticas do dia UTC, mas uma janela deslizante de 24h das últimas 24h.
AllMiniTickers		Estatísticas de mini-ticker de janela móvel de 24h para todos os símbolos que mudaram, em um array. Estas NÃO são as estatísticas do dia UTC, mas uma janela móvel de 24h para as 24h anteriores. Observe que apenas os tickers que mudaram estarão presentes no array.
Ticker	Symbol	Estatísticas de ticker de janela deslizante de 24h para um único símbolo. Estas NÃO são as estatísticas do dia UTC, mas uma janela deslizante de 24h para as 24h anteriores.
AllMarketTickers		Estatísticas de ticker de janela deslizante de 24h para todos os símbolos que mudaram, em um array. Estas NÃO são as estatísticas do dia UTC, mas uma janela deslizante de 24h para as 24h anteriores. Observe que apenas os tickers que mudaram estarão presentes no array.
BookTicker	Symbol	Envia qualquer atualização do melhor preço ou quantidade de bid ou ask em tempo real para um símbolo especificado.
AllBookTickers		Envia qualquer atualização no preço ou na quantidade do melhor bid ou ask em tempo real para todos os símbolos.
PartialBookDepth	Symbol, Depth	Os <levels> bids e asks principais, enviados a cada segundo. Os <levels> válidos são 5, 10 ou 20.
DiffDepth	Symbol	Atualizações de preço e profundidade de quantidade do livro de ordens usadas para gerenciar um livro de ordens localmente.

Após uma assinatura / cancelamento de assinatura bem-sucedido, o cliente recebe uma mensagem sobre isso, onde `id` é o resultado do método `Subscribed` / `Unsubscribed`.

```
{
  "result": null,
  "id": 1
}
```

User Data Stream API

Requer uma ApiKey válida obtida da sua conta binance, e a ApiKey deve ser definida na propriedade Binance.ApiKey do componente.

Os dados a seguir são enviados ao cliente toda vez que há uma alteração. Não há necessidade de assinar nenhum método; isso é feito automaticamente se você definir uma ApiKey válida.

Method	Descrição
Account Update	O estado da conta é atualizado com o evento outboundAccountInfo.
Balance Update	A atualização de saldo ocorre durante o seguinte: • Depósitos ou saques da conta • Transferência de fundos entre contas (por exemplo, de Spot para Margin)
Order Update	As ordens são atualizadas com o evento executionReport.

REST API

O endpoint base é: <https://api.binance.com>. Todos os endpoints retornam um objeto ou array JSON. Os dados são retornados em ordem ascendente. Mais antigos primeiro, mais recentes por último.

Acesso às Opções da REST API, utilizando a propriedade REST_API.BinanceOptions.

EndPoints da API Pública

Esses endpoints podem ser acessados sem nenhuma autorização.

General EndPoints

Method	Parâmetros	Descrição
Ping		Testa a conectividade com a Rest API.
GetServerTime		Testar a conectividade com a Rest API e obter a hora atual do servidor.
GetExchangeInformation		Regras de negociação atuais do exchange e informações do símbolo

Market Data EndPoints

Method	Parâmetros	Descrição
GetOrderBook	Symbol	Obtém o Order Book.
GetTrades	Symbol	Obter negociações recentes

GetHistorical-Trades	Symbol	Obtém negociações mais antigas.
GetAggregate-Trades	Symbol	Obtém negociações agregadas e compactadas. Negociações que são executadas ao mesmo tempo, da mesma ordem, com o mesmo preço terão a quantidade agregada.
GetKLines	Symbol, Interval	Barras de Kline/candlestick de um símbolo. As Klines são identificadas exclusivamente pelo seu open time.
GetAveragePrice	Symbol	Preço médio atual de um símbolo.
Get24hrTicker	Symbol	Estatísticas de variação de preço da janela móvel de 24 horas. Tenha cuidado ao acessar isto sem um símbolo.
GetPriceTicker	Symbol	Último preço de um símbolo.
GetPriceTickers	Symbols	Preço mais recente para um array de símbolos. Exemplo: ["BTCUSDT","BNBUSDT"]
GetBookTicker	Symbol	Melhor preço/quantidade no order book para um símbolo ou símbolos.
GetUIKLines	Symbol, Interval	Barras Kline/candlestick para um símbolo. A resposta é similar a GetKLines, otimizada para apresentação de gráficos de candlestick.
GetRollingWindowTicker	Symbol, Symbols, WindowSize	Estatísticas de variação de preço em janela móvel. Nota: o WindowSize padrão é 1d se não for especificado.
GetTradingDay-Ticker	Symbol, Symbols, Type	Estatísticas de variação de preço de um dia de negociação.

Private API EndPoints

Requer uma APIKey e uma APISecret para ser autorizado pelo servidor.

Account Data EndPoints

Method	Parâmetros	Descrição
NewOrder	Symbol, Side, Type	Envia uma nova ordem.
PlaceMarketOrder	Side, Symbol, Quantity	Coloca uma Nova Market Order
PlaceMarket-QuoteOrder	Side, Symbol, QuoteOrderQty	Coloca uma Nova Ordem de Cotação de Mercado
PlaceLimitOrder	Side, Symbol, Quantity, LimitPrice	Coloca uma nova ordem limitada
PlaceStopOrder	Side, Symbol, Quantity, StopPrice, LimitPrice	Coloca uma Nova Ordem Stop
PlaceStopTrailingOrder	Side, Symbol, Quantity, TrailingDelta, LimitPrice	Coloca uma Nova Ordem Stop Trailing
PlaceTakeProfitOrder	Side, Symbol, Quantity, StopPrice, LimitPrice	Coloca uma Nova Take Profit Order
PlaceTakeProfitTrailingOrder	Side, Symbol, Quantity, TrailingDelta, LimitPrice	Coloca uma Nova Ordem Take ProfitTrailing
PlaceLimitMakerOrder	Side, Symbol, Quantity	Coloca uma nova Limit Market Order

TestNewOrder	Symbol, Side, Type	Testa a criação de nova ordem e a assinatura/recvWindow long. Cria e valida uma nova ordem, mas não a envia ao matching engine.
QueryOrder	Symbol	Verifica o status de uma ordem.
CancelOrder	Symbol	Cancela uma ordem ativa. Cancela uma ordem ativa. OrderId ou OrigClientOrderId deve ser enviado.
CancelAllOpenOrders	Symbol (opcional)	
GetOpenOrders		Obter todas as ordens abertas em um símbolo. Tenha cuidado ao acessar isto sem um símbolo.
GetAllOrders	Symbol	Obtém todas as ordens da conta; ativas, canceladas ou preenchidas.
NewOCO	Symbol, Side, Quantity, Price, StopPrice	Envia uma nova OCO
CancelOCO	Symbol	Cancela uma Lista de Ordens inteira
QueryOCO	Symbol	Recupera um OCO específico com base nos parâmetros opcionais informados
GetAllOCO		Recupera todas as OCO com base nos parâmetros opcionais fornecidos
GetOpenOCO		Obtém todas as OCO Abertas.
GetAccountInformation		Obtém as informações da conta atual.
GetAccountTradeList	Symbol	Obtém os trades de uma conta e símbolo específicos.
CancelReplaceOrder	Symbol, Side, Type, CancelReplaceMode	Cancela uma ordem existente e coloca uma nova ordem no mesmo símbolo.
NewOrderListOCO	Symbol, Side, Quantity, AboveType, BelowType	Coloca uma nova lista de ordens OCO.
NewOrderListOTO	Symbol, WorkingType, WorkingSide, WorkingQuantity, WorkingPrice, PendingType, PendingSide, PendingQuantity	Registra uma nova lista de ordens OTO (One-Triggers-the-Other).
NewOrderListOTOCO	Symbol, WorkingType, WorkingSide, WorkingQuantity, WorkingPrice, PendingSide, PendingAboveType, PendingBelowType, PendingQuantity	Coloca uma nova lista de ordens OTOCO (One-Triggers-a-One-Cancels-the-Other).
NewSOROrder	Symbol, Side, Type, Quantity	Coloca uma ordem utilizando Smart Order Routing (SOR).
TestSOROrder	Symbol, Side, Type, Quantity	Testar uma nova ordem utilizando Smart Order Routing (SOR). Cria e valida uma nova ordem, mas não a envia ao motor de correspondência.
GetOrderRateLimitUsage		Exibe a contagem de ordens atual do usuário para todos os intervalos.
GetPreventedMatches	Symbol	Exibe a lista de ordens que expiraram por causa de STP (Self Trade Prevention).
GetAllocations	Symbol	Recupera as alocações resultantes da colocação de ordens SOR.
GetAccountCommission	Symbol	Obter as taxas de comissão atuais da conta.

Convert EndPoints

Method	Parâmetros	Descrição
GetAllConvert-Pairs	FromAsset, ToAsset	Consulta todos os pares de tokens conversíveis e os respectivos limites superior/inferior dos tokens
GetConvertAssetInfo		Consulta as informações de precisão suportadas de um ativo
SendConvertQuoteRequest	FromAsset, ToAsset	Solicita uma cotação para os pares de tokens solicitados
AcceptConvertQuote	Quoteld	Aceita a cotação oferecida pelo ID da cotação.
GetConvertOrderStatus	OrderId ou Quoteld	Consulta o status da ordem por ID da ordem.
PlaceConvertLimitOrder	BaseAsset, QuoteAsset, Side, LimitPrice	Permite que os usuários coloquem uma limit order. baseAsset ou quoteAsset podem ser determinados através do endpoint exchangeInfo. O preço limite é definido de baseAsset para quoteAsset. Utiliza-se baseAmount ou quoteAmount.
CancelConvertLimitOrder	OrderId	Permite que os usuários cancelem uma ordem limite
GetConvertLimitOpenOrders		Permite que os usuários consultem todas as limit orders existentes
GetConvertTradeHistory	StartTime, EndTime	O intervalo máximo entre startTime e endTime é de 30 dias.

Wallet EndPoints

(*os endpoints de wallet só funcionam com o servidor de produção, não com o demo)

Method	Descrição
GetWalletSystemStatus	Obtém o status do sistema.
GetWalletAllCoinsInformation	Obtém informações de moedas (disponíveis para depósito e saque) para o usuário.
GetWalletDailyAccountSnapshot	Type: "SPOT", "MARGIN", "FUTURES" - O período de tempo da consulta deve ser menor que 30 dias - Suporta consulta apenas dentro do último mês - Se startTime e endTime não forem enviados, retorna os registros dos últimos 7 dias por padrão
SetWalletDisableFastWithdrawSwitch	Esta requisição desabilitará o switch fastwithdraw na sua conta. Você precisa habilitar a opção "trade" para a api key que solicita este endpoint.
SetWalletEnableFastWithdrawSwitch	Esta requisição habilitará o switch de fastwithdraw na sua conta. Você precisa habilitar a opção "trade" para a chave de api que faz requisições a este endpoint. Quando o Fast Withdraw Switch está ativado, a transferência de fundos para uma conta Binance será feita instantaneamente. Não há transação on-chain, nem ID de transação, nem taxa de saque.
WalletWithdraw	Submeter uma requisição de saque.
GetWalletDepositHistory	Obtém o histórico de depósitos.

GetWalletWithdrawHistory	Buscar histórico de saques (Withdraw).
GetWalletDepositAddress	Buscar endereço de depósito com rede.
GetWalletAccountStatus	Busca o detalhe do status da conta.
GetWalletAccountAPI-TradingStatus	Obtém os detalhes do status de trading da API da conta.
GetWalletDustLog	Retorna apenas os últimos 100 registros Retorna apenas os registros após 2020/12/01
GetWalletAssetsConvertedBNB	
WalletDustTransfer	Converte ativos de dust em BNB. Você precisa habilitar a permissão Spot & Margin Trading para a API Key que solicita este endpoint.
GetWalletAssetDividendRecord	Consulta o registro de dividendos de ativos.
GetWalletAssetDetail	Busca detalhes dos ativos suportados na Binance.
GetWalletTradeFee	Obter taxa de trade
WalletUserUniversalTransfer	Você precisa habilitar a opção Permits Universal Transfer para a API Key que requisita este endpoint. MAIN_UMFUTURE Transferência da conta Spot para a conta USDⓈ-M Futures ENUM de Type: • MAIN_CMFUTURE Transferência da conta Spot para a conta COIN-M Futures • MAIN_MARGIN Transferência da conta Spot para a conta Margin (cross) • UMFUTURE_MAIN Transferência da conta USDⓈ-M Futures para a conta Spot • UMFUTURE_MARGIN Transferência da conta USDⓈ-M Futures para a conta Margin (cross) • CMFUTURE_MAIN Transferência da conta COIN-M Futures para a conta Spot • CMFUTURE_MARGIN Transferência da conta COIN-M Futures para a conta Margin(cross) • MARGIN_MAIN Transferência da conta Margin (cross) para a conta Spot • MARGIN_UMFUTURE Transferência da conta Margin (cross) para USDⓈ-M Futures • MARGIN_CMFUTURE Transferência da conta Margin (cross) para COIN-M Futures • ISOLATEDMARGIN_MARGIN Transferência da conta isolated margin para a conta Margin(cross) • MARGIN_ISOLATEDMARGIN Transferência da conta Margin(cross) para a conta isolated margin • ISOLATEDMARGIN_ISOLATEDMARGIN Transferência da conta isolated margin para a conta isolated margin • MAIN_FUNDING Transferência da conta Spot para a conta Funding • FUNDING_MAIN Transferência da conta Funding para a conta Spot • FUNDING_UMFUTURE Transferência da conta Funding para a conta UMFUTURE • UMFUTURE_FUNDING Transferência da conta UMFUTURE para a conta Funding • MARGIN_FUNDING Transferência da conta MARGIN para a conta Funding • FUNDING_MARGIN Transferência da conta Funding para a conta Margin • FUNDING_CMFUTURE Transferência da conta Funding para a conta CMFUTURE • CMFUTURE_FUNDING Transferência da conta CMFUTURE para a conta Funding
GetWalletQueryUserUniversalTransferHistory	• fromSymbol deve ser enviado quando o type é ISOLATEDMARGIN_MARGIN e ISOLATEDMARGIN_ISOLATEDMARGIN • toSymbol deve ser enviado quando o type é MARGIN_ISOLATEDMARGIN e ISOLATEDMARGIN_ISOLATEDMARGIN • Suporta consulta apenas dos últimos 6 meses • Se startTime e endTime não forem enviados, retorna os registros dos últimos 7 dias por padrão
GetWalletFundingWallet	Atualmente suporta a consulta dos seguintes ativos de negócio : Binance Pay, Binance Card, Binance Gift Card, Stock Token
GetWalletUserAsset	Obter ativos do usuário, apenas para dados positivos.

GetWalletApiKeyPer- mission	

Eventos

As mensagens do Binance são recebidas no componente `TsgcWebSocketClient`; você pode utilizar os seguintes eventos:

OnConnect

Após uma conexão bem-sucedida ao servidor Binance.

OnDisconnect

Após uma desconexão do servidor Binance

OnMessage

Mensagens enviadas pelo servidor ao cliente são tratadas neste evento.

OnError

Se houver algum erro no protocolo, este evento será chamado.

OnException

Se houver uma exceção não tratada, este evento será chamado.

Além disso, há um evento específico no Componente da API Binance, chamado **OnBinanceHTTPException**, que é disparado cada vez que há um erro ao chamar uma Requisição HTTP (REST API ou WebSocket User Stream).

(*) Devido a mudanças nos servidores da Binance, versões do Indy anteriores ao Rad Studio 10.1 não conseguirão conectar aos Test Servers. Esse problema não afeta a edição Enterprise nem se a versão do Indy tiver sido atualizada para a mais recente.

Binance | Conectar à WebSocket API

Para conectar à API WebSocket da Binance, basta criar um novo cliente da API Binance e anexá-lo a um TsgcWebSocketClient.

Veja abaixo um exemplo:

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient(NULL);
TsgcWSAPI_Binance *oBinance = new TsgcWSAPI_Binance(NULL);
oBinance->Client = oClient;
oClient->Active = true;
```

Binance | Subscribe WebSocket Channel

A Binance oferece uma variedade de canais nos quais você pode assinar para obter atualizações em tempo real de dados de mercado, ordens... Veja abaixo um exemplo de como assinar um Ticker:

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient(NULL);
TsgcWSAPI_Binance *oBinance = new TsgcWSAPI_Binance(NULL);
oBinance->Client = oClient;
oBinance->SubscribeTicker("bnbbtc");

void OnMessage(TsgcWSConnection *Connection, const string aText)
{
    // here you will receive the ticker updates
}
```

Binance | Obter dados de mercado

A Binance oferece dados de mercado públicos por meio de REST Endpoints; quando você chama um desses endpoints, obtém um snapshot dos dados de mercado solicitados.

Os Market Data Endpoints não exigem autenticação, então estão disponíveis livremente para todos os usuários.

Exemplo: para obter um snapshot do ticker BNBBTC, faça a seguinte chamada:

```
TsgcWSAPI_Binance *oBinance = new TsgcWSAPI_Binance(this);  
ShowMessage(oBinance->REST_API->GetPriceTicker("BNBBTC"));
```

Binance | Private REST API

A REST API da Binance oferece endpoints públicos e privados. Os endpoints Privados exigem que as mensagens sejam assinadas para aumentar a segurança das transações.

Primeiro você deve fazer login em sua conta Binance e criar uma nova API; você obterá os seguintes valores:

- ApiKey
- ApiSecret

Esses campos devem ser configurados na propriedade Binance do componente cliente da Binance API. Uma vez configurado, você pode começar a fazer requisições privadas à REST API Pro da Binance

*Requisições privadas requerem que a sua máquina local tenha a hora local sincronizada; caso contrário, as requisições serão rejeitadas pelo servidor da Binance. Verifique o artigo a seguir sobre isso: [Binance Private Requests Time](#).

```
TsgcWSAPI_Binance *oBinance = new TsgcWSAPI_Binance(this);
oBinance->Binance->ApiKey = "<your api key>";
oBinance->Binance->ApiSecret = "<your api secret>";
ShowMessage(oBinance->REST_API->GetAccountInformation());
```

Binance | Trade Spot

A Binance permite que você faça trade spot utilizando sua API REST.

Configuração

Primeiro você deve criar uma **API Key** na sua conta binance e adicionar privilégios para negociar com Spot.

Uma vez feito isso, você pode iniciar o spot trading.

Primeiro, **defina sua ApiKey e seu ApiSecret** no Componente Binance Client; isto será usado para assinar as requisições enviadas ao servidor Binance.

Place an Order

Para colocar uma nova ordem, basta chamar o método **REST_API.NewOrder** do componente Binance Client.

Dependendo do tipo de ordem (market, limit...), a API requer mais ou menos campos.

Campos Obrigatórios

- **Symbol:** o símbolo do id do produto, exemplo: BNBBTC
- **Side:** BUY ou SELL
- **type:** o tipo da ordem
 - LIMIT
 - MARKET
 - STOP_LOSS
 - STOP_LOSS_LIMIT
 - TAKE_PROFIT
 - TAKE_PROFIT_LIMIT
 - LIMIT_MAKER

Campos Obrigatórios Adicionais com base no Type

- **LIMIT:** timeInForce, quantity, price
- **MARKET:** quantity ou quoteOrderQty
- **STOP_LOSS / TAKE_PROFIT:** quantity, stopPrice
- **STOP_LOSS_LIMIT / TAKE_PROFIT_LIMIT:** timeInForce, quantity, price, stopPrice
- **LIMIT_MAKER:** quantity, price

Ao enviar uma ordem, há 2 possibilidades:

1. **Successful:** a função NewOrder retorna a mensagem enviada pelo servidor binance.
2. **Error:** a exceção é retornada no evento OnBinanceHTTPException.

Place Market Order 1 BNBBTC

```
TsgcWSAPI_Binance *oBinance = new TsgcWSAPI_Binance(this);
oBinance->Binance->ApiKey = "<api key>";
oBinance->Binance->ApiSecret = "<api secret>";
ShowMessage(oBinance->REST_API->NewOrder("BNBBTC", "BUY", "MARKET", "", 1));
```

Place Limit Order 1 BNBBTC at 0.009260

```
TsgcWSAPI_Binance *oBinance = new TsgcWSAPI_Binance(this);
oBinance->Binance->ApiKey = "<api key>";
```

```
oBinance->Binance->ApiSecret = "<api secret>";  
ShowMessage(oBinance->REST_API->NewOrder("BNBBTC", "BUY", "LIMIT", "GTC", 1, 0, 0.009260));
```

-

Binance | Private Requests Time

Quando você faz uma requisição privada à Binance, a mensagem é assinada para aumentar a segurança das requisições. A mensagem pega o horário local e o envia dentro da mensagem assinada; se o horário local tiver uma diferença maior que 5 segundos em relação aos servidores da Binance, a requisição será rejeitada. Então, é importante verificar se seu horário local está sincronizado, você pode fazer isso utilizando o método de sincronização de horário do seu SO.

A lógica é a seguinte:

```
if (timestamp < (serverTime + 1000) && (serverTime - timestamp) <= recvWindow) {  
  // process request  
} else {  
  // reject request  
}
```

É recomendado utilizar um `recvWindow` pequeno, de 5000 ou menos! O máximo não pode ultrapassar 60000 milissegundos.

Você pode verificar a hora do servidor da Binance chamando o método **GetServerTime**, que retornará a hora do servidor da Binance

O **RecvWindow** tem o padrão **5000**; este valor pode ser aumentado utilizando a propriedade **REST_API.BinanceOptions.RecvWindow**.

Binance | Saque

A Binance permite que você utilize a Wallet API para submeter uma requisição de Withdraw, apenas os seguintes parâmetros são obrigatórios:

- Coin
- Address
- Amount

```
TsgcWSAPI_Binance *oBinance = new TsgcWSAPI_Binance(this);
oBinance->Binance->ApiKey = "<your api key>";
oBinance->Binance->ApiSecret = "<your api secret>";
ShowMessage(oBinance->REST_API->WalletWithdraw("BTC", "7213fea8e94b4a5593d507237e5a555b", 0.25));
```


API Binance Futures

Binance

A Binance é uma exchange internacional de criptomoedas multilíngue. Ela oferece algumas APIs para acessar os dados da Binance. Este componente permite que você obtenha os Market Streams WebSocket da Binance Futures.

<https://binance-docs.github.io/apidocs/futures/en>

<https://binance-docs.github.io/apidocs/delivery/en>

Futures Contracts

A API da Binance tem 2 tipos de métodos: públicos e privados. Os métodos públicos podem ser acessados sem autenticação, por exemplo: obter preços de ticker. Alguns são privados e relacionados a dados do usuário; esses métodos exigem o uso das API keys da Binance.

- **ApiKey:** você pode solicitar uma nova api key na sua conta binance, basta copiar o valor para esta propriedade.
- **ApiSecret:** O API secret é necessário apenas para a REST_API; a api websocket requer apenas a ApiKey para alguns métodos.
- **TestNet:** se habilitado, conectará à Binance Demo Account (por padrão false).
 - **HTTPLogOptions:** armazena em um arquivo de texto um log das requisições HTTP
 - **Enabled:** se habilitado, armazenará todas as requisições HTTP da WebSocket API.
 - **FileName:** caminho completo do nome do arquivo onde os logs serão armazenados
 - **REST:** armazena em um arquivo de texto um log das requisições da REST API
 - **Enabled:** se habilitado, armazenará todas as Requisições HTTP da REST API.
 - **FileName:** caminho completo do nome do arquivo onde os logs serão armazenados.
- **UserStream:** se habilitado, o cliente receberá notificações sobre Atualizações de Account, Orders ou Balance (por padrão true).
- **ListenKeyOnDisconnect:** esta propriedade especifica o que fazer quando o cliente se desconecta dos servidores Binance com uma ListenKey ativa.
 - **blkodDeleteListenKey:** Exclui a ListenKey ativa fazendo uma Requisição HTTP aos Servidores Binance (este é o padrão).
 - **blkodClearListenKey:** Não exclui a ListenKey dos Servidores Binance e apenas limpa o valor do campo.
 - **blkodDoNothing:** não faz nada, então da próxima vez que conectar ao Binance tentará utilizar a mesma ListenKey.
- **UseCombinedStreams:** se habilitado, combinará os streams da seguinte forma: `{"stream": "<streamName>", "data": <rawPayload>}` (por padrão desabilitado)

O cliente pode conectar a Binance Futures **USDT** ou **COIN**, defina qual contrato você quer negociar usando a propriedade **FuturesContracts**:

- **bfcUSDT:** conecta-se à API USD-M Futures.
- **bfcCOIN:** conecta à API COIN-M Futures.

O cliente pode se conectar às contas Production ou Demo da Binance. Se a propriedade **TestNet** estiver habilitada, ele se conectará à conta Demo; caso contrário, conectará aos servidores de produção da Binance.

WebSocket Stream API

O cliente pode assinar / cancelar a assinatura de eventos após uma conexão bem-sucedida. Os seguintes métodos de Assinatura / Cancelamento de assinatura são suportados.

Method	Parâmetros	Descrição

AggregateTrades	Symbol	Os Aggregate Trade Streams enviam informações de negociação agregadas para uma única ordem taker a cada 100 milissegundos.
MarkPrice	Symbol, UpdateSpeed	Mark price e funding rate para um único símbolo, enviados a cada 3 segundos ou a cada segundo.
AllMarkPrice	UpdateSpeed	Mark price e funding rate para todos os símbolos enviados a cada 3 segundos ou a cada segundo.
KLine	Symbol, Interval	O Kline/Candlestick Stream envia atualizações via push para as klines/candlestick atuais a cada 250 milissegundos (se existentes).
MiniTicker	Symbol	Estatísticas de mini-ticker de janela móvel de 24h para um único símbolo. Estas NÃO são as estatísticas do dia UTC, mas uma janela móvel de 24h do requestTime até 24h antes.
AllMiniTicker		Estatísticas do mini-ticker da janela móvel de 24h para todos os símbolos. Estas NÃO são as estatísticas do dia UTC, mas uma janela móvel de 24h a partir do requestTime até 24h antes. Observe que apenas os tickers que mudaram estarão presentes no array.
Ticker	Symbol	Estatísticas de ticker de janela móvel de 24h para um único símbolo. Estas NÃO são as estatísticas do dia UTC, mas uma janela móvel de 24h do requestTime até 24h antes.
AllMarketTickers		Estatísticas de ticker de janela deslizante de 24h para todos os símbolos. Estas NÃO são as estatísticas do dia UTC, mas uma janela deslizante de 24h do requestTime até 24h antes. Note que apenas os tickers que mudaram estarão presentes no array.
BookTicker	Symbol	Envia qualquer atualização do melhor preço ou quantidade de bid ou ask em tempo real para um símbolo especificado.
AllBookTickers		Envia qualquer atualização no preço ou na quantidade do melhor bid ou ask em tempo real para todos os símbolos.
LiquidationOrders	Symbol	Os Liquidation Order Streams enviam informações de ordens de liquidação forçada para um símbolo específico
AllLiquidationOrders		Os All Liquidation Order Streams enviam informações de ordens de liquidação forçada para todos os símbolos no mercado.
PartialBookDepth	Symbol, Depth	Bids e asks do topo, valores válidos são 5, 10 ou 20.
DiffDepth	Symbol	Bids e asks, enviados a cada 250 milissegundos, 500 milissegundos, 100 milissegundos ou em tempo real (se existente)

Após uma assinatura / cancelamento de assinatura bem-sucedido, o cliente recebe uma mensagem sobre isso, onde id é o resultado do método Subscribed / Unsubscribed.

```
{
  "result": null,
  "id": 1
}
```

User Data Stream API

Requer uma ApiKey válida obtida da sua conta binance, e a ApiKey deve ser definida na propriedade Binance.ApiKey do componente.

Os dados a seguir são enviados ao cliente toda vez que há uma alteração. Não há necessidade de assinar nenhum método; isso é feito automaticamente se você definir uma ApiKey válida.

Method	Descrição

Margin Call	Quando a razão de risco da posição do usuário está muito alta, este stream será enviado. Esta mensagem é usada apenas como informação de orientação de risco e não é recomendada para estratégias de investimento. No caso de um mercado altamente volátil, pode haver a possibilidade de que a posição do usuário tenha sido liquidada ao mesmo tempo em que este stream é enviado.
Balance and Position Update	A atualização de saldo ocorre durante o seguinte: - Quando o saldo ou a posição forem atualizados, este evento será enviado. - Quando a "FUNDING FEE" altera o saldo do usuário.
Order Update	Quando uma nova ordem é criada, a mudança de status da ordem enviará esse evento.

REST API

Todos os endpoints retornam um objeto ou array JSON. Os dados são retornados em ordem ascendente. Mais antigos primeiro, mais recentes por último.

EndPoints da API Pública

Esses endpoints podem ser acessados sem nenhuma autorização.

General EndPoints

Method	Parâmetros	Descrição
Ping		Testa a conectividade com a Rest API.
GetServerTime		Testar a conectividade com a Rest API e obter a hora atual do servidor.
GetExchangeInformation		Regras de negociação atuais do exchange e informações do símbolo

Market Data EndPoints

Method	Parâmetros	Descrição
GetOrderBook	Symbol	Obtém o Order Book.
GetTrades	Symbol	Obter negociações recentes
GetHistorical-Trades	Symbol	Obtém negociações mais antigas.
GetAggregate-Trades	Symbol	Obtém negociações agregadas e compactadas. Negociações que são executadas ao mesmo tempo, da mesma ordem, com o mesmo preço terão a quantidade agregada.
GetKLines	Symbol, Interval	Barras de Kline/candlestick de um símbolo. As Klines são identificadas exclusivamente pelo seu open time.
Get24hrTicker	Symbol	Estatísticas de variação de preço da janela móvel de 24 horas. Tenha cuidado ao acessar isto sem um símbolo.
GetPriceTicker	Symbol	Preço mais recente de um símbolo ou símbolos.
GetBookTicker	Symbol	Melhor preço/quantidade no order book para um símbolo ou símbolos.
GetMarkPrice	Symbol	Mark Price e Funding Rate

GetFundingRateHistory	Symbol	
GetOpenInterest	Symbol	Obtém o open interest atual de um símbolo específico.
GetOpenInterestStatistics	Symbol, Period	
GetTopTraderAccountRatio	Symbol, Period	
GetTopTraderPositionRatio	Symbol, Period	
GetGlobalAccountRatio	Symbol, Period	
GetTakerVolume	Symbol, Period	
GetContinuousKLines	Pair, ContractType, Interval	Barras de kline/candlestick para um tipo de contrato específico.
GetIndexPriceKLines	Pair, Interval	Barras de Kline/candlestick para o preço de índice de um par.
GetMarkPriceKLines	Symbol, Interval	Barras de Kline/candlestick para o mark price de um símbolo.
GetPremiumIndexKLines	Symbol, Interval	Barras de kline do índice premium de um símbolo.
GetFundingInfo		Obtém informações da taxa de financiamento para todos os símbolos.
GetPriceTickerViewV2	Symbol	Preço mais recente para um símbolo ou símbolos (V2).
GetIndexInfo	Symbol	Obter informações do índice.
GetAssetIndex	Symbol	Obtém o índice de ativos para o modo multi-assets.
GetConstituents	Symbol	Obter os constituintes do índice.
GetDeliveryPrice	Pair	Obtém o preço de entrega.
GetBasis	Pair, ContractType, Period	Obtém dados de base.

EndPoints da API Privada

Requer uma APIKey e uma APISecret para ser autorizado pelo servidor.

EndPoints de Conta e Negociações

Method	Parâmetros	Descrição
ChangePositionMode	DualPosition	Alterar o modo de posição do usuário (Hedge Mode ou One-way Mode) em TODOS os símbolos
GetCurrentPositionMode		Obtém o modo de posição do usuário (Hedge Mode ou One-way Mode) em TODOS os símbolos
NewOrder	Symbol, Side, PositionSide, Type	Envia uma nova ordem.
PlaceMarketOrder	Side, Symbol, Quantity	

PlaceLimitOrder	Side, Symbol, Quantity, Limit-Price	
PlaceStopOrder	Side, Symbol, Quantity, Stop-Price, LimitPrice	
PlaceTrailingStopOrder	Side, Symbol, Quantity, aActivationPrice, aCallbackRate	
QueryOrder	Symbol	Verifica o status de uma ordem.
CancelOrder	Symbol	Cancela uma ordem ativa. OrderId ou OrigClientOrderId deve ser enviado.
CancelAllOpenOrders	Symbol	
AutoCancelAllOpenOrders	Symbol, CountdownTimer	Cancela todas as ordens abertas do símbolo especificado ao final da contagem regressiva especificada.
QueryCurrentOpenOrder	Symbol	
GetOpenOrders	Symbol	Obter todas as ordens abertas em um símbolo. Tenha cuidado ao acessar isto sem um símbolo.
GetAllOrders	Symbol	Obtém todas as ordens da conta; ativas, canceladas ou preenchidas.
GetAccountBalance		
GetAccountInformation		Obtém as informações da conta atual.
ChangeInitialLeverage	Symbol, Leverage	Alterar a alavancagem inicial do usuário em um mercado de símbolo específico.
ChangeMarginType	Symbol, MarginType	
ModifyIsolatedPositionMargin	Symbol, Amount, Type	
GetPositionMarginChangeHistory	Symbol	
GetPositionInformation	Symbol	
GetAccountTradeList	Symbol	
GetIncomeHistory	Symbol	
GetNotionalLeverageBracket	Symbol	
TestNewOrder	Symbol, Side, PositionSide, Type	Testa a criação de nova ordem e a assinatura/recvWindow long. Cria e valida uma nova ordem, mas não a envia ao matching engine.
ModifyOrder	Symbol	Modificar uma ordem existente.
NewBatchOrders	BatchOrders	Coloca múltiplas ordens.
ModifyBatchOrders	BatchOrders	Modificar múltiplas ordens.
CancelBatchOrders	Symbol	Cancela múltiplas ordens.
GetOrderAmendment	Symbol	Obtém o histórico de modificações de ordens.

Countdown-CancelAll	Symbol, CountdownTime	Cancela todas as ordens abertas do símbolo especificado ao final da contagem regressiva especificada.
GetForceOrders	Symbol	Obtém as ordens de liquidação forçada do usuário.
GetADLQuantile	Symbol	Obtém a estimativa de quantil ADL para posições.
GetAccountBalanceV3		Obtém o saldo da conta de futuros (V3).
GetAccountInformationV3		Obtém as informações atuais da conta (V3).
GetPositionInformationV3	Symbol	Obtém as informações de posição atuais (V3).
GetCommissionRate	Symbol	Obtém a taxa de comissão do usuário.
GetAccountConfig		Obtém a configuração atual da conta.
GetSymbolConfig	Symbol	Obtém a configuração do símbolo.
GetOrderRateLimit		Obtém o rate limit de ordens do usuário.
GetApiTradingStatus	Symbol	Obtém os indicadores de regras quantitativas de negociação da API.
ChangeMultiAssetsMode	MultiAssetsMargin	Altera o modo multi-assets do usuário. Multi-Assets Mode: true; Single-Asset Mode: false.
GetMultiAssetsMode		Obtém o modo multi-assets atual do usuário.
SetFeeBurn	FeeBurn	Altera o status de fee burn de BNB do usuário.
GetFeeBurn		Obtém o status de queima de taxa BNB do usuário.
CreateListenKey		Inicia um novo user data stream. O stream será fechado após 60 minutos, a menos que um keepalive seja enviado.
KeepAliveListenKey		Mantém vivo um user data stream para evitar um timeout.
CloseListenKey		Fechar um user data stream.

Eventos

As mensagens de Binance Futures são recebidas no componente `TsgcWebSocketClient`, você pode utilizar os seguintes eventos:

OnConnect

Após uma conexão bem-sucedida ao servidor Binance.

OnDisconnect

Após uma desconexão do servidor Binance

OnMessage

Mensagens enviadas pelo servidor ao cliente são tratadas neste evento.

OnError

Se houver algum erro no protocolo, este evento será chamado.

OnException

Se houver uma exceção não tratada, este evento será chamado.

Além disso, há um evento específico no Componente da API Binance, chamado **OnBinanceHTTPException**, que é disparado cada vez que há um erro ao chamar uma Requisição HTTP (REST API ou WebSocket User Stream).

(*) Devido a alterações nos Servidores da Binance, as versões do Indy anteriores ao Rad Studio 10.1 não conseguirão conectar aos Servidores de Teste. Este problema não afeta a Enterprise Edition nem se a versão do Indy tiver sido atualizada para a mais recente.

API Binance Futures | Trade

A Binance permite que você negocie futuros usando sua REST API.

Configuração

Primeiro você deve criar uma **API Key** na sua conta binance e adicionar privilégios para negociar com Futures.

Uma vez feito isso, você pode começar a negociar com futuros.

Primeiro você deve selecionar se deseja negociar com futuros **USDT** ou **COIN**; há uma propriedade chamada `FuturesContracts` onde você pode definir qual contrato de futuros deseja negociar

Em seguida, **defina sua ApiKey e seu ApiSecret** no componente `Binance Futures Client`, eles serão utilizados para assinar as requisições enviadas ao servidor da Binance.

Place an Order

Para colocar uma nova ordem, basta chamar o método **REST_API.NewOrder** do componente `Binance Futures Client`.

Dependendo do tipo da ordem (market, limit...), a API requer mais ou menos campos.

Campos Obrigatórios

- **Symbol:** o símbolo do id do produto, exemplo: `BTCUSD_210326`
- **Side:** `BUY` ou `SELL`
- **type:** o tipo da ordem
 - `LIMIT`
 - `MARKET`
 - `STOP`
 - `TAKE_PROFIT`
 - `STOP_MARKET`
 - `TAKE_PROFIT_MARKET`
 - `TRAILING_STOP_MARKET`

Campos Obrigatórios Adicionais com base no Type

- **LIMIT:** `timeInForce`, `quantity`, `price`
- **MARKET:** `quantity`
- **STOP/TAKE_PROFIT:** `quantity`, `price`, `stopPrice`
- **STOP_MARKET/TAKE_PROFIT_MARKET:** `stopPrice`
- **TRAILING_STOP_MARKET:** `callbackRate`

Ao enviar uma ordem, há 2 possibilidades:

1. **Successful:** a função `NewOrder` retorna a mensagem enviada pelo servidor binance.
2. **Error:** a exceção é retornada no evento `OnBinanceHTTPException`.

API SocketIO

SocketIO

O Socket.IO é uma biblioteca JavaScript para aplicações web em tempo real. Ele habilita comunicação bidirecional em tempo real entre clientes web e servidores. Possui duas partes: uma biblioteca do lado do cliente que roda no navegador e uma biblioteca do lado do servidor para Node.js. Ambos os componentes têm uma API quase idêntica. Como o Node.js, é orientado a eventos.

Messages Types

0: open (Enviado pelo servidor quando um novo transporte é aberto (recheck))

1: close (Solicita o fechamento deste transporte, mas não encerra a conexão em si.)

2: ping (Enviado pelo cliente. O servidor deve responder com um pacote pong contendo os mesmos dados)
exemplo

o cliente envia: 2probe

server sends: 3probe

3: pong (Enviado pelo servidor para responder a pacotes de ping.)

4: mensagem de string (mensagem real, cliente e servidor devem chamar seus callbacks com os dados.)

example:

42/chat,[{"join","{room:1}"]

4 é o tipo de pacote de mensagem no protocolo engine.io

2 é o tipo EVENT no protocolo socket.io

/chat é o dado que é processado pelo socket.io

o socket.io disparará o evento "join"

passará os dados "room: 1". É possível omitir o namespace somente quando ele é /.

5: upgrade (Antes de o engine.io trocar de transporte, ele testa se o servidor e o cliente conseguem se comunicar por esse transporte. Se este teste for bem-sucedido, o cliente envia pacotes de upgrade que solicitam ao servidor que esvazie seu cache no transporte antigo e troque para o novo transporte.)

6: noop (Um pacote noop. Utilizado principalmente para forçar um ciclo de poll quando uma conexão WebSocket de entrada é recebida.)

Properties

API: especifica a versão do SocketIO:

ioAPI0: suporta servidores socket.io 0.* (selecionado por padrão)

ioAPI1: suporta servidores socket.io 1.*

ioAPI2: suporta servidores socket.io 2.*

ioAPI3: suporta servidores socket.io 3.*

ioAPI4: suporta servidores socket.io 4.*

Base64: se habilitado, as mensagens binárias são recebidas como base64.

HandShakeCustomURL: permite personalizar a URL para obter a sessão socket.io.

HandShakeTimestamp: habilite apenas se você quiser enviar timestamp como parâmetro quando uma nova sessão é solicitada (habilite esta propriedade se você tentar acessar um servidor python gevent-socketio).

HandShakeAuthToken: se o servidor exigir um token para autenticação, defina aqui o token de autenticação.

Namespace: permite definir um namespace ao conectar-se ao servidor.

Polling: desabilitando esta propriedade, o cliente se conectará diretamente ao servidor usando websocket como transporte.

Parameters: permite que você defina os parâmetros de conexão.

EncodeParameters: se habilitado, os parâmetros são codificados.

Métodos

Utilize o método WriteData para enviar mensagens ao servidor socket.io (seguindo a seção Message Types).

1. Chame o método "add user" com um parâmetro usando John como nome do usuário.

```
WriteData("42[\"add user\", \"John\"]");
```

Eventos

OnHTTPRequest

Antes de uma nova conexão websocket ser estabelecida, o servidor socket.io exige que o cliente abra uma nova conexão HTTP para obter um novo id de sessão. Em alguns casos, o servidor socket.io exige autenticação usando cabeçalhos HTTP. Você pode usar este evento para adicionar cabeçalhos HTTP personalizados, como autorização Basic ou autenticação por token Bearer.

OnAfterConnect

Este evento é chamado após a conexão socket.io ser bem-sucedida e o cliente poder enviar mensagens ao servidor. Aqui você pode assinar namespaces, por exemplo.

OnHTTPConnectionSSL

Quando um servidor WebSocket requer conexões seguras, você pode obter uma mensagem de erro como esta quando um cliente tenta conectar ao servidor:

Error connecting with SSL. error:XXXXXXXX:SSL routines:ssl3_read_bytes:tlsv1 alert protocol version

Este erro significa que seu cliente está tentando se conectar usando uma versão do TLS que não é suportada pelo servidor.

Para resolver este erro, você deve tratar OnSSLAFTERCreateHandler do componente cliente WebSocket e definir uma versão mais recente do TLS.

Por exemplo: aqui estamos definindo o TLS 1.2 como versão de protocolo.

```
void OnHTTPConnectionSSL(TObject *Sender; TIdSSLIOHandlerSocketBase *aSSLHandler)
{
    static_cast<tidsslhandlersocketopenssl*>(aSSLHandler)->SSLOptions->Method = sslvTLSv1_2;
}
```

API Coinbase

Coinbase

APIs suportadas

- [WebSockets API](#): conecta a um servidor websocket público e fornece atualizações de dados de mercado em tempo real.
- [REST API](#): A REST API tem endpoints para gerenciamento de conta e de ordens, bem como dados de mercado públicos.

Most common uses

- **WebSockets API**
 - [Como Conectar à WebSocket API](#)
 - [Como Assinar um Canal WebSocket](#)
- **REST API**
 - [Como obter dados de mercado](#)
 - [Como utilizar a REST API privada](#)
 - [Como registrar ordens](#)
 - [Como utilizar a conta SandBox](#)
 - [Private Requests Time](#)

WebSockets API

O feed WebSocket está disponível publicamente e fornece atualizações de dados de mercado em tempo real para ordens e trades. Dois endpoints são suportados em produção:

- **Market Data** é o feed que fornece atualizações tanto para ordens quanto para negociações. A maioria dos canais agora está disponível sem autenticação.
- **User Order Data** fornece atualizações para as ordens do usuário.

Você pode se inscrever nos seguintes canais:

Method	Arguments	Descrição
SubscribeHeartBeat		Pings do servidor em tempo real para manter todas as conexões abertas
SubscribeStatus	aProductId : id do produto	Envia todos os produtos e moedas em um intervalo predefinido
SubscribeCandles	aProductId : id do produto	Atualizações em tempo real sobre product candles
SubscribeTicker	aProductId : id do produto	Atualizações de preço em tempo real toda vez que ocorre um match
SubscribeTicker-Batch	aProductId : id do produto	Atualizações de preço em tempo real a cada 5000 milissegundos
SubscribeLevel2	aProductId : id do produto	Todas as atualizações e a maneira mais fácil de manter o snapshot do order book
SubscribeMarket-Trades	aProductId : id do produto	Atualizações em tempo real cada vez que um trade de mercado acontece

SubscribeUser	aProductId: id do produto	Envia apenas mensagens que incluem o usuário autenticado
SubscribeFutures-BalanceSummary		Atualizações em tempo real toda vez que o saldo de futuros de um usuário muda

Os canais User e FuturesBalanceSummary requerem autenticação, portanto primeiro solicite suas API keys em sua conta Coinbase e então defina os valores na propriedade Coinbase do componente:

- ApiKey
- ApiSecret

A autenticação resultará em alguns benefícios:

1. As mensagens em que você é uma das partes são expandidas e têm campos mais úteis
2. Você receberá mensagens privadas, como informações de ciclo de vida sobre as ordens stop que você colocou

REST API

Private Endpoints

Endpoints privados estão disponíveis para gerenciamento de ordens e gerenciamento de conta.

Antes de poder assinar qualquer requisição, você deve criar uma chave de API através do site da Coinbase Pro. A chave de API estará vinculada a um perfil específico. Ao criar uma chave, você terá 3 informações que deve lembrar:

- Key
- Secret
- Passphrase

A Key e o Secret serão gerados aleatoriamente e fornecidos pela Coinbase Pro; a Passphrase será fornecida por você para proteger ainda mais seu acesso à API. A Coinbase Pro armazena o hash com salt da sua passphrase para verificação, mas não pode recuperar a passphrase se você a esquecer.

Você pode restringir a funcionalidade das chaves de API. Antes de criar a chave, você deve escolher quais permissões gostaria que a chave tivesse. As permissões são:

- View - Permite a uma chave permissões de leitura. Isso inclui todos os endpoints GET.
- Transfer - Permite que uma chave transfira moeda em nome de uma conta, incluindo depósitos e saques. Habilite com cautela - as transferências por API key IGNORARÃO a autenticação de dois fatores.
- Trade - Permite que uma chave registre ordens, bem como recupere dados de negociação. Isso inclui POST /orders e vários endpoints GET.

Accounts

Method	Arguments	Descrição
ListAccounts		Obtém uma lista de contas de negociação a partir do perfil da chave de API.
GetAccount	aAccountId: id da conta	Informações de uma única conta. Utilize este endpoint quando você souber o account_id. A chave de API deve pertencer ao mesmo perfil que a conta.

Orders

Method	Arguments	Descrição
PlaceNewOrder	aOrder: classe que contém todos os campos possíveis de uma ordem	Coloca uma nova ordem. Utilize apenas se você precisar acessar opções avançadas de ordem.
PlaceMarketOrder	aSide: buy ou sell aProductId: id do produto aQuoteSize: A quantidade do segundo Asset no Trading Pair. aBaseSize: A quantidade do primeiro Asset no Trading Pair aClient_oid: ID da Ordem selecionado por você para identificar sua ordem	Coloca uma nova ordem a Mercado.
PlaceLimitOrder	aSide: buy ou sell aProductId: id do produto aQuoteSize: A quantidade do segundo Asset no Trading Pair. aBaseSize: A quantidade do primeiro Asset no Trading Pair aLimitPrice: limite de preço Client_oid: Order ID selecionado por você para identificar sua ordem	Coloca uma nova ordem Limit.
PlaceStopOrder	aSide: buy ou sell ProductId: id do produto aBaseSize: A quantidade do primeiro Ativo no Par de Negociação StopPrice: preço do stop aLimitPrice: limite de preço aStopDirection: loss ou entry Client_oid: Order ID selecionado por você para identificar sua ordem	Coloca uma nova Stop Order
CancelOrder	aOrderId: id da ordem	Cancela uma ordem previamente colocada. A ordem deve pertencer ao profile ao qual a chave de API pertence.
EditOrder	aOrderId: id da ordem aPrice: preço aSize: Quantidade	Edita uma ordem com um novo tamanho especificado ou um novo preço
EditOrderPreview	aOrderId: id da ordem aPrice: preço aSize: Quantidade	Faz uma prévia de uma requisição de edição de ordem com um novo tamanho ou um novo preço especificado.
ListOrders		Obtém uma lista de ordens filtradas por parâmetros de query opcionais (product_id, order_status, etc).
GetOrder	aOrderId: id da ordem	Obter uma única ordem pelo ID da ordem.
PreviewOrder		Pré-visualiza uma ordem.
ClosePosition	aOrderId: id da ordem aProductId: id do produto aSize: quantidade	Coloca uma ordem para fechar quaisquer posições abertas de um product_id especificado.

Market Data

Method	Arguments	Descrição
GetPublicProducts		Obter uma lista dos pares de moedas disponíveis para negociação.

GetPublicProduct	aProductId: id do produto	Obtém informações sobre um único produto pelo product ID.
GetPublicProductBook	aProductId: id do produto	Obtém uma lista de bids/asks para um único produto. O nível de detalhe exibido pode ser personalizado com o parâmetro limit.
GetPublicProductCandles	aProductId: id do produto aStart: início do intervalo de tempo aEnd: fim do intervalo de tempo aGranularity: O timeframe que cada candle representa.	Obtém as taxas de um único produto pelo ID do produto, agrupadas em buckets.
GetTrades	aProductId: id do produto	Obtém informações de snapshot por ID de produto sobre as últimas negociações (ticks) e o melhor bid/ask.
GetTime		Obtém o horário atual da Coinbase Advanced API.

Fills

Method	Arguments	Descrição
GetFillsByOrderId		Obter uma lista de fills filtrada por id de ordem
GetFillsByProductId		Obtém uma lista de fills filtrados por product id
GetFillsByTradeId		Obtém uma lista de fills filtrada por trade id

Convert

Method	Arguments	Descrição
CreateConvertQuote		Cria uma cotação de conversão entre moedas.
CommitConvertTrade		Confirma uma negociação de conversão.
GetConvertTrade		Obtém os detalhes do trade de conversão.

Fees

Method	Arguments	Descrição
GetTransactionSummary		Obtém o resumo das taxas de transação.

Products (Autenticado)

Method	Arguments	Descrição
ListProducts		Lista os produtos disponíveis.
GetProduct	aProductId: id do produto	Obtém um produto específico.
GetProductBook	aProductId: id do produto	Obtém o order book do produto.
GetProductCandles	aProductId: id do produto	Obter candles OHLCV do produto.
GetMarketTrades	aProductId: id do produto	Obtém negociações recentes do mercado.
GetBestBidAsk		Obtém os melhores preços de bid/ask.

Portfolios

Method	Arguments	Descrição
ListPortfolios		Lista todos os portfólios.
CreatePortfolio		Criar um novo portfólio.
DeletePortfolio		Excluir um portfólio.

GetPortfolioBreak-down		Obtém os detalhes do breakdown do portfólio.
MovePortfolioFunds		Move fundos entre portfólios.

Perpetuals

Method	Arguments	Descrição
GetPerpetualsPortfolioSummary		Obtém o resumo do portfólio de perpétuos.
ListPerpetualsPositions		Lista posições de perpétuos.
GetPerpetualsPosition		Obtém uma posição específica de perpétuos.

Coinbase | Conectar à WebSocket API

Para se conectar à WebSocket API da Coinbase, basta criar um novo cliente API da Coinbase e anexá-lo ao TsgcWebSocketClient. Veja abaixo um exemplo:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase();
oCoinbase->Client = oClient;
oClient->Active = true;
```


Coinbase | Subscribe WebSocket Channel

A Coinbase oferece uma variedade de canais onde você pode assinar para obter atualizações em tempo real de dados de mercado, ordens... Encontre abaixo um exemplo de como assinar um 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 | Obter Dados de Mercado

A Coinbase oferece dados públicos de mercado por meio de REST Endpoints; ao chamar um desses endpoints, você obterá um snapshot dos dados de mercado solicitados.

Os Market Data Endpoints não exigem autenticação, então estão disponíveis livremente para todos os usuários.

Exemplo: para obter um snapshot do livro de BTC-USD, faça a seguinte chamada

```
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase(this);  
ShowMessage(oCoinbase->REST_API->GetPublicProductBook("BTC-USD"));
```

Coinbase Pro | Private REST API

A REST API da Coinbase oferece endpoints públicos e privados. Os endpoints privados exigem que as mensagens sejam assinadas para aumentar a segurança das transações.

Primeiro você deve fazer login na sua conta Coinbase e criar uma nova API, você obterá os seguintes valores:

- ApiKey
- ApiSecret

Estes campos devem ser configurados na propriedade Coinbase do componente cliente da API Coinbase. Uma vez configurado, você pode começar a fazer requisições privadas à REST API da Coinbase

```
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase(this);
oCoinbase->Coinbase->ApiKey = "<your api key>";
oCoinbase->Coinbase->ApiSecret = "<your api secret>";
ShowMessage(oCoinbase->REST_API->ListAccounts);
```

Coinbase Pro | Private Requests Time

Quando você faz uma requisição privada à Coinbase, a mensagem é assinada para aumentar a segurança das requisições. A mensagem pega o horário local e o envia dentro da mensagem assinada; se o horário local tiver uma diferença maior que 30 segundos em relação aos servidores da Coinbase, a requisição será rejeitada. Por isso, é importante verificar se o seu horário local está sincronizado; você pode fazer isso utilizando o método de sincronização de horário do seu SO.

Você pode verificar o horário do servidor da Coinbase Pro chamando o método **GetTime**, que retornará o horário do servidor da Coinbase Pro

Coinbase Pro | Place Orders

Para colocar novas ordens na Coinbase, primeiro você precisa de suas API keys para acessar seus dados privados. Confira o seguinte artigo: [Como Utilizar a Private REST API](#).

Uma vez que você tenha configurado suas API keys, pode começar a colocar ordens

Market Order

Coloca uma nova Market Order, comprar 0.002 contratos de BTC-USD

```
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase(this);
oCoinbase->Coinbase->ApiKey = "your api key";
oCoinbase->Coinbase->ApiSecret = "your api secret";
oCoinbase->Coinbase->ApiPassphrase = "your passphrase";
ShowMessage(oCoinbase->REST_API>PlaceMarketOrder(coisBuy, "BTC-USD", 0.002
, 0
));
```

Limit Order

Coloque uma nova Ordem Limit, comprar 0.002 contratos de BTC-USD ao limite de preço de 10000

```
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase(this);
oCoinbase->Coinbase->ApiKey = "your api key";
oCoinbase->Coinbase->ApiSecret = "your api secret";
oCoinbase->Coinbase->ApiPassphrase = "your passphrase";
ShowMessage(oCoinbase->REST_API>PlaceLimitOrder(coisBuy, "BTC-USD", 0.002,
0,
10000));
```

Coinbase Pro SandBox Account

A Coinbase permite que você utilize uma conta SandBox onde você pode negociar sem fundos reais. Essa conta requer a criação de chaves de API diferentes das da conta de produção.

Para usar a conta SandBox, basta definir a propriedade **Coinbase.SandBox** como **true**, antes de fazer qualquer requisição à REST API.

```
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase(this);
oCoinbase->Coinbase->ApiKey = "your api key";
oCoinbase->Coinbase->ApiSecret = "your api secret";
oCoinbase->Coinbase->SandBox = true;
ShowMessage(oCoinbase->REST_API>ListAccounts);
```

API SignalRCore

SignalRCore

O ASP.NET Core SignalR é uma biblioteca de código aberto que simplifica a adição de funcionalidades web em tempo real a aplicações. A funcionalidade web em tempo real permite que o código do lado do servidor envie conteúdo aos clientes instantaneamente.

Bons candidatos para o SignalR:

- Aplicativos que requerem atualizações de alta frequência do servidor. Exemplos são jogos, redes sociais, votações, leilões, mapas e aplicativos de GPS.
- Dashboards e aplicações de monitoramento. Exemplos incluem dashboards corporativos, atualizações instantâneas de vendas ou alertas de viagem.
- Aplicações colaborativas. Aplicações de quadro branco e softwares de reunião de equipe são exemplos de aplicações colaborativas.
- Apps que exigem notificações. Redes sociais, e-mail, chat, jogos, alertas de viagem e muitos outros apps utilizam notificações.

O componente `sgcWebSockets` `SignalRCore` utiliza `WebSocket` como transporte para se conectar a um servidor `SignalRCore`; se este transporte não for suportado, um erro será gerado.

Hubs

`SignalRCore` utiliza hubs para comunicação entre clientes e servidores. `SignalRCore` fornece 2 protocolos de hub: protocolo de texto baseado em JSON e protocolo binário baseado em `MessagePack`. O componente `sgcWebSockets` implementa apenas o protocolo de texto JSON para comunicação com servidores `SignalRCore`.

Para configurar qual Hub o cliente utilizará, basta definir o nome do Hub na propriedade **`SignalRCore/Hub`** antes de o cliente se conectar ao servidor.

Conexão

Quando um cliente abre uma nova conexão com o servidor, envia uma mensagem de requisição que contém o protocolo de formato e a versão. O `sgcWebSockets` sempre envia o protocolo de formato como JSON. O servidor responderá com um erro se o protocolo não for suportado por ele; este erro pode ser tratado utilizando o evento **`OnSignalRCoreError`**, e se a conexão for bem-sucedida, o evento **`OnSignalRCoreConnect`** será chamado.

Quando um cliente se conecta a um servidor `SignalRCore`, ele pode enviar um `ConnectionId` que identifica o cliente entre sessões, de modo que, se você tiver uma desconexão, o cliente possa reconectar ao servidor passando o mesmo connection id anterior. Para obter um novo connection id, basta conectar normalmente ao servidor e você pode saber o `ConnectionId` utilizando **`OnBeforeConnectEvent`**. Se você quiser reconectar ao servidor e passar um connection id anterior, utilize o método **`ReConnect`** e passe **`ConnectionId`** como parâmetro.

Protocolo SignalRCore

O Protocolo SignalR é um protocolo para RPC bidirecional sobre qualquer transporte baseado em mensagens. Qualquer uma das partes na conexão pode invocar procedimentos na outra parte, e os procedimentos podem retornar zero ou mais resultados ou um erro. Exemplo: o cliente pode solicitar um método do servidor e o servidor pode solicitar um método do cliente. As seguintes mensagens são trocadas entre o servidor e os clientes:

- **`HandshakeRequest`**: o cliente envia ao servidor para acordar o formato da mensagem.
- **`HandshakeResponse`**: o servidor responde ao cliente uma confirmação da mensagem `HandshakeRequest` anterior. Contém um erro se o handshake falhou.
- **`Close`**: chamado pelo cliente ou servidor quando uma conexão é fechada. Contém um erro se a conexão foi fechada por causa de um erro.

- **Invocation:** o cliente ou servidor envia uma mensagem a outro peer para invocar um método com ou sem argumentos.
- **StreamInvocation:** o cliente ou o servidor envia uma mensagem a outro peer para invocar um método de streaming com ou sem argumentos. A Resposta será dividida em diferentes itens.
- **StreamItem:** é uma resposta de um StreamInvocation anterior.
- **Completion:** significa que uma invocation ou StreamInvocation anterior foi concluída. Pode conter um resultado se o processo foi bem-sucedido ou um erro se houver algum erro.
- **CancellInvocation:** cancela uma requisição StreamInvocation anterior.
- **Ping:** é uma mensagem para verificar se a conexão ainda está ativa.

SignalRCore Encoding

O SignalRCore permite que você use as seguintes codificações:

- **JSON:** atualmente, a única codificação suportada.
- **MessagePack**

Atualmente, apenas JSON é suportado, embora MessagePack possa ser utilizado codificando as mensagens enviadas por meio de uma biblioteca messagepack externa. Veja a seção MessagePack abaixo para mais informações.

A configuração do Protocolo de Codificação é definida na propriedade SignalRCore.Protocol. Por padrão, o valor é **srcpJSON**.

Autorização

A autenticação pode ser habilitada para associar um usuário a cada conexão e filtrar quais usuários podem acessar recursos. A autenticação é implementada utilizando Bearer Tokens: o cliente fornece um access token e o servidor valida este token e o utiliza para identificar o usuário.

Em Web APIs padrão, os bearer tokens são enviados em um Cabeçalho HTTP, mas ao usar websockets, o token é transmitido como um parâmetro de query string.

Os seguintes métodos são suportados:

srcRequestToken

Se a Autenticação estiver habilitada, o fluxo é:

1. Primeiro tenta obter um token válido do servidor. Abre uma conexão HTTP contra Authentication.RequestToken.URL e faz um POST utilizando os dados de User e Password.
2. Se a anterior for bem-sucedida, um token é retornado. Caso contrário, um erro é retornado.
3. Se um token for retornado, então abre uma nova conexão HTTP para negociar uma nova conexão. Aqui, o token é passado como um Cabeçalho HTTP.
4. Se o anterior for bem-sucedido, abre uma conexão websocket e passa o token como parâmetro de query string.

- **Authentication.Enabled:** se ativo, a autorização será usada antes de uma conexão websocket ser estabelecida.
- **Authentication.Username:** o nome de usuário fornecido ao servidor para autenticar.
- **Authentication.Password:** a palavra secreta fornecida ao servidor para autenticar.
- **Authentication.RequestToken.PostFieldUsername:** nome do campo para transmitir o username (depende da configuração, verifique a página javascript do http para ver qual nome é utilizado).
- **Authentication.RequestToken.PostFieldPassword:** nome do campo para transmitir a senha (depende da configuração, verifique a página javascript http para ver qual nome é utilizado).
- **Authentication.RequestToken.URL:** url onde o token é solicitado.
- **Authentication.RequestToken.QueryFieldToken:** nome do parâmetro de query string utilizado na conexão websocket.

srcSetToken

Aqui, você passa o token diretamente para o servidor SignalRCore (porque o token foi obtido de outro servidor).

- **Authentication.Enabled:** se ativo, a autorização será utilizada antes de uma conexão websocket ser estabelecida.
- **Authentication.SetToken.Token:** valor do token obtido.

O Access token pode ser enviado como parâmetro de query (esta é a opção padrão) ou enviado como cabeçalho HTTP como um Bearer Token. Use a propriedade `Authentication.TokenParam` para configurar este comportamento.

- **srctQuery:** o `access_token` é passado na query url da conexão websocket.
- **srctHeader:** o `access_token` é passado como um header http como um Bearer Token.

srcaBasic

Esta opção usa **Basic Authentication**, este método de autenticação requer a configuração do componente `SignalRCore` e do `TsgcWebSocketClient`.

Exemplo: se o servidor exige basic authentication e o nome de usuário é "user" e a senha é "secret", configure os componentes conforme mostrado abaixo.

```
// 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;
```

Comunicação entre Cliente e Servidor

Há três tipos de interações entre servidor e clientes:

Invocations

O Caller envia uma mensagem ao Callee e espera uma mensagem indicando que a invocação foi concluída e, opcionalmente, um resultado da invocação

Exemplo: o cliente invoca o método `SendMessage` e passa como parâmetros o nome de usuário e a mensagem de texto. Envia um `Invocation Id` para obter uma mensagem de resultado do servidor.

```
SignalRCore->Invoke("SendMessage", ARRAYOFCNST(("John", "Hello All.")), "id-000001");
void OnSignalRCoreCompletion(TObject *Sender, TSignalRCore_Completion *Completion)
{
    if (Completion->Error != "")
    {
        ShowMessage("Something goes wrong.");
    }
    else
    {
        ShowMessage("Invocation Successful!");
    }
}
```

Invocações Não Bloqueantes

O Caller envia uma mensagem ao Callee e não espera mais nenhuma mensagem para esta invocação. As invocações podem ser enviadas sem um valor de Invocation ID. Isto indica que a invocação é "non-blocking".

Exemplo: o cliente invoca o método `SendMessage` e passa como parâmetros o nome do usuário e a mensagem de texto. O cliente não espera nenhuma resposta do servidor sobre o resultado da invocação.

```
SignalRCore->Invoke("SendMessage", ARRAYOFCNST(("John", "Hello All.")));
```

Streaming Invocations

O Caller envia uma mensagem ao Callee e espera um ou mais resultados retornados pelo Callee, seguidos por uma mensagem indicando o fim da invocação.

Exemplo: o cliente invoca o método `Counter` e solicita 10 números com um intervalo de 500 milissegundos.

```
SignalRCore->InvokeStream("Counter", [10, 500], "id-000002");

void OnSignalRCoreStreamItem(TObject *Sender, TSignalRCore_StreamItem *StreamItem, bool &Cancel)
{
    DoLog("#stream item: " + StreamItem->Item);
}

void OnSignalRCoreCompletion(TObject *Sender, TSignalRCore_Completion *Completion)
{
    if (Completion->Error != "")
    {
        ShowMessage("Something goes wrong.");
    }
    else
    {
        ShowMessage("Invocation Successful!");
    }
}
```

Invocations

Para realizar uma única invocação, o Caller segue o seguinte fluxo básico:

```
void Invoke(const string aTarget, const Array of Const aArguments, const string aInvocationId)
void InvokeStream(const string aTarget, const Array of Const aArguments, const String aInvocationId)
```

Aloque um valor único de Invocation ID (string arbitrária, escolhida pelo Caller) para representar a invocação. Chame o método `Invoke` ou `InvokeStream` contendo o Target sendo invocado, Arguments e InvocationId (se você não enviar InvocationId, não obterá o resultado de conclusão).

Se a Invocation estiver marcada como não-bloqueante (consulte "Non-Blocking Invocations" abaixo), pare aqui e retorne imediatamente o controle à aplicação. Trate a mensagem `StreamItem` ou `Completion` com um Invocation ID correspondente.

```
SignalRCore->InvokeStream("Counter", [10, 500], "id-000002");

void OnSignalRCoreStreamItem(TObject *Sender, TSignalRCore_StreamItem *StreamItem, bool &Cancel)
{
    if (StreamItem->InvocationId == "id-000002")
    {
        DoLog("#stream item: " + StreamItem->Item);
    }
}

void OnSignalRCoreCompletion(TObject *Sender, TSignalRCore_Completion *Completion)
{
    if (Completion->InvocationId == "id-000002")
    {
        if (Completion->Error != "")
        {

```

```

    ShowMessage("Something goes wrong.");
}
else
{
    ShowMessage("Invocation Successful!");
}
}
}

```

Você pode realizar uma única invocação e aguardar a conclusão.

```

bool InvokeAndWait(const String aTarget, System::TVarRec *aArguments, string aInvocationId,
    TSignalRCore_Completion &Completion, const int aTimeout = 10000);
bool InvokeStreamAndWait(const String aTarget, System::TVarRec *aArguments, string aInvocationId,
    TSignalRCore_Completion &Completion, const int aTimeout = 10000);

```

Aloque um valor de Invocation ID único (string arbitrária, escolhida pelo Caller) para representar a invocação. Chame o método `InvokeAndWait` ou `InvokeStreamAndWait` contendo o Target sendo invocado, os Arguments e o `InvocationId`. O programa aguardará até que o evento de conclusão seja chamado ou o Time out seja excedido.

```

{
    TSignalRCore_Completion oCompletion;
    if (SignalRCore->InvokeStreamAndWait("Counter", ARRAYOFCNST((10, 500)), "id-000002", oCompletion))
    {
        DoLog("#invoke stream ok: " + oCompletion->Result);
    }
    else
    {
        DoLog("#invocke stream error: " + oCompletion->Error);
    }
}

void OnSignalRCoreStreamItem(TObject *Sender, TSignalRCore_StreamItem *StreamItem, bool &Cancel)
{
    if (StreamItem->InvocationId == "id-000002")
    {
        DoLog("#stream item: " + StreamItem->Item);
    }
}

```

Cancel Invocation

Se o cliente quiser parar de receber mensagens `StreamItem` antes que o Servidor envie uma mensagem `Completion`, o cliente pode enviar uma mensagem `CancelInvocation` com o mesmo `InvocationId` utilizado na mensagem `StreamInvocation` que iniciou o stream.

```

void OnSignalRCoreStreamItem(TObject *Sender, TSignalRCore_StreamItem *StreamItem, bool &Cancel)
{
    if (StreamItem->InvocationId == "id-000002")
    {
        Cancel = true;
    }
}

```

Resultados do Cliente

Uma Invocation só é considerada concluída quando a mensagem `Completion` é recebida. Se o cliente receber uma Invocation do servidor, o evento `OnSignalRCoreInvocation` será chamado.

```

void OnSignalRCoreInvocation(TObject *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.");
```

Close Connection

Enviado pelo cliente quando uma conexão é fechada. Contém um motivo de erro se a conexão foi fechada por causa de um erro.

```
SignalRCore->Close("Unexpected message").

// If the server close connection by any reason, OnSignalRCoreClose event will be called.
void OnSignalRCoreClose(TObject *Sender, TSignalRCore_Close *Close)
{
    DoLog("#closed: " + Close->Error);
}
```

Ping

O protocolo SignalR Hub suporta mensagens "Keep Alive" utilizadas para garantir que a conexão de transporte subjacente permaneça ativa. Essas mensagens ajudam a garantir:

Que proxies não fechem a conexão subjacente durante períodos ociosos (quando poucas mensagens estão sendo enviadas). Se a conexão subjacente for derrubada sem ser encerrada graciosamente, a aplicação é informada o mais rápido possível.

O comportamento keep alive é obtido chamando o método Ping ou habilitando o HeartBeat no cliente WebSocket. Se o servidor enviar um ping ao cliente, o cliente enviará automaticamente uma resposta e o evento OnSignalR-CoreKeepAlive será chamado.

```
void OnSignalRCoreKeepAlive(TObject *Sender)
{
    DoLog("#keepalive");
}
```

MessagePack

Na Codificação MsgPack do Protocolo SignalR, cada Mensagem é representada como um único array MsgPack contendo itens que correspondem às propriedades da mensagem do hub protocol fornecido. Os itens do array podem ser valores primitivos, arrays (por exemplo, argumentos de método) ou objetos (por exemplo, valor de argumento). O primeiro item do array é o tipo da mensagem.

Consulte a [documentação do MessagePack](#) para ver como codificar as mensagens enviadas.

Toda vez que uma nova mensagem é recebida, ela é despachada no evento OnSignalRCoreMessagePack. A mensagem pode ser acessada lendo o parâmetro Data Stream. O parâmetro JSON está vazio por padrão; se você converter a mensagem MessagePack em JSON, o componente processará a mensagem JSON como se a codificação estivesse utilizando JSON (de modo que os eventos OnSignalRCoreCompletion, OnSignalRCoreInvocation... serão despachados).

API SignalR

SignalR

O componente SignalR usa WebSocket como transporte para conectar a um servidor SignalR; se este transporte não for suportado, um erro será gerado.

O componente cliente SignalR tem uma propriedade chamada `SignalR` onde você pode definir os seguintes dados:

- **Hubs:** contém uma lista de hubs aos quais o cliente está se inscrevendo.
- **ProtocolVersion:** a versão do protocolo utilizada pelo cliente, suporta as versões de protocolo de 1.2 a 1.5
- **UserAgent:** user agent utilizado para conectar ao servidor SignalR.

O cliente suporta o envio de dados Text ou Binary.

Mensagens de Hubs

A Hubs API torna possível invocar métodos do servidor a partir do cliente e métodos do cliente a partir do servidor. O protocolo utilizado para conexão persistente não é rico o suficiente para permitir expressar a semântica de RPC (remote procedure call). Isso não significa, no entanto, que o protocolo utilizado para conexões de hub seja completamente diferente do protocolo utilizado para conexões persistentes. Em vez disso, o protocolo utilizado para conexões de hub é, em sua maior parte, uma extensão do protocolo para conexões persistentes.

Quando um cliente invoca um método do servidor, ele não envia mais uma string de fluxo livre como era o caso para conexões persistentes. Em vez disso, ele envia uma string JSON contendo todas as informações necessárias para invocar o método. Aqui está um exemplo de mensagem que um cliente enviaria para invocar um método do servidor:

```
WriteData("{\"H\":\"chathub\",\"M\":\"Send\",\"A\":[\"CBuilder Client\",\"Test message\"],\"I\":0}
```

O payload possui as seguintes propriedades:

I – identificador de invocação – permite que você associe respostas com requisições

H – o nome do hub

M – o nome do método

A – arguments (um array, pode estar vazio se o método não tiver nenhum parâmetro)

Se o argumento string tiver **aspas duplas**, substitua " por \"

Exemplo: se o argumento é `{\"test\":1}`, envie o argumento como `{\"test\\\":1}`

```
WriteData("{\"H\":\"chathub\",\"M\":\"Send\",\"A\":[\"{\\\"test\\\":1}\",\"I\":0}");
```

Autorização

A autenticação pode ser habilitada para associar um usuário a cada conexão e filtrar quais usuários podem acessar recursos. A autenticação é implementada utilizando Bearer Tokens: o cliente fornece um access token e o servidor valida este token e o utiliza para identificar o usuário.

Atualmente, apenas Bearer Tokens são suportados:

Aqui, você passa o token diretamente ao servidor Signal (porque o token foi obtido de outro servidor).

- **Authentication.Enabled:** se ativo, a autorização será utilizada antes de uma conexão websocket ser estabelecida.
- **Authentication.Authentication:** 2 tipos de autenticação são suportados: bearer token ou cookies. Ambos requerem uma forma externa de obter os valores necessários.
 - **BearerToken:** valor do token obtido.
 - **Cookie:** define o valor do cookie necessário.

```
TsgcWSAPI_Signal *oSignalR = new TsgcWSAPI_Signal(NULL);
oSignalR->SignalR->Enabled = true;
oSignalR->SignalR->Authentication = srcBearerToken;
oSignalR->SignalR->BearerToken->Token = "token here";
```

O componente tem os seguintes eventos:

OnSignalRConnect

Este evento é gerado quando o cliente se conecta com sucesso ao servidor.

OnSignalRDisconnect

Este evento é gerado quando o cliente é desconectado do servidor.

OnSignalRError

Este evento é chamado quando há um erro na conexão WebSocket.

OnSignalRMessage

O protocolo utilizado para a conexão persistente é bastante simples. As mensagens enviadas ao servidor são apenas strings brutas. Não há um formato específico em que elas devam estar. As mensagens enviadas ao cliente são mais estruturadas. As propriedades que você pode encontrar na mensagem são as seguintes:

C – id da mensagem, presente em todas as mensagens que não são KeepAlive
M – um array contendo os dados reais.

```
{"C": "d-9B7A6976-B, 2|C, 2", "M": ["Welcome!"]}
```

OnSignalRBinary

Este evento é chamado quando dados binários são recebidos do servidor.

OnSignalRResult

Quando um método do servidor é invocado, o servidor retorna uma confirmação de que a invocação foi concluída enviando o id da invocação ao cliente e — se o método retornou um valor — o valor de retorno, ou — se a invocação do método falhou — o erro.

Aqui estão exemplos de resultados de uma chamada de método do servidor:

```
{"I":"0"}
```

Um método void do servidor cujo identificador de invocação era "0" foi concluído com sucesso.

```
{"I":"0", "R":42}
```

Um método de servidor que retorna um número cujo identificador de invocação era "0" foi concluído com sucesso e retornou o valor 42.

```
{"I":"0", "E":"Error occurred"}
```

OnSignalRKeepAlive

Este evento é gerado quando uma mensagem KeepAlive é recebida do servidor.

API Kraken

Kraken

Visão geral

A WebSockets API oferece atualizações de dados de mercado em tempo real. WebSockets é um protocolo bidirecional que oferece os dados em tempo real mais rápidos, ajudando você a construir aplicações em tempo real. Os tipos de mensagem públicos apresentados abaixo não requerem autenticação. Mensagens de dados privados podem ser assinadas em um endpoint autenticado separado.

A Kraken também oferece uma REST API com dados públicos de mercado e dados privados do usuário (que exigem autenticação).

Configuração

A API privada requer a criação de uma chave de API a partir da sua conta Kraken.

A Kraken permite um ambiente de teste no protocolo WebSocket; habilite a propriedade Beta a partir da propriedade Kraken para usar este recurso beta.

APIs suportadas

- [WebSockets Public API](#): conecta-se a um servidor WebSocket público.
- [WebSockets Private API](#): conecta a um servidor WebSocket privado e requer uma API Key e API Secret para Autenticar contra o servidor.
- [REST Public API](#): conecta-se a um servidor REST público.
- [REST Private API](#): conecta a um servidor REST público e requer uma API Key e um API Secret para autenticar no servidor.

Kraken Examples

Como se conectar a um servidor WebSocket público

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient(NULL);
TsgcWSAPI_Kraken *oKraken = new TsgcWSAPI_Kraken(NULL);
oKraken->Client = oClient;
oClient->Active = true;
```

Como Conectar a um Servidor WebSocket Privado

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient(NULL);
TsgcWSAPI_Kraken *oKraken = new TsgcWSAPI_Kraken(NULL);
oKraken->Kraken->ApiKey = "your api key";
oKraken->Kraken->ApiSecret = "your api secret";
oKraken->Client = oClient;
oClient->Active = true;
```


Como Obter o Ticker da API REST

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient(NULL);
TsgcWSAPI_Kraken *oKraken = new TsgcWSAPI_Kraken(NULL);
oKraken->Client = oClient;
ShowMessage(oKraken->GetTicker(ARRAYOFCNST(("XBTUSD"))));
```

REST API Methods

Endpoints Públicos

Method	Arguments	Descrição
GetSystemStatus		Obtém o status atual do sistema.

Private Endpoints

Method	Arguments	Descrição
GetExtendedBalance		Obtém informações estendidas de saldo.
AmendOrder		Alterar uma ordem existente.
CancelAllOrders		Cancela todas as ordens abertas.
CancelAllOrdersAfter		Dead man's switch - cancela todas as ordens após o timeout.
EditOrder		Edita uma ordem existente.
AddOrderBatch		Adiciona várias ordens em lote.
CancelOrderBatch		Cancela múltiplas ordens em lote.
GetWithdrawalMethods		Obtém os métodos de saque disponíveis.
GetWithdrawalAddresses		Obtém os endereços de saque.

Como Obter o Saldo da Conta a partir da REST API

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient(NULL);
TsgcWSAPI_Kraken *oKraken = new TsgcWSAPI_Kraken(NULL);
oKraken->Kraken->ApiKey = "your api key";
oKraken->Kraken->ApiSecret = "your api secret";
oKraken->Client = oClient;
ShowMessage(oKraken->GetAccountBalance());
```

API Kraken | WebSockets Public API

Conexão

URL: `wss://ws.kraken.com` (v1) ou `wss://ws.kraken.com/v2` (v2, recomendado)

O componente agora suporta WebSocket API v2 através da propriedade **Version** (padrão: 2). Defina `Kraken.Version := 1` para utilizar o endpoint legado v1.

Assim que o socket estiver aberto, você pode assinar um canal público enviando uma mensagem de requisição de assinatura.

Considerações Gerais

- Todas as mensagens enviadas e recebidas via WebSockets são codificadas no formato JSON
- Todos os campos de ponto flutuante (incluindo timestamps) são entre aspas para preservar a precisão.
- O formato de cada par negociável é A/B, onde A e B são ISO 4217-A3 para ativos padronizados e o símbolo único popular se não padronizado.
- Os timestamps não devem ser considerados únicos nem como aliases para transaction ids. Além disso, a granularidade dos timestamps não é representativa das taxas de transação.

Supported Pairs

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

Métodos

Ping

O cliente pode enviar ping ao servidor para determinar se a conexão está ativa, e o servidor responde com pong. Este é um ping em nível de aplicação, ao contrário do ping padrão no padrão WebSockets, que é iniciado pelo servidor

Ticker

As informações de ticker incluem os melhores preços de ask e bid, volume de 24h, último preço negociado, preço médio ponderado por volume, etc. para um determinado par de moedas. Uma mensagem de ticker é publicada toda vez que uma negociação ou um grupo de negociações ocorre.

Faça subscribe a um ticker chamando o método `SubscribeTicker`:

```
SubscribeTicker(["XBT/USD"]);
```

Se a assinatura for bem-sucedida, o evento **OnKrakenSubscribed** será chamado:

```
void OnKrakenSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription, string ChannelName,
```

```
int ReqID)
{
    DoLog("#subscribed: " + Subscription + " " + Pair + " " + ChannelName);
}
```

Cancelar a assinatura chamando o método UnSubscribeTicker:

```
UnSubscribeTicker(ARRAYOFCONST(("XBT/USD")));
```

Se o cancelamento da assinatura for bem-sucedido, o evento OnKrakenUnSubscribed será chamado:

```
void OnKrakenUnSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#unsubscribed: " + Subscription + " " + Pair);
}
```

Se houver um erro ao tentar assinar / cancelar a assinatura, o evento OnKrakenSubscriptionError será chamado.

```
void OnKrakenSubscriptionError(TObject *Sender, string ErrorMessage, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#subscription error: " + ErrorMessage);
}
```

As atualizações de ticker serão notificadas no 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 assinado para OHLC, um snapshot do último candle válido (independentemente do endtime) será enviado, seguido de atualizações do candle em andamento. Por exemplo, se uma assinatura for feita para um candle de 1 min e não houver trades por 5 min, um snapshot do último candle de 1 min de 5 min atrás será publicado. O endtime pode ser usado para determinar que se trata de um candle antigo.

Assine um OHLC chamando o método `SubscribeOHLC`, você deve passar o par e o intervalo.

```
SubscribeOHLC(ARRAYOFCONST((L"XBT/USD")), kin1min);
```

Se a assinatura for bem-sucedida, o evento `OnKrakenSubscribed` será chamado:

```
void OnKrakenSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription, string ChannelName,
int ReqID)
{
    DoLog("#subscribed: " + Subscription + " " + Pair + " " + ChannelName);
}
```

Cancele a assinatura chamando o método `UnSubscribeOHLC`:

```
UnSubscribeOHLC(ARRAYOFCONST(("XBT/USD")), kin1min);
```

Se o cancelamento da assinatura for bem-sucedido, o evento `OnKrakenUnSubscribed` será chamado:

```
void OnKrakenUnSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
int ReqID)
{
    DoLog("#unsubscribed: " + Subscription + " " + Pair);
}
```

Se houver um erro ao tentar assinar / cancelar a assinatura, o evento `OnKrakenSubscriptionError` será chamado.

```
void OnKrakenSubscriptionError(TObject *Sender, string ErrorMessage, string Pair, string Subscription,
int ReqID)
{
    DoLog("#subscription error: " + ErrorMessage);
}
```

As atualizações de OHLC serão notificadas no evento `OnKrakenData`.

```
[
  42,
  [
    "1542057314.748456",
    "1542057360.435743",
    "3586.70000",
    "3586.70000",
    "3586.60000",
    "3586.60000",
    "3586.68894",
    "0.03373000",
  ],
  2,
  "ohlc-5",
  "XBT/USD"
]
```

Trade

Feed de negociações para um par de moedas.

Assine o feed Trade chamando o método `SubscribeTrade`.

```
SubscribeTrade(ARRAYOFCONST((L"XBT/USD")));
```

Se a assinatura for bem-sucedida, o evento `OnKrakenSubscribed` será chamado:

```
void OnKrakenSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
    string ChannelName, int ReqID)
{
    DoLog("#subscribed: " + Subscription + " " + Pair + " " + ChannelName);
}
```

UnSubscribe chamando o método UnSubscribeTrade:

```
UnSubscribeTrade(ARRAYOFCNST((L"XBT/USD")));
```

Se o cancelamento da assinatura for bem-sucedido, o evento OnKrakenUnSubscribed será chamado:

```
void OnKrakenUnSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#unsubscribed: " + Subscription + " " + Pair);
}
```

Se houver um erro ao tentar assinar / cancelar a assinatura, o evento OnKrakenSubscriptionError será chamado.

```
void OnKrakenSubscriptionError(TObject *Sender, string ErrorMessage, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#subscription error: " + ErrorMessage);
}
```

As atualizações de trade serão notificadas no evento OnKrakenData.

```
[
    0,
    [
        [
            "5541.20000",
            "0.15850568",
            "1534614057.321597",
            "s",
            "1",
            ""
        ],
        [
            "6060.00000",
            "0.02455000",
            "1534614057.324998",
            "b",
            "1",
            ""
        ]
    ],
    "trade",
    "XBT/USD"
]
```

Book

Níveis do livro de ordens. Na assinatura, um snapshot será publicado na profundidade especificada; após o snapshot, as atualizações de nível serão publicadas.

Assine um Book chamando o método SubscribeBook; você deve passar o par e a profundidade (depth).

```
SubscribeBook(ARRAYOFCNST((L"XBT/USD")), kde10);
```

Se a assinatura for bem-sucedida, o evento OnKrakenSubscribed será chamado:

```
void OnKrakenSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription, string ChannelName,
    int ReqID)
{
    DoLog("#subscribed: " + Subscription + " " + Pair + " " + ChannelName);
}
```

Cancelar a assinatura chamando o método `UnSubscribeBook`:

```
UnSubscribeBook(ARRAYOFCONST((L"XBT/USD")), kde10);
```

Se o cancelamento da assinatura for bem-sucedido, o evento `OnKrakenUnSubscribed` será chamado:

```
void OnKrakenUnSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#unsubscribed: " + Subscription + " " + Pair);
}
```

Se houver um erro ao tentar assinar / cancelar a assinatura, o evento `OnKrakenSubscriptionError` será chamado.

```
void OnKrakenSubscriptionError(TObject *Sender, string ErrorMessage, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#subscription error: " + ErrorMessage);
}
```

As atualizações do book serão notificadas no 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 de spread para mostrar o melhor preço de compra e venda para o par de ativos assinado. O volume de compra e o volume de venda também fazem parte da mensagem.

Assina o feed de Spread chamando o método `SubscribeSpread`.

```
SubscribeSpread(ARRAYOFCONST(("XBT/USD")));
```

Se a assinatura for bem-sucedida, o evento `OnKrakenSubscribed` será chamado:

```
void OnKrakenSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription, string ChannelName,
int ReqID)
{
    DoLog("#subscribed: " + Subscription + " " + Pair + " " + ChannelName);
}
```

Cancele a assinatura chamando o método UnSubscribeSpread:

```
UnSubscribeSpread(ARRAYOFCNST((L"XBT/USD")));
```

Se o cancelamento da assinatura for bem-sucedido, o evento OnKrakenUnSubscribed será chamado:

```
void OnKrakenUnSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
int ReqID)
{
    DoLog("#unsubscribed: " + Subscription + " " + Pair);
}
```

Se houver um erro ao tentar assinar / cancelar a assinatura, o evento OnKrakenSubscriptionError será chamado.

```
void OnKrakenSubscriptionError(TObject *Sender, string ErrorMessage, string Pair, string Subscription,
int ReqID)
{
    DoLog("#subscription error: " + ErrorMessage);
}
```

As atualizações de spread serão notificadas no evento OnKrakenData.

```
[
    0,
    [
        "5698.40000",
        "5700.00000",
        "1542057299.545897",
        "1.01234567",
        "0.98765432"
    ],
    "spread",
    "XBT/USD"
]
```

Outros métodos

Você pode assinar / cancelar a assinatura de todos os canais com um único método:

```
SubscribeAll(ARRAYOFCNST((L"XBT/USD")));
UnSubscribeAll(ARRAYOFCNST((L"XBT/USD")));
```

O valor do intervalo OHLC é 1 se todos os canais estiverem assinados.

Eventos

OnConnect: quando o cliente websocket está conectado ao servidor.

OnKrakenConnect: chamado após conexão websocket bem-sucedida e quando o servidor envia o status do sistema.

OnKrakenSystemStatus: chamado quando o status do sistema muda.

OnKrakenSubscribed: chamado após uma assinatura bem-sucedida a um canal.

OnKrakenUnSubscribed: chamado após uma desinscrição bem-sucedida de um canal.

OnKrakenSubscriptionError: chamado se houver um erro ao tentar assinar / cancelar a assinatura.

OnKrakenData: chamado toda vez que uma assinatura de canal recebe uma atualização.

API Kraken | WebSockets Private API

Conexão

URL: wss://ws-auth.kraken.com (v1) ou wss://ws-auth.kraken.com/v2 (v2, recomendado)
O componente agora suporta a WebSocket API v2 por meio da propriedade **Version** (padrão: 2).

Uma vez que o socket esteja aberto, você pode se inscrever em canais de dados privados enviando uma mensagem de requisição de subscribe autenticada.

Autenticação

O cliente da API deve solicitar um "token" de autenticação por meio do seguinte endpoint da REST API "GetWebSocketsToken" para conectar-se aos endpoints Private de WebSockets. O token deve ser utilizado dentro de 15 minutos após sua criação. O token não expira uma vez que uma conexão a uma mensagem private da API de WebSockets (openOrders ou ownTrades) seja mantida.

Para obter um Token Websockets, uma API Key e um API Secret devem ser definidos no componente Kraken Options, a api key fornecida pela Kraken em sua conta

```
Kraken->ApiKey = "api key";
Kraken->ApiSecret = "api secret";
```

Métodos

OwnTrades

Obtenha uma lista dos seus próprios trades; na primeira assinatura, você obtém uma lista dos últimos 50 trades

```
SubscribeOwnTrades();
```

Mais tarde, você pode cancelar a assinatura de OwnTrades chamando o método UnSubscribeOwnTrades.

```
UnSubscribeOwnTrades();
```

Exemplo de resposta do servidor

```
[
  [
    {
      "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"
]
```


Ordens abertas

Feed para mostrar todas as ordens em aberto pertencentes à chave de API do usuário autenticado. O snapshot inicial fornecerá a lista de todas as ordens em aberto e, em seguida, quaisquer atualizações à lista de ordens em aberto serão enviadas. Para atualizações de mudança de status, como 'closed', os campos orderid e status estarão presentes no payload

```
SubscribeOpenOrders();
```

Posteriormente, você pode cancelar a assinatura de OpenOrders, chamando o método UnSubscribeOpenOrders .

```
UnSubscribeOpenOrders();
```

Exemplo de resposta do servidor

```
[
  [
    {
      "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"
]
```

Add Order

Envia uma nova ordem à Kraken

```
TsgcWSKrakenOrder *oKrakenOrder = new TsgcWSKrakenOrder();
oKrakenOrder->Pair = "XBT/USD";
oKrakenOrder->_Type = kosBuy;
oKrakenOrder->OrderType = kotMarket;
oKrakenOrder->Volume = 1;
AddOrder(oKrakenOrder);
```

Lista de parâmetros da Ordem

```
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

```

Exemplo de resposta do servidor

```

{
  "descr": "buy 0.01770000 XBTUSD @ limit 4000",
  "event": "addOrderStatus",
  "status": "ok",
  "txid": "ONPNXH-KMKMU-F4MR5V"
}

```

Cancelar Ordem

Cancelar ordem

```
CancelOrder("Order Id");
```

Exemplo de resposta do servidor

```

{
  "event": "cancelOrderStatus",
  "status": "ok"
}

```

API Kraken | REST Public API

Conexão

URL: <https://api.kraken.com>

A API Pública da Kraken não requer nenhuma autenticação.

Configuração

A única configuração é se deve habilitar um log para requisições HTTP REST. Habilite HTTPLogOptions se você deseja salvar em um arquivo de texto o log de todas as Requisições/Respostas HTTP

Eventos

OnKrakenHTTPException: este evento é chamado se houver qualquer exceção ao fazer uma requisição HTTP da API REST.

Métodos

GetServerTime

Este método serve para ajudar a aproximar o desvio de tempo entre o servidor e o cliente. Retorna o Time no formato Unix.

```
{"error":[],"result":{"unixtime":1586705546,"rfc1123":"Sun, 12 Apr 20 15:32:26 +0000"}}
```

GetAssets

Retorna informações sobre Assets

```
{"error":[],"result":{"ADA":{"aclass":"currency","altname":"ADA","decimals":8,"display_decimals":6}}}}
```

GetAssetPairs

Retorna informações sobre um par de ativos

```
Kraken->REST_API->GetAssetPairs(ARRAYOFCONST((L"XBTUSD")));
```

GetTicker

Retorna informações do ticker

```
Kraken->REST_API->GetTicker(ARRAYOFCONST((L"XBTUSD")));
```

GetOHLC

Retorna dados Open-High-Low-Close.

```
Kraken->REST_API->GetOHLC("XBTUSD");
```

GetOrderBook

Retorna um Array com o nome do par e a profundidade do mercado.

```
Kraken->REST_API->GetOrderBook("XBTUSD");
```

GetTrades

Retorna dados recentes de trade de um par.

```
Kraken->REST_API->GetTrades("XBTUSD");
```

GetSpread

Retorna dados recentes de spread de um par.

```
Kraken->REST_API->GetSpread("XBTUSD");
```

API Kraken | REST Private API

Conexão

URL: <https://api.kraken.com>

Autenticação

A REST Private API requer uma API Key e uma API Secret; esses valores são fornecidos pela Kraken na sua conta.

```
Kraken->ApiKey = "api key";  
Kraken->ApiSecret = "api secret";
```

Métodos

GetAccountBalance

Retorna o saldo da sua conta.

```
Kraken->REST_API->GetAccountBalance();
```

GetTradeBalance

Retorna informações sobre seus trades.

```
Kraken->REST_API->GetTradeBalance();
```

GetOpenOrders

Retorna uma lista de ordens abertas.

```
Kraken->REST_API->GetOpenOrders();
```

GetClosedOrders

Retorna uma lista de ordens fechadas.

```
Kraken->REST_API->GetClosedOrders();
```

QueryOrders

Consulta informações sobre uma ordem.

```
Kraken->REST_API->QueryOrders("1234");
```

GetTradesHistory

Retorna um array de informações de trade.

```
Kraken->REST_API->GetTradesHistory();
```

QueryTrades

Consulta informações sobre uma negociação.

```
Kraken->REST_API->QueryTrades("1234");
```

GetOpenPositions

Retorna informações da posição.

```
Kraken->REST_API->GetOpenPositions("1234");
```

GetLedgers

Retorna um array associativo de informações de ledgers.

```
Kraken->REST_API->GetLedgers();
```

QueryLedgers

Retorna um array associativo de informações de ledgers.

```
Kraken->REST_API->QueryLedgers("1234");
```

GetTradeVolume

Retorna informações sobre o volume de negociação.

```
Kraken->REST_API->GetTradeVolume();
```

AddExport

Adiciona uma nova exportação de relatório.

```
Kraken->REST_API->AddExport("Report All Trades");
```

ExportStatus

Get Status of reports

```
Kraken->REST_API->ExportStatus();
```

RetrieveExport

Obtém o Report pelo id do report.

```
Kraken->REST_API->RetrieveExport("GOCO");
```

RemoveExport

Remove o Report pelo id do report.

```
Kraken->REST_API->RemoveExport("GOCO");
```

Add Order

Adiciona uma nova ordem

```
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

Cancela uma ordem aberta por id

```
Kraken->REST_API->CancelOrder("1234");
```

API Kraken Futures

Kraken Futures

Visão geral

A **REST API** permite que você acesse com segurança os métodos da sua conta Kraken Futures. Exemplos de Métodos da REST API:

- requisita informações de preço atuais ou históricas
- verifique o saldo da sua conta e o PnL
- seus parâmetros de margem e limiares estimados de liquidação
- colocar ou cancelar ordens (individualmente ou em lote)
- ver suas ordens abertas
- posições abertas ou histórico de negociações
- solicita um saque de ativo digital

Esses métodos são chamados de "endpoints" e são explicados na seção REST API.

A **Websocket API** permite que você estabeleça com segurança um canal de comunicação com a plataforma Kraken Futures para receber informações em tempo real. Isto permite escutar atualizações em vez de enviar requisições continuamente. Esses canais são chamados de inscrições.

Alguns dos endpoints permitem realizar tarefas sensíveis, como iniciar um saque de ativos digitais. Para acessar esses endpoints com segurança, a API utiliza técnicas de criptografia desenvolvidas pela National Security Agency.

Configuração

Para usar a API, você precisa gerar um par de **API keys** únicas (se quiser acesso às APIs privadas):

1. Faça login na sua **conta Kraken Futures**.
2. Clique no seu nome no canto superior direito.
3. Selecione "Settings" no menu suspenso.
4. Selecione a aba "Create Key" no painel da API.
5. Pressione o botão "Create Key".
6. Visualize suas chaves Pública e Privada e registre-as em algum lugar seguro.

Copie as chaves pública e privada para a propriedade **KrakenOptions** do componente.

```
KrakenOptions.ApiKey  
KrakenOptions.ApiSecret
```

APIs suportadas

- **API Pública WebSockets**: conecta-se a um servidor WebSocket público.
- **WebSockets Private API**: conecta-se a um servidor WebSocket privado e requer uma API Key e uma API Secret para autenticar contra o servidor.
- **REST Public API**: conecta-se a um servidor REST público.
- **REST Private API**: conecta-se a um servidor REST público e requer uma API Key e uma API Secret para autenticar no servidor.

REST API Methods

Private Endpoints

Method	Arguments	Descrição
BatchOrder		Enviar ordens em lote (place/cancel/edit).
GetOrderStatus		Obtém o status de ordens específicas.
GetPNL-Currency-Preferences		Obtém as preferências de moeda do PNL.
SetPNL-Currency-Preference		Define a preferência de moeda do PNL para um símbolo.
GetLeverageSettings		Obtém as preferências de alavancagem.
SetLeverageSettings		Define a alavancagem para um símbolo.

API Kraken Futures | WebSockets Public API

Conexão

URL: `wss://futures.kraken.com/ws/v1`

Assim que o socket estiver aberto, você pode assinar um canal público enviando uma mensagem de requisição de assinatura.

Métodos

Ticker

Este endpoint retorna dados de mercado atuais para todos os contratos e índices de Futures atualmente listados. Autenticação não é necessária.

Faça subscribe a um ticker chamando o método **SubscribeTicker**:

```
SubscribeTicker(ARRAYOFCONST(("PI_XBTUSD")));
```

Se a assinatura for bem-sucedida, o evento **OnKrakenFuturesSubscribed** será chamado:

```
void OnKrakenFuturesSubscribed(TObject *Sender, string Feed, string ProductId)
{
    DoLog("#subscribed: " + Feed + " " + ProductId);
}
```

Cancele a assinatura chamando o método **UnSubscribeTicker**:

```
UnSubscribeTicker(ARRAYOFCONST(("PI_XBTUSD")));
```

Se o cancelamento da assinatura for bem-sucedido, o evento **OnKrakenFuturesUnSubscribed** será chamado:

```
void OnKrakenFuturesUnSubscribed(TObject *Sender, string Feed, string ProductId)
{
    DoLog("#unsubscribed: " + Feed + " " + ProductId);
}
```

Se houver um erro ao tentar fazer subscribe / unsubscribe, o evento **OnKrakenFuturesError** será chamado.

```
void OnKrakenFuturesError(TObject *Sender, string Error)
{
    DoLog("#error: " + Error);
}
```

As atualizações de ticker serão notificadas no 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

O feed de trade retorna informações sobre negociações executadas. Assine o feed Trade chamando o método **SubscribeTrade**.

```
SubscribeTrade(ARRAYOFCONST(("PI_XBTUSD")));
```

Se a assinatura for bem-sucedida, o evento **OnKrakenFuturesSubscribed** será chamado:

```

void OnKrakenFuturesSubscribed(TObject *Sender, string Feed, string ProductId,)
{
  DoLog("#subscribed: " + Feed + " " + ProductId);
}

```

Cancele a assinatura chamando o método **UnSubscribeTrade**:

```
UnSubscribeTrade(ARRAYOFCONST(("PI_XBTUSD")));
```

Se o cancelamento da assinatura for bem-sucedido, o evento **OnKrakenFuturesUnSubscribed** será chamado:

```

void OnKrakenFuturesUnSubscribed(TObject *Sender, string Feed, string ProductId)
{
  DoLog("#unsubscribed: " + Feed + " " + ProductId);
}

```

Se houver um erro ao tentar fazer subscribe / unsubscribe, o evento **OnKrakenFuturesError** será chamado.

```

void OnKrakenFuturesError(TObject *Sender, string Error)
{
  DoLog("#error: " + Error);
}

```

As atualizações de trade serão notificadas no 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
    }
    ]
}

```

Book

Este feed retorna informações sobre o livro de ordens.

Inscreva-se em um Book chamando o método `SubscribeBook`; você deve passar o `Symbol`.

```
SubscribeBook(ARRAYOFCONST(("PI_XBTUSD")));
```

Se a assinatura for bem-sucedida, o evento **OnKrakenFuturesSubscribed** será chamado:

```
void OnKrakenFuturesSubscribed(TObject *Sender, string Feed, string ProductId)
{

```

```
DoLog("#subscribed: " + Feed + " " + ProductId);
}
```

Cancele a assinatura chamando o método **UnSubscribeBook**:

```
UnSubscribeBook(ARRAYOFCONST(("PI_XBTUSD")));
```

Se o cancelamento da assinatura for bem-sucedido, o evento **OnKrakenFuturesUnSubscribed** será chamado:

```
void OnKrakenFuturesUnSubscribed(TObject *Sender, string Feed, string ProductId)
{
    DoLog("#unsubscribed: " + Feed + " " + ProductId);
}
```

Se houver um erro ao tentar fazer subscribe / unsubscribe, o evento **OnKrakenFuturesError** será chamado.

```
void OnKrakenFuturesError(TObject *Sender, string Error)
{
    DoLog("#error: " + Error);
}
```

As atualizações de book serão notificadas no 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

O feed ticker lite retorna informações de ticker sobre os produtos listados. Assine o feed Spread chamando o método **SubscribeTickerLite**.

```
SubscribeTickerLite(ARRAYOFCONST(("PI_XBTUSD")));
```

Se a assinatura for bem-sucedida, o evento **OnKrakenFuturesSubscribed** será chamado:

```
void OnKrakenFuturesSubscribed(TObject *Sender, string Feed, string ProductId)
{
    DoLog("#subscribed: " + Feed + " " + ProductId);
}
```

Cancele a assinatura chamando o método **UnSubscribeTickerLite**:

```
UnSubscribeTickerLite(ARRAYOFCONST(("PI_XBTUSD")));
```

Se o cancelamento da assinatura for bem-sucedido, o evento **OnKrakenFuturesUnSubscribed** será chamado:

```
void OnKrakenFuturesUnSubscribed(TObject *Sender, string Feed, string ProductId)
{
    DoLog("#unsubscribed: " + Feed + " " + ProductId);
}
```

Se houver um erro ao tentar fazer subscribe / unsubscribe, o evento **OnKrakenFuturesError** será chamado.

```
void OnKrakenFuturesError(TObject *Sender, string Error)
{
    DoLog("#error: " + Error);
}
```

As atualizações de spread serão notificadas no 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

O feed de heartbeat publica uma mensagem de heartbeat em intervalos cronometrados.

```

SubscribeHeartBeat();
UnSubscribeHeartBeat();

```

Eventos

OnConnect: quando o cliente websocket está conectado ao servidor.

OnKrakenFuturesConnect: chamado após uma conexão websocket bem-sucedida e quando o servidor envia o status do sistema.

OnKrakenFuturesSubscribed: chamado após uma assinatura bem-sucedida em um canal.

OnKrakenFuturesUnSubscribed: chamado após uma desinscrição bem-sucedida de um canal.

OnKrakenFuturesError: chamado se houver algum erro durante a inscrição/cancelamento de inscrição.

OnKrakenData: chamado toda vez que uma assinatura de canal recebe uma atualização.

API Kraken Futures | WebSockets Private API

Conexão

URL: wss://futures.kraken.com/ws/v1

Autenticação

As requisições de subscribe e unsubscribe aos feeds privados do WebSocket exigem uma mensagem de challenge assinada com o api_secret do usuário.

O desafio (challenge) é obtido conforme mostrado na seção WebSocket API Public (utilizando a api_key).

Requisições autenticadas devem incluir tanto a mensagem de desafio original (original_challenge) quanto a assinada (signed_challenge) em formato JSON.

Para obter um Challenge de WebSockets, uma API Key e um API Secret devem ser definidos no componente Kraken Options, a api key fornecida pela Kraken na sua conta

```
Kraken->ApiKey = "api key";
Kraken->ApiSecret = "api secret";
```

Métodos

Open Orders Verbose

Este feed de assinatura publica informações sobre as ordens abertas do usuário. Este feed adiciona informações extras sobre todas as post-only orders que falharam ao cruzar o book.

```
SubscribeOpenOrdersVerbose();
```

Posteriormente, você pode cancelar a assinatura de OpenOrdersVerbose, chamando o método **UnSubscribeOpenOrdersVerbose**

```
UnSubscribeOpenOrdersVerbose();
```

Exemplo de resposta do servidor

```
{
  'feed': 'open_orders_verbose_snapshot',
  'account': '0f9c23b8-63e2-40e4-9592-6d5aa57c12',
  {
    'instrument': 'PI_XBTUSD',
    'time': 1567428848005,
    'last_update': ...
  }
}
```

Posições Abertas

Este feed de inscrição publica as posições abertas da conta do usuário.

```
SubscribeOpenPositions();
```

Posteriormente, você pode cancelar a assinatura de OpenPositions chamando o método **UnSubscribeOpenPositions**

```
UnSubscribeOpenPositions();
```

Exemplo de resposta do servidor

```
{
  "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 da Conta

Este feed de assinatura publica informações da conta.

```
SubscribeAccountLog();
```

Mais tarde, você pode cancelar a assinatura de AccountLog chamando o método **UnSubscribeAccountLog**

```
UnSubscribeAccountLog();
```

Exemplo de resposta do servidor

```
{
  '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

Este feed de assinatura publica informações de fills.

```
SubscribeFills();
```

Mais tarde, você pode cancelar a assinatura de Fills chamando o método **UnSubscribeFills**

```
UnSubscribeFills();
```

Exemplo de resposta do servidor

```
{
  "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"
    }
  ]
}
```

Ordens abertas

Este feed de inscrição publica informações sobre as ordens abertas do usuário.

```
SubscribeOpenOrders();
```

Posteriormente, você pode cancelar a inscrição em OpenOrders, chamando o método **UnSubscribeOpenOrders**

```
UnSubscribeOpenOrders();
```

Exemplo de resposta do servidor

```
{
  "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"
    }
  ]
}
```

Account Balance And Margins

Este feed de subscrição retorna informações de saldo e margem para a conta do cliente.

```
SubscribeAccountBalanceAndMargins();
```

Posteriormente, você pode cancelar a assinatura de AccountBalance, chamando o método **UnSubscribeAccountBalanceAndMargins**

```
UnSubscribeAccountBalanceAndMargins();
```

Exemplo de resposta do servidor

```
{
  "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
}

```

Notifications

Este feed de assinatura publica notificações ao cliente.

```
SubscribeNotifications();
```

Posteriormente, você pode cancelar a assinatura das Notificações, chamando o método **UnSubscribeNotifications**

```
UnSubscribeNotifications();
```

Exemplo de resposta do servidor

```
{  
  
  "feed": "notifications_auth",  
  
  "notifications": [  
  
    {  
  
      "id": 5,  
  
      "type": "market",  
  
      "priority": "low",  
  
      "note": "A note describing the notification.",  
  
      "effective_time": 1520288300000  
    },  
  
    ...  
  ]  
}
```

API Kraken Futures | REST Public API

Conexão

URL: <https://futures.kraken.com/derivatives/api/v3>

A API Pública do Kraken Futures não requer nenhuma autenticação.

Configuração

A única configuração é se deve habilitar um log para requisições HTTP REST. Habilite HTTPLogOptions se você deseja salvar em um arquivo de texto o log de todas as Requisições/Respostas HTTP

Eventos

OnKrakenHTTPException: este evento é chamado se houver qualquer exceção ao fazer uma requisição HTTP da API REST.

Métodos

GetFeeSchedules

Este endpoint lista todas as tabelas de taxas. A autenticação não é necessária.

```
KrakenFutures->REST_API->GetFeeSchedules();
```

Order Book

Este endpoint retorna o order book não cumulativo completo dos contratos Futures atualmente listados.

```
KrakenFutures->REST_API->GetOrderBook("PI_XBTUSD");
```

Tickers

Este endpoint retorna dados de mercado atuais para todos os contratos e índices de Futures atualmente listados.

```
KrakenFutures->REST_API->GetTickers();
```

Instruments

Este endpoint retorna especificações para todos os contratos e índices de Futuros atualmente listados.

```
KrakenFutures->REST_API->GetInstruments();
```

History

Este endpoint retorna as últimas 100 negociações a partir do valor lastTime especificado; se nenhum valor for especificado, retornará as últimas 100 negociações. Este endpoint retorna apenas histórico de negociações de no máximo 7 dias a partir do momento em que é chamado ou desde a última versão do trading engine (o que ocorrer primeiro).

```
KrakenFutures->REST_API->GetHistory("PI_XBTUSD");
```


API Kraken Futures | REST Private API

Conexão

URL: <https://futures.kraken.com/derivatives/api/v3>

Autenticação

A REST Private API requer uma API Key e uma API Secret; esses valores são fornecidos pela Kraken na sua conta.

```
Kraken->ApiKey = "api key";
Kraken->ApiSecret = "api secret";
```

Métodos

EditOrderByOrderId

Este endpoint permite editar uma ordem existente de um contrato de Futures atualmente listado.

aOrderId: ID da ordem que você deseja editar

aSize: O tamanho associado à ordem

aLimitPrice: O preço limite associado à ordem.

aStopPrice: O preço de stop associado a uma ordem stop. Obrigatório se o tipo de ordem antigo for Stop.

```
KrakenFutures->REST_API->EditOrderByOrderId("Order_Id", 2, 1000);
```

EditOrderByCliOrderId

Este endpoint permite editar uma ordem existente de um contrato de Futures atualmente listado.

aCliOrderId: A identidade da ordem especificada pelo usuário. Deve ser globalmente única.

aSize: O tamanho associado à ordem

aLimitPrice: O preço limite associado à ordem.

aStopPrice: O preço de stop associado a uma ordem stop. Obrigatório se o Order Type for Stop.

```
KrakenFutures->REST_API->EditOrderByCliOrderId("Cli_Order_Id", 2, 1000);
```

SendMarketOrder

Este endpoint permite que você envie uma Market Order.

aSide: A direção da ordem: buy ou sell.

aSymbol: O símbolo dos futuros

aSize: O tamanho associado à ordem.

```
KrakenFutures->REST_API->SendMarketOrder(kosfBuy, "PI_XBTUSD", 1);
```

SendLimitOrder

Este endpoint permite que você envie uma Limit Order.

aSide: A direção da ordem: buy ou sell.

aSymbol: O símbolo dos futuros

aSize: O tamanho associado à ordem.

aLimitPrice: O preço limite associado à ordem.

```
KrakenFutures->REST_API->SendLimitOrder(kosfBuy, "PI_XBTUSD", 1, 1000);
```

SendStopOrder

Este endpoint permite que você envie uma Stop Order.

aSide: A direção da ordem: buy ou sell.

aSymbol: O símbolo dos futuros

aSize: O tamanho associado à ordem.

aStopPrice: O preço de stop associado a uma ordem stop.

aLimitPrice: O preço limite associado à ordem.

```
KrakenFutures->REST_API->SendStopOrder(kosfBuy, "PI_XBTUSD", 1, 1000, 900);
```

SendTakeProfitOrder

Este endpoint permite que você envie uma Take Profit Order.

aSide: A direção da ordem: buy ou sell.

aSymbol: O símbolo dos futuros

aSize: O tamanho associado à ordem.

aStopPrice: O preço de stop associado a uma ordem stop.

aLimitPrice: O preço limite associado à ordem.

```
KrakenFutures->REST_API->SendTakeProfitOrder(kosfBuy, "PI_XBTUSD", 1, 1000, 900);
```

SendOrder

Este endpoint permite enviar uma ordem limit, stop, take profit ou immediate-or-cancel para um contrato Futures atualmente listado.

OrderType: selecione um dos seguintes: kotfLMT, kotfPOST, kotfMKT, kotfSTP, kotfTAKE_PROFIT, kotfIOC

Symbol: O símbolo dos futuros

Side: A direção da ordem (buy ou sell).

Size: O tamanho associado à ordem.

StopPrice: O preço de stop associado a uma ordem stop.

LimitPrice: O limit price associado à ordem.

TriggerSignal: Ao colocar uma ordem Stop ou TakeProfit, o sinal usado para disparo, selecione um dos seguintes kotsMark, kotsIndex, kotsLast

CliOrderId: A identidade da ordem que é especificada pelo usuário. Deve ser globalmente única.

ReduceOnly: Defina como true se você quiser que a ordem apenas reduza uma posição existente. Qualquer ordem que aumente uma posição existente será rejeitada. Padrão false.

```
TsgcHTTPKrakenFuturesOrder *oOrder = new TsgcHTTPKrakenFuturesOrder(this);
try
{
    oOrder->Side = kosfBuy;
    oOrder->Symbol = "PI_XBTUSD";
    oOrder->OrderType = kotfMKT;
    oOrder->Size = 1;
    KrakenFutures->REST_API->SendOrder(oOrder);
}
```

```

}
__finally
{
    oOrder->Free;
}

```

CancelOrderByOrderId

Este endpoint permite cancelar uma ordem aberta para um contrato de Futures.

aOrderId: ID da ordem que você deseja editar

```
KrakenFutures->REST_API->CancelOrderByOrderId("Order_Id");
```

CancelOrderByCliOrderId

Este endpoint permite cancelar uma ordem aberta para um contrato de Futures.

aCliOrderId: A identidade da ordem especificada pelo usuário. Deve ser globalmente única.

```
KrakenFutures->REST_API->CancelOrderByCliOrderId("Cli_Order_Id");
```

GetFills

Este endpoint retorna informações sobre ordens executadas (filled) para todos os contratos de futures.

aLastFillDate: Se não fornecido, retorna os últimos 100 fills em qualquer contrato de futuros. Se fornecido, retorna as 100 entradas anteriores a lastFillTime.

```
KrakenFutures->REST_API->GetFills("2020-07-22T13:45:00.000Z");
```

Transfer

Este endpoint permite que você transfira fundos entre duas contas de margem com a mesma moeda de colateral, ou entre uma conta de margem e sua conta cash.

aFromAccount: O nome da conta cash ou margin de onde mover os fundos.

aToAccount: O nome da conta cash ou margin para a qual mover os fundos.

aUnit: A unidade a transferir.

aAmount: O valor a transferir.

```
KrakenFutures->REST_API->Transfer("FI_XBTUSD", "cash", "xbt", 1.5);
```

GetOpenPositions

Este endpoint retorna o tamanho e o preço médio de entrada de todas as posições abertas em contratos de Futures. Isso inclui contratos de Futures que venceram, mas ainda não foram liquidados.

```
KrakenFutures->REST_API->GetOpenPositions();
```

GetNotifications

Este endpoint fornece as notificações da plataforma.

```
KrakenFutures->REST_API->GetNotifications();
```

GetAccounts

Este endpoint retorna informações-chave relacionadas a todas as suas contas Kraken Futures, que podem ser contas cash ou contas margin. Isto inclui saldos de ativos digitais, saldos de instrumentos, requisitos de margem, estimativas de gatilho de margem e informações auxiliares como fundos disponíveis, PnL de posições abertas e valor do portfólio.

```
KrakenFutures->REST_API->GetAccounts();
```

CancelAllOrders

Este endpoint permite cancelar uma ordem aberta para um contrato de Futures.

Symbol: Um produto de futuros para cancelar todas as ordens abertas (opcional)

```
KrakenFutures->REST_API->CancelAllOrders();
```

CancelAllOrdersAfter

Este endpoint fornece um mecanismo Dead Man's Switch para proteger o cliente contra falhas de rede. O cliente pode enviar uma requisição com um timeout em segundos que dispara um cronômetro de contagem regressiva que cancelará todas as ordens do cliente quando o timeout expirar.

aTimeout: O timeout especificado em segundos.

```
KrakenFutures->REST_API->CancelAllOrdersAfter(60);
```

GetOpenOrders

Este endpoint retorna informações sobre todas as ordens abertas para todos os contratos de Futures.

```
KrakenFutures->REST_API->OpenOrders();
```

GetHistoricalOrders

Este endpoint retorna ordens históricas feitas em uma conta.

aSince: O DateTime Since

aBefore: O DateTime Before

aSort: "asc" para ordenação ascendente "desc" para descendente

aContinuationToken: Continuation token fornecido por uma resposta anterior, que pode ser usado em uma chamada para retornar o próximo conjunto de resultados disponíveis

```
KrakenFutures->REST_API->GetHistoricalOrders(Now, Now - 5);
```

GetHistoricalTriggers

Este endpoint retorna os gatilhos históricos feitos em uma conta.

aSince: O DateTime Since

aBefore: O DateTime Before

aSort: "asc" para ordenação ascendente "desc" para descendente

aContinuationToken: Continuation token fornecido por uma resposta anterior, que pode ser usado em uma chamada para retornar o próximo conjunto de resultados disponíveis

```
KrakenFutures->REST_API->GetHistoricalTriggers(Now, Now - 5);
```

GetHistoricalExecutions

Este endpoint retorna as execuções históricas feitas em uma conta.

aSince: O DateTime Since

aBefore: O DateTime Before

aSort: "asc" para ordenação ascendente "desc" para descendente

aContinuationToken: Continuation token fornecido por uma resposta anterior, que pode ser usado em uma chamada para retornar o próximo conjunto de resultados disponíveis

```
KrakenFutures->REST_API->GetHistoricalExecutions(Now, Now - 5);
```

WithdrawalToSpotWallet

Este endpoint permite submeter uma requisição para sacar ativos digitais de uma Kraken Futures wallet para sua Kraken Spot wallet.

aCurrency: O ativo digital que será retirado, por exemplo xbt ou xrp.

aAmount: A quantidade de moeda que deve ser sacada.

```
KrakenFutures->REST_API->WithdrawalToSpotWallet("xbt", 1000);
```

GetFeeScheduleVolumes

Este endpoint retorna seu volume em USD dos últimos 30 dias.

```
KrakenFutures->REST_API->GetFeeScheduleVolumes();
```

GetAccountLogCSV

Este endpoint permite que os clientes baixem um arquivo csv de seus logs de conta.

```
KrakenFutures->REST_API->GetAccountLogCSV();
```

API Pusher

Pusher

O Pusher é uma plataforma fácil e confiável com bons recursos baseados no protocolo WebSocket: mensageria pub/sub flexível, listas de usuários ao vivo (presence), autenticação...

A versão da API WebSocket do Pusher é 7.

Os dados são enviados bidirecionalmente sobre um WebSocket como dados de texto contendo JSON codificado em UTF8 (frames WebSocket binários não são suportados).

Você pode chamar o método **Ping** para testar a conexão com o servidor. Essencialmente, quaisquer mensagens recebidas da outra parte são consideradas como indicação de que a conexão está ativa. Na ausência de qualquer mensagem, qualquer uma das partes pode verificar se o outro lado está respondendo enviando uma mensagem de ping, à qual a outra parte deve responder com um pong.

Antes de conectar, você deve preencher os seguintes campos:

```
Pusher->Cluster = "eu"; // cluster where your pusher account is located
Pusher->Key = "9c3b7ef25qe97a00116c"; // your pusher api key
Pusher->Name = "js"; // optional, name of your application
Pusher->Version = "4.1"; // optional, version of your application
Pusher->TLS = true; // if encrypted, set to true
Pusher->Secret = "2dc792e1916ac49e6b3f"; // pusher secret string (needed for private and presence channels)
```

Importante

O Pusher exige que o cliente websocket conecte a uma URL usando os campos anteriores (key, cluster...), esses campos são usados para construir a url e isso é feito quando você atribui o cliente no componente pusher. Portanto, para ter certeza de que a URL é construída corretamente, defina o cliente depois de ter preenchido os campos de configuração do pusher. Veja abaixo o pseudocódigo:

```
// configure pusher fields
pusher.cluster = ...
pusher.key = ...
// set client
pusher.client = websocket client
// start connection
websocket client.Active = true;
```

Após uma conexão bem-sucedida, o evento **OnPusherConnect** é gerado e você obtém os seguintes campos:

- Socket ID: Um identificador único para o cliente conectado.
- Timeout: O número de segundos de inatividade do servidor após os quais o cliente deve iniciar uma mensagem de ping (isso é tratado automaticamente pelo componente).

Em caso de erro, **OnPusherError** será gerado e informações sobre o erro serão fornecidas. Um erro pode ser enviado pelo Pusher em resposta a autenticação inválida, um comando inválido, etc.

4000-4099

Indica um erro que resulta no fechamento da conexão pelo Pusher, e que tentar reconectar com os mesmos parâmetros não terá sucesso.

- 4000: O aplicativo aceita apenas conexões SSL, reconecte usando wss://
- 4001: A aplicação não existe
- 4003: Aplicação desabilitada
- 4004: A aplicação excedeu a cota de conexão
- 4005: Caminho não encontrado
- 4006: Formato de string de versão inválido
- 4007: Unsupported protocol version
- 4008: Nenhuma versão de protocolo fornecida

4100-4199

Indica um erro que resulta no fechamento da conexão pelo Pusher, e que o cliente pode reconectar após 1s ou mais.

- 4100: Capacidade excedida

4200-4299

Indica um erro que resulta no fechamento da conexão pelo Pusher, e que o cliente pode reconectar imediatamente.

- 4200: Reconexão genérica imediata
- 4201: Resposta Pong não recebida: o ping foi enviado ao cliente, mas nenhuma resposta foi recebida - veja as mensagens ping e pong

4202: Fechado após inatividade: O cliente esteve inativo por um longo período (atualmente 24 horas) e o cliente não suporta ping. Atualize para um rascunho WebSocket mais recente ou implemente a versão 5 ou superior deste protocolo.

4300-4399

Qualquer outro tipo de erro.

4301: Evento de cliente rejeitado devido ao rate limit

Canais

Os canais são um conceito fundamental no Pusher. Cada aplicação tem uma série de canais, e cada cliente pode escolher em quais canais se inscreve.

Os canais fornecem:

- Uma forma de filtrar dados. Por exemplo, em uma aplicação de chat, pode haver um canal para pessoas que querem discutir 'cachorros'
- Uma forma de controlar o acesso a diferentes streams de informação. Por exemplo, uma aplicação de gerenciamento de projetos quereria autorizar pessoas a receber atualizações sobre o 'projectX'

É altamente recomendável que os canais sejam usados para filtrar seus dados e que isso não seja feito usando eventos. Isso ocorre porque todos os eventos publicados em um canal são enviados a todos os assinantes, independentemente de seu vínculo de evento.

Os canais não precisam ser criados explicitamente e são instanciados sob demanda do cliente. Isso significa que criar um canal é fácil. Basta instruir um cliente a assiná-lo.

Os seguintes tipos de canais são suportados:

- **Canais públicos** podem ser assinados por qualquer pessoa que conheça seus nomes
- **Canais privados** introduzem um mecanismo que permite que seu servidor controle o acesso aos dados que você está transmitindo
- **Os canais de presença** são uma extensão dos canais privados. Eles permitem que você registre informações do usuário na assinatura e deixe que outros membros do canal saibam quem está online
- **Cache channels** lembram o último evento disparado e o enviam como o primeiro evento aos novos assinantes (variantes public, private e presence)
- Os **canais Private-Encrypted** fornecem criptografia de ponta a ponta utilizando NaCl secretbox, garantindo que nem mesmo o Pusher possa ler os dados da mensagem

Canais Públicos

Canais públicos devem ser utilizados para dados acessíveis publicamente, pois não exigem nenhuma forma de autorização para serem assinados.

Você pode assinar e cancelar a assinatura de canais a qualquer momento. Não há necessidade de esperar o Pusher terminar de conectar primeiro.

Exemplo: assinar o canal "my-channel".

```
CBuilder
APIPusher->Subscribe("my-channel");
```

Se você assinar com sucesso, o evento **OnPusherSubscribe** será gerado; se houver um erro, você receberá uma mensagem no evento **OnPusherError**.

Todas as mensagens do canal assinado serão recebidas no evento **OnPusherEvent**.

Quando o método Publish é chamado e o canal é Público, o componente, em vez de utilizar o protocolo WebSocket, utiliza o protocolo HTTP e chama o método TriggerEvent (a publicação não é permitida utilizando o protocolo websocket).

Private Channels

Requer Indy 10.5.7 ou posterior

Canais privados devem ser utilizados quando o acesso ao canal precisa ser restrito de alguma forma. Para que um usuário possa assinar um canal privado, a permissão deve ser autorizada.

Exemplo: assina o canal "my-private-channel".

```
CBuilder
APIPusher->Subscribe("my-private-channel", pscPrivateChannel);
```

Se você assinar com sucesso, o evento **OnPusherSubscribe** será gerado; se houver um erro, você receberá uma mensagem no evento **OnPusherError**.

Todas as mensagens do canal assinado serão recebidas no evento **OnPusherEvent**.

Canais de presença

Requer Indy 10.5.7 ou posterior

Os canais Presence se baseiam na segurança dos canais Private e expõem o recurso adicional de ter ciência de quem está inscrito naquele canal. Isso torna extremamente fácil construir funcionalidades de sala de chat e do tipo "quem está online" na sua aplicação. Pense em salas de chat, colaboradores em um documento, pessoas vendo a mesma página web, competidores em um jogo, esse tipo de coisa.

Os canais de presença são assinados a partir da API do cliente da mesma forma que os canais privados, mas o nome do canal deve ser prefixado com `presence-`. Assim como nos canais privados, uma requisição HTTP é feita a uma URL de autenticação configurável para determinar se o usuário atual tem permissões para acessar o canal. As informações sobre os usuários que se inscrevem e cancelam a inscrição em um canal podem então ser acessadas vinculando-se a eventos no presence channel, e o estado atual dos usuários inscritos no canal está disponível por meio da propriedade `channel.members`.

Exemplo: assina o canal "my-presence-channel".

```
APIPusher->Subscribe("my-presence-channel", pscPresenceChannel,
    '{"user_id":"'John_Smith'", "user_info":{"name":"'John Smith'"}'}');
```

Se você assinar com sucesso, o evento **OnPusherSubscribe** será gerado; se houver um erro, você receberá uma mensagem no evento **OnPusherError**.

Todas as mensagens do canal assinado serão recebidas no evento **OnPusherEvent**.

Cache Channels

Um canal de cache lembra o último evento disparado e o envia como o primeiro evento a novos assinantes.

Quando um evento é acionado em um canal de cache, o Pusher Channels armazena esse evento em cache e, quando um cliente assina um canal de cache, se existir um valor em cache, este é enviado ao cliente como o primeiro evento naquele canal. Esse comportamento ajuda os desenvolvedores a fornecer o estado inicial sem adicionar lógica adicional para buscá-lo em outro lugar.

Os seguintes Cache Channels são suportados:

- Public Cache Channel
- Private Cache Channel
- Presence Cache Channel

Exemplo: assinar o canal de cache público "my-cache-channel".

```
APIPusher->Subscribe("my-cache-channel", pscCacheChannel);
```

Se você assinar com sucesso, o evento **OnPusherSubscribe** será gerado; se houver um erro, você receberá uma mensagem no evento **OnPusherError**.

Todas as mensagens do canal assinado serão recebidas no evento **OnPusherEvent**.

Se não houver evento em cache ao assinar um canal de cache, o evento **OnPusherCacheMiss** será gerado, fornecendo o nome do canal. Isso permite que sua aplicação trate o caso em que nenhum dado em cache está disponível.

Private-Encrypted Channels

Os canais Private-Encrypted fornecem criptografia de ponta a ponta para as mensagens. Como os canais privados, eles exigem autenticação, mas, adicionalmente, todas as cargas úteis de dados são criptografadas usando NaCl secretbox, de modo que apenas assinantes autorizados possam ler o conteúdo. Nem mesmo a própria Pusher consegue descriptografar as mensagens.

Para utilizar canais privados-criptografados, você deve fornecer um **SharedSecret** durante a autenticação. O segredo compartilhado é utilizado para criptografar e descriptografar os dados da mensagem.

Exemplo: inscreva-se em um canal privado criptografado "my-encrypted-channel".

```
APIPusher->Subscribe("my-encrypted-channel", pscPrivateEncryptedChannel);
```


Também está disponível uma variante `private-encrypted-cache`, combinando criptografia com o comportamento de canal de cache:

```
APIPusher->Subscribe("my-encrypted-cache-channel", pscPrivateEncryptedCacheChannel);
```

Ao usar o evento **OnPusherAuthentication** com canais `private-encrypted`, você pode definir a propriedade **SharedSecret** no objeto de resposta para fornecer a chave de criptografia:

```
void OnPusherAuthenticationEvent(TObject *Sender,
    TsgcWSPusherRequestAuthentication *AuthRequest,
    TsgcWSPusherResponseAuthentication *AuthResponse)
{
    AuthResponse->SharedSecret = "your-shared-secret-key";
}
```

Presence Events

Os canais Presence fornecem eventos adicionais que notificam sua aplicação quando os usuários entram ou saem de um canal, e permitem que você rastreie as contagens de assinaturas.

OnPusherMemberAdded

Gerado quando um novo membro se inscreve em um presence channel. Fornece o nome do canal, o ID do usuário e as informações do usuário do membro que ingressou.

```
void PUSHERPusherMemberAdded(TObject *Sender,
    String Channel, String UserId, String UserInfo)
{
    Log("Member joined: " + UserId + " on " + Channel);
}
```

OnPusherMemberRemoved

Gerado quando um membro cancela a assinatura de um presence channel. Fornece o nome do canal, o user ID e as informações do usuário do membro que saiu.

```
void PUSHERPusherMemberRemoved(TObject *Sender,
    String Channel, String UserId, String UserInfo)
{
    Log("Member left: " + UserId + " on " + Channel);
}
```

OnPusherSubscriptionCount

Gerado quando a contagem de inscrições muda em um canal. Fornece o nome do canal e o número atual de assinantes. Este evento deve estar habilitado no seu painel do Pusher.

```
void PUSHERPusherSubscriptionCount(TObject *Sender,
    String Channel, int SubscriptionCount)
{
    Log(Channel + " has " + IntToStr(SubscriptionCount) + " subscribers");
}
```

OnPusherCacheMiss

Gerado ao assinar um canal de cache que não tem nenhum evento em cache. Fornece o nome do canal. Isso permite que a sua aplicação trate o caso em que nenhum dado em cache está disponível, por exemplo, buscando os dados de outra fonte.

```
void PUSHERPusherCacheMiss(TObject *Sender, String Channel)
{
    Log("Cache miss on: " + Channel);
}
```

Publicar Mensagens

Você não apenas pode receber mensagens dos canais assinados, mas também pode enviar mensagens a outros usuários assinados.

Chame o método **Publish** para enviar uma mensagem a todos os usuários assinados do canal.

Exemplo: enviar um evento a todos os usuários inscritos de "my-channel"

```
APIPusher->Publish("my-event", "my-channel");
```

Publique no máximo 10 mensagens por segundo por cliente (conexão). Quaisquer eventos disparados acima desse limite de taxa serão rejeitados pela Pusher API. Isso não é um problema de sistema, é um problema do cliente. 100 clientes em um canal enviando mensagens nessa taxa teriam, cada um, que processar também 1.000 mensagens por segundo! Embora alguns navegadores modernos possam ser capazes de lidar com isso, muito provavelmente não é uma boa ideia.

REST API

A API está hospedada em <http://api-CLUSTER.pusher.com>, onde CLUSTER é substituído pelo cluster de seus próprios apps (por exemplo, eu).

Códigos de status HTTP são usados para indicar o sucesso ou não das requisições. Os seguintes status são comuns:

200 Requisição bem-sucedida. O corpo conterá um hash JSON de dados de resposta

400 Erro: detalhes no corpo da resposta

401 Erro de autenticação: o corpo da resposta conterá uma explicação

403 Forbidden: app desabilitado ou acima da cota de mensagens

As seguintes funções da REST API foram implementadas.

Function	Descrição
TriggerEvent	Dispara um novo evento no canal especificado. Suporta os parâmetros opcionais SocketId (para excluir um cliente) e Info.
TriggerBatchEvents	Dispara múltiplos eventos em uma única requisição HTTP. Aceita um array JSON de objetos de evento.
GetChannels	Fornecer uma lista de todos os canais ativos. Suporta os parâmetros opcionais FilterBy-Prefix e Info.
GetChannel	Fornecer informações sobre um canal específico. Suporta um parâmetro Info opcional.
GetUsers	Fornecer uma lista de todos os usuários conectados a um canal.
TerminateUserConnections	Encerra todas as conexões de um determinado usuário pelo seu user ID.

TriggerEvent

Dispara um evento em um ou mais canais. Requer o nome do evento, o nome do canal e o payload de dados.

Parameter	Descrição
aEventName	O nome do evento a disparar.
aChannel	O nome do canal no qual disparar o evento.

aData	Os dados do evento (string JSON).
aSocketId (opcional)	Um socket ID a excluir do recebimento do evento. Útil para impedir que o remetente receba sua própria mensagem.
alInfo (opcional)	Uma lista de atributos separados por vírgula a incluir na resposta (por exemplo, "subscription_count").

```
// trigger event on a channel
APIPusher->TriggerEvent("my-event", "my-channel", "Hello World");

// trigger event excluding the sender
APIPusher->TriggerEvent("my-event", "my-channel", "Hello World", "123.456");

// trigger event requesting subscription_count in the response
APIPusher->TriggerEvent("my-event", "my-channel", "Hello World", "", "subscription_count");
```

TriggerBatchEvents

Dispara múltiplos eventos em uma única chamada de API, o que é mais eficiente do que fazer requisições separadas para cada evento. O parâmetro batch deve ser uma string JSON contendo um array de objetos de evento, onde cada objeto tem os campos "channel", "name" e "data".

```
APIPusher->TriggerBatchEvents(
    "[{"channel\":\"my-channel\", \"name\":\"my-event\", \"data\":\"hello\"}, {"channel\":\"my-channel-2\", \"name\":\"my-event\", \"data\":\"world\"}]");
```

GetChannels

Retorna uma lista de canais ativos. Suporta parâmetros opcionais para filtrar os resultados e solicitar informações adicionais.

Parameter	Descrição
aFilterByPrefix (opcional)	Filtra os canais por um prefixo de nome (por exemplo, "presence-" para listar apenas canais de presença).
alInfo (opcional)	Uma lista de atributos separados por vírgula a incluir na resposta (por exemplo, "user_count").

```
// get all channels
APIPusher->GetChannels();

// get only presence channels with user count
APIPusher->GetChannels("presence-", "user_count");
```

GetChannel

Retorna informações sobre um canal específico.

Parameter	Descrição
aChannel	O nome do canal sobre o qual obter informações.
alInfo (opcional)	Uma lista separada por vírgulas de atributos a incluir (por exemplo, "user_count,subscription_count").

```
// get channel info
APIPusher->GetChannel("presence-my-channel");

// get channel info with user count
APIPusher->GetChannel("presence-my-channel", "user_count,subscription_count");
```

GetUsers

Retorna uma lista de usuários conectados a um canal de presence. O nome do canal deve incluir o prefixo completo (por exemplo, "presence-my-channel").

```
APIPusher->GetUsers("presence-my-channel");
```

TerminateUserConnections

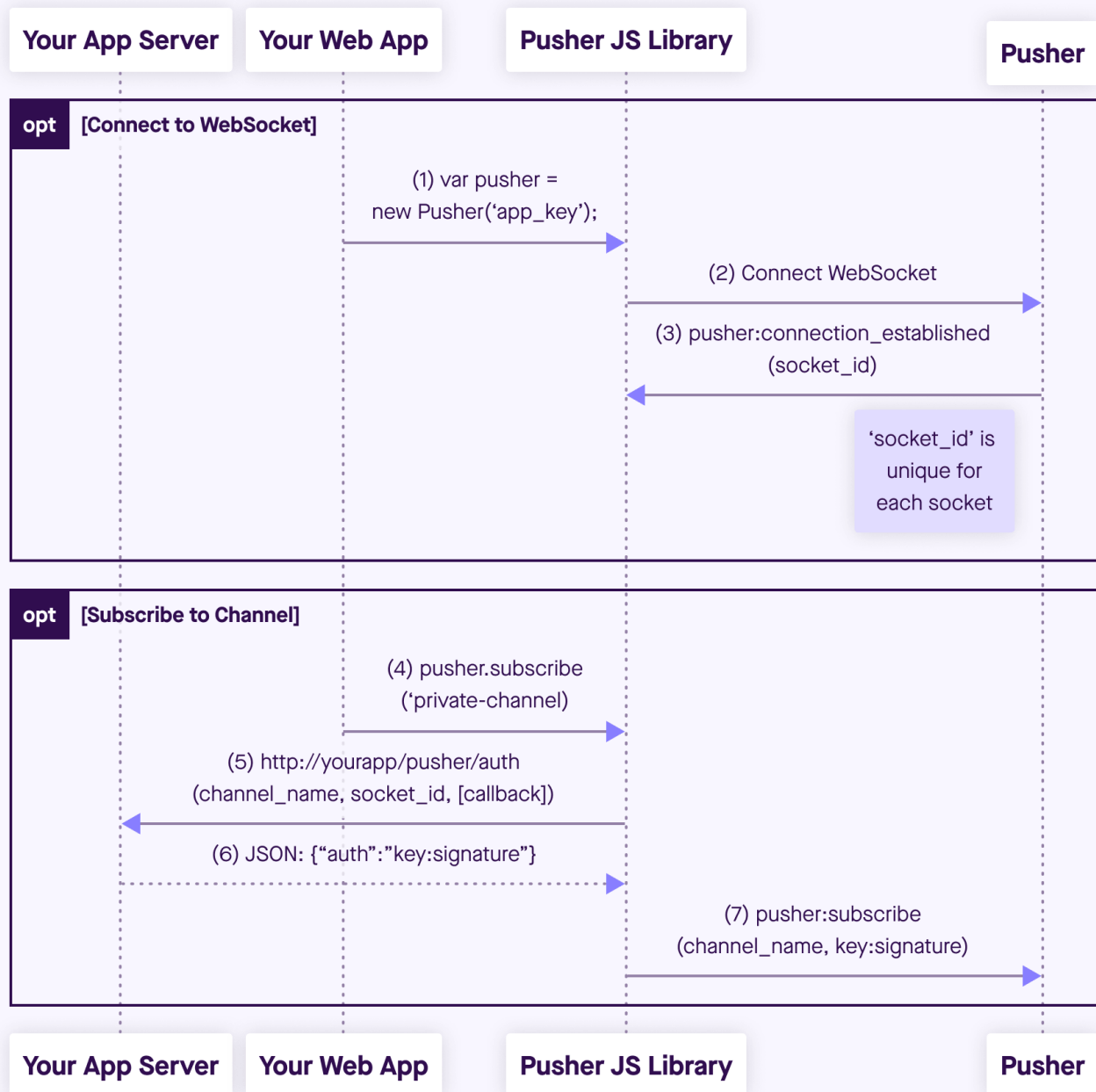
Encerra todas as conexões estabelecidas por um determinado usuário. Isso pode ser utilizado para forçar um usuário específico a se desconectar de todos os canais. O ID do usuário deve corresponder ao "user_id" utilizado quando o usuário assinou um canal de presence.

```
APIPusher->TerminateUserConnections("1234");
```

Custom Authentication

A Pusher só permite assinar canais privados ou de presença; se a conexão fornecer um token de autenticação, isso permite que você restrinja o acesso.

Você pode construir seu próprio fluxo de Autenticação, utilizando o evento **OnPusherAuthentication**; este evento é chamado antes de a mensagem de assinatura ser assinada com a chave secreta fornecida pelo Pusher. Este evento possui 2 parâmetros: uma requisição de autenticação com campos como SocketId, nome do canal... que podem ser utilizados pelo seu próprio servidor de autenticação para autenticar ou não a requisição. Encontre abaixo uma captura de tela que mostra o fluxo de autenticação do pusher



Quando um cliente se conecta ao servidor pusher, ele envia a Key fornecida pelo pusher e o servidor retorna um id de identificação (`socket_id`).

Quando um cliente assina um canal privado (ou de presença), o cliente `sgcWebSockets` usa a Secret Key fornecida pelo pusher para criar uma assinatura que é incluída na mensagem de assinatura. Usando o evento `OnPusherAuthentication`, você pode capturar os campos necessários para assinar a mensagem, implementar seus próprios métodos de autenticação e, se bem-sucedido, retornar a assinatura, e essa assinatura será incluída na mensagem de assinatura e enviada ao servidor.

Exemplo:

```

TsgcWebSocketClient *oClient = new TsgcWebSocketClient(NULL);
TsgcWSAPI_Pusher *oPusher = new TsgcWSAPI_Pusher(NULL);
oPusher->Client = oClient;
oPusher->Cluster = "eu";
Pusher->Name = "js";
Pusher->Version = "4.1";
Pusher->TLS = true;
Pusher->Key = "9c3b7ef25qe97a00116c";
Pusher->Secret = ""; // the secret key is not known by the client, only by the authentication module

oPusher->OnPusherAuthentication = OnPusherAuthenticationEvent;

private void OnPusherAuthenticationEvent(TObject *Sender, TsgcWSPusherRequestAuthentication
*AuthRequest, TsgcWSPusherResponseAuthentication *AuthResponse)
{

```

```
// if the authentication request is successful return the signature
if (CustomAuthentication(AuthRequest->Channel, AuthRequest->SocketID))
{
    AuthResponse->Signature = GetCustomAuthenticationSignature();
}
}
```

O formato da assinatura é:

Canais privados: key:HMAC256(SocketID, ChannelName)

Canais de presença: key: HMAC256(SocketID, ChannelName, Data)

O objeto **TsgcWSPusherResponseAuthentication** fornece as seguintes propriedades:

Property	Descrição
Secret	A chave secreta da Pusher utilizada para computar a assinatura HMAC. Pré-preenchida com Pusher.Secret se configurada.
Assinatura	A assinatura de autenticação computada. Se deixada vazia, o componente a calculará automaticamente utilizando o Secret.
SharedSecret	A chave secreta compartilhada para canais privados criptografados. Necessária ao se inscrever em pscPrivateEncryptedChannel ou pscPrivateEncryptedCacheChannel. Usada para criptografia de ponta a ponta dos dados da mensagem.

API Bitmex

Bitmex

Bitmex é uma exchange de criptomoedas e plataforma de negociação de derivativos.

As seguintes APIs são suportadas:

1. **WebSocket streams:** permite que você faça subscribe a alguns métodos e obtenha dados em tempo real. Os eventos são enviados aos clientes pelo servidor para os assinantes. Usa o WebSocket como protocolo.
2. **REST API:** os clientes podem solicitar dados de mercado e de conta ao servidor. Requer uma API Key e Secret para autenticar e usa HTTPs como protocolo.

Properties

A API da Bitmex tem 2 tipos de métodos: públicos e privados. Os métodos públicos podem ser acessados sem autenticação, por exemplo: obter preços de tickers. Alguns são privados e relacionados a dados do usuário; esses métodos exigem o uso das chaves de API da Bitmex.

- **ApiKey:** você pode solicitar uma nova api key na sua conta Bitmex, basta copiar o valor para esta propriedade.
- **ApiSecret:** é o secret da API, mantenha-o em segurança.
- **TestNet:** se habilitado, conectará à Bitmex Demo Account (por padrão false).
- **HTTPLogOptions:** armazena em um arquivo de texto um log das requisições HTTP
 - **Enabled:** se habilitado, armazenará todas as requisições HTTP da WebSocket API.
 - **FileName:** caminho completo do nome do arquivo onde os logs serão armazenados

Most common uses

- **WebSockets API**
 - [Como se conectar à WebSocket API](#)
 - [Como assinar um canal WebSocket](#)
- **REST API**
 - [Como Colocar uma Ordem na Bitmex](#)

WebSocket API

Subscribe / Unsubscribe

A BitMEX permite assinar dados em tempo real. Esse acesso não tem limite de taxa uma vez conectado e é a melhor maneira de obter os dados mais atualizados para os seus programas. Em alguns tópicos, você pode passar um Symbol para filtrar eventos por símbolo, por exemplo: trades, quotes...

Os seguintes tópicos de assinatura estão disponíveis sem autenticação:

- **btmAnnouncement:** Anúncios do Site
- **btmChat:** chat Trollbox
- **btmConnected:** Estatísticas de usuários/bots conectados
- **btmFunding:** Atualizações das taxas de funding de swap. Enviado a cada intervalo de funding (normalmente 8 h)
- **btmInstrument:** Atualizações de instrumento incluindo turnover e bid/ask
- **btmInsurance:** Atualizações diárias do Insurance Fund
- **btmLiquidation:** Ordens de liquidação à medida que são inseridas no book
- **btmOrderBookL2_25:** Os 25 níveis principais do order book de nível 2

- **btmOrderBookL2**: Livro de ordens completo de nível 2
- **btmOrderBook10**: Top 10 níveis usando o push tradicional de book completo
- **btmPublicNotifications**: Notificações em todo o sistema (utilizadas para mensagens de curta duração)
- **btmQuote**: Topo do book
- **btmQuoteBin1m**: bins de cotação de 1 minuto
- **btmQuoteBin5m**: bins de cotação de 5 minutos
- **btmQuoteBin1h**: bins de cotação de 1 hora
- **btmQuoteBin1d**: bins de cotação de 1 dia
- **btmSettlement**: Liquidações
- **btmTrade**: Negociações ao vivo
- **btmTradeBin1m**: bins de negociação de 1 minuto
- **btmTradeBin5m**: bins de trade de 5 minutos
- **btmTradeBin1h**: bins de trade de 1 hora
- **btmTradeBin1d**: bins de negociação de 1 dia

Os seguintes assuntos requerem autenticação:

- **btmAffiliate**: Status de afiliado, como o total de usuários indicados e % de payout
- **btmExecution**: Execuções individuais; pode haver várias por ordem
- **btmOrder**: Atualizações ao vivo sobre suas ordens
- **btmMargin**: Atualizações do saldo atual da sua conta e dos requisitos de margem
- **btmPosition**: Atualizações sobre suas posições
- **btmPrivateNotifications**: Notificações individuais - atualmente não utilizado
- **btmTransact**: Atualizações de depósito/saque
- **btmWallet**: Dados de saldo de endereço Bitcoin, incluindo o total de depósitos & saques

Exemplo de mensagens recebidas:

```
{
  "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},
  ]
}
```

Autenticação

Se você deseja assinar streams bloqueados por usuário, você deve autenticar-se primeiro. Observe que uma autenticação inválida fechará a conexão.

O uso da API da BitMEX requer uma API Key.

Permanent API Keys podem ser bloqueadas a intervalos de endereço IP e revogadas à vontade sem comprometer suas credenciais principais. Elas também não exigem renovação.

Para usar a autenticação por API Key, você deve gerar uma API Key na sua conta.

Chame o método **Authenticate** antes de assinar qualquer tópico autenticado.

CancelAllAfter (Dead Man's Switch)

O método **CancelAllAfter** implementa o recurso Dead Man's Switch. Quando chamado com um valor de timeout (em milissegundos), instrui o servidor a cancelar todas as ordens abertas se nenhuma chamada subsequente a CancelAllAfter for recebida dentro do período de timeout. Isso é útil para garantir que as ordens sejam canceladas em caso de desconexão de rede.

REST API

Method	Descrição
GetExecutions	Isto retorna todas as transações brutas, o que inclui abertura e cancelamento de ordens, e mudanças de status de ordens.
GetExecutionsTradeHistory	Isto retorna Transactions mais focadas.
GetInstruments	Isto retorna todos os instrumentos e índices, incluindo aqueles que foram liquidados ou não estão listados. Utilize este endpoint se você deseja consultar instrumentos individuais ou usar um filtro complexo.
GetOrders	Para obter apenas as ordens abertas
PlaceOrder	Coloque uma ordem bruta utilizando o objeto TsgcHTTPBitmexOrder.
PlaceMarketOrder	Coloca uma nova ordem MARKET.
PlaceLimitOrder	Coloca uma nova ordem LIMIT.
PlaceStopOrder	Coloca uma nova ordem STOP.
PlaceStopLimitOrder	Coloca uma nova ordem STOPLIMIT.
AmendOrder	Modifica uma ordem existente.
CancelOrder	Cancela uma Ordem ativa.
CancelAllOrders	Cancela Todas as Ordens Ativas.
CancelAllOrdersAfter	Cancelar Todas as Ordens após algum tempo.
ClosePosition	Fecha uma posição aberta.
GetOrderBook	Obter o OrderBook Atual em formato vertical
GetPosition	Obter suas posições.
SetPositionIsolate	Habilita margem isolada ou margem cruzada por posição.
SetPositionLeverage	Escolha a alavancagem por posição.
SetPositionRiskLimit	Atualize seu risk limit.
SetPositionTransferMargin	Transfere patrimônio para dentro ou para fora de uma posição.
GetQuotes	Get Quotes
GetTrades	Get Trades
GetFunding	Obter dados de funding.
GetInsurance	Obtém dados do fundo de seguro.

GetTradeBucketed	Obtém dados de negociação agrupados em buckets (OHLCV) com tamanhos de bin configuráveis.
GetQuoteBucketed	Obtém dados de cotação agrupados (bucketed) com tamanhos de bin configuráveis.
GetSettlement	Obter dados de liquidação.
GetLiquidation	Obtém ordens de liquidação.
GetInstrumentIndices	Obtém índices de instrumentos.
GetInstrumentCompositeIndex	Obter dados de índice composto para instrumentos.
GetStats	Obtém estatísticas de toda a exchange.
GetStatsHistory	Obtém estatísticas históricas do exchange.
GetStatsHistoryUSD	Obter estatísticas históricas de câmbio em USD.
GetUserMargin	Obtém os dados de margem da sua conta.
GetUserWallet	Obtenha as informações da sua carteira.
GetUserWalletHistory	Obtém o histórico de transações da sua carteira.
GetUserWalletSummary	Obter um resumo da sua carteira.

Bitmex | Connect WebSocket API

Para conectar à API WebSocket da Bitmex, basta criar um novo cliente API Bitmex e anexá-lo ao TsgcWebSocketClient.

Veja abaixo um exemplo:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bitmex oBitmex = new TsgcWSAPI_Bitmex();
oBitmex->Client = oClient;
oClient->Active = true;
```

Bitmex | Subscribe WebSocket Channel

A Bitmex oferece uma variedade de canais nos quais você pode se inscrever para obter atualizações em tempo real de dados de mercado, ordens... Encontre abaixo um exemplo de como se inscrever em um Trade Channel:

```
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 trade updates
}
```

Bitmex | Como Colocar Ordens

A REST API da Bitmex oferece endpoints públicos e privados. Os endpoints privados exigem que as mensagens sejam assinadas para aumentar a segurança das transações.

Primeiro você deve fazer login na sua conta Bitmex e criar uma nova API; você obterá os seguintes valores:

- ApiKey
- ApiSecret

Esses campos devem ser configurados na propriedade Bitmex do componente cliente da Bitmex API. Uma vez configurados, você pode começar a fazer requisições privadas à Bitmex REST API.

Tipos de Ordem

Todas as ordens exigem um símbolo. Todos os outros campos são opcionais, exceto quando especificado de outra forma.

Estes são os ordTypes válidos:

- **Limit:** O tipo de ordem padrão. Especifique um orderQty e um price.
- **Market:** Uma Market order tradicional. Uma Market order será executada até ser preenchida ou até que seu preço de falência seja atingido, ponto no qual será cancelada.
- **Stop:** Uma ordem Stop Market. Especifique um orderQty e stopPx. Quando o stopPx é atingido, a ordem será inserida no book.
 - Em ordens de venda, a ordem é acionada se o preço de acionamento for menor que o stopPx. Em compras, maior.
 - Nota: as ordens Stop não consomem margem até serem acionadas. Certifique-se de que a margem necessária esteja disponível em sua conta para que possa ser acionada totalmente.
 - As Close Stops não exigem um orderQty. Veja as Execution Instructions abaixo.
- **StopLimit:** Como uma Stop Market, mas insere uma Limit order em vez de uma Market order. Especifique orderQty, stopPx e price.
- **MarketIfTouched:** Semelhante a um Stop, mas os disparos são feitos na direção oposta. Útil para ordens Take Profit.
- **LimitIfTouched:** Como acima; use para ordens Take Profit Limit.
- **Pegged:** As pegged orders permitem que os usuários enviem um limit price relativo ao preço de mercado atual. Especifique um pegPriceType e pegOffsetValue.
 - As pegged orders devem ter um execInst Fixed. Isso significa que o limit price é definido no momento em que a ordem é aceita e não muda conforme o preço de referência muda.
 - PrimaryPeg: O preço é definido em relação ao near touch price.
 - MarketPeg: O preço é definido em relação ao far touch price.
 - Um pegPriceType enviado sem ordType é tratado como uma Pegged order.

Execution Instructions

Os seguintes execInsts são suportados. Se usar múltiplos, separe com uma vírgula (por exemplo, LastPrice,Close).

- **ParticipateDoNotInitiate:** Também conhecida como ordem Post-Only. Se esta ordem fosse executada na colocação, ela será cancelada em vez disso. Isso visa protegê-lo do far touch se movendo em sua direção enquanto a ordem está em trânsito. Não se destina a especular sobre o far touch se afastando após o envio; consideramos tal comportamento abusivo e monitoramos por isso.
- **MarkPrice, LastPrice, IndexPrice:** Utilizados por ordens stop e if-touched para determinar o preço de acionamento. Utilize apenas um. Por padrão, MarkPrice é utilizado. Também utilizado para ordens Pegged para definir o valor de LastPeg.
- **ReduceOnly:** Uma ordem ReduceOnly só pode reduzir sua posição, não aumentá-la. Se você tiver uma ordem limite ReduceOnly que permanece no order book enquanto a posição é reduzida por outras ordens, então sua quantidade de ordem será reduzida ou cancelada. Se houver várias ordens ReduceOnly, a menos agressiva será reduzida primeiro.

- **Close:** Close implica ReduceOnly. Uma ordem Close cancelará outras ordens limit ativas com o mesmo lado e símbolo se a quantidade aberta exceder a posição atual. Isto é útil para stops: ao cancelar essas ordens, um Close Stop tem garantia de ter a margem necessária para executar, e só pode executar até o tamanho total da sua posição. Se orderQty não for especificado, uma ordem Close tem uma orderQty igual ao tamanho da sua posição atual.
 - Note que uma ordem Close sem uma orderQty requer um side, para que a BitMEX saiba se deve disparar acima ou abaixo do stopPx.
- **LastWithinMark:** Usado por ordens stop com LastPrice para permitir gatilhos de stop apenas quando:
 - Para ordens Sell Stop Market / Stop Limit
 - Last Price $\leq$ Stop Price
 - Last Price $\geq$ Mark Price $\times (1 - 5\%)$
 - Para ordens Buy Stop Market / Stop Limit:
 - Last Price $\geq$ Stop Price
 - Last Price $\leq$ Mark Price $\times (1 + 5\%)$
- **Fixed:** As ordens pegged devem ter um execInst de Fixed. Isso significa que o limit price é definido no momento em que a ordem é aceita e não muda conforme o reference price muda.

Pegged Orders

As ordens pegged permitem que os usuários submetam um preço limite relativo ao preço de mercado atual. O preço limite é definido uma vez quando a ordem é submetida e não muda com o preço de referência. Este tipo de ordem não se destina à especulação sobre o far touch se afastando após a submissão. Consideramos tal comportamento abusivo e o monitoramos.

Ordens pegged têm um ordType de Pegged e um execInst de Fixed.

Um pegPriceType e pegOffsetValue também devem ser submetidos:

- PrimaryPeg - o preço é definido em relação ao preço near touch
- MarketPeg - o preço é definido em relação ao far touch price

Trailing Stop Pegged Orders

Utilize pegPriceType de TrailingStopPeg para criar Trailing Stops.

O preço é definido no momento do envio e atualiza uma vez por segundo se o preço subjacente (last/mark/index) tiver se movido mais de 0,1%. O stopPx então se move conforme o mercado se afasta do peg e congela conforme o mercado se aproxima dele.

Use pegOffsetValue para definir o stopPx da sua ordem. O peg é definido para o preço de gatilho especificado no execInst (padrão MarkPrice). Use um offset negativo para ordens stop-sell e buy-if-touched.

Requer ordType: Stop, StopLimit, MarketIfTouched, LimitIfTouched.

Trailing Stops

Você pode usar pegPriceType de 'TrailingStopPeg' para criar Trailing Stops. O stopPx vinculado se moverá conforme o mercado se afasta do peg e congelará conforme o mercado se move em sua direção.

Para utilizar, combine com pegOffsetValue para definir o stopPx da sua ordem. O peg é definido com o preço de disparo especificado no execInst (padrão 'MarkPrice'). Utilize um offset negativo para ordens stop-sell e buy-if-touched.

Requer ordType: 'Stop', 'StopLimit', 'MarketIfTouched', 'LimitIfTouched'.

Acompanhando Suas Ordens

Se você quiser acompanhar os IDs de ordem por conta própria, defina um clOrdID exclusivo por ordem. Este clOrdID retornará como uma propriedade na ordem e em quaisquer execuções relacionadas (incluindo no WebSocket) e pode ser usado para obter ou cancelar a ordem. O comprimento máximo é de 36 caracteres.

Exemplos:

```
// 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

A Bitfinex é uma das maiores e mais avançadas plataformas de negociação de criptomoedas do mundo. Os usuários podem trocar Bitcoin, Ethereum, Ripple, EOS, Bitcoin Cash, Iota, NEO, Litecoin, Ethereum Classic...

A versão da API WebSocket da Bitfinex é 2.0

Cada mensagem enviada e recebida pelo canal WebSocket da Bitfinex é codificada no formato JSON

Um símbolo pode ser um par de negociação ou uma moeda de margem:

- Os símbolos de pares de negociação são formados prefixando um "t" antes do par (ou seja tBTCUSD, tETHUSD).
- Os símbolos de moeda de margem são formados prefixando um "f" antes da moeda (ou seja, fUSD, fBTC, ...)

Após uma conexão bem-sucedida, o evento **OnBitfinexConnect** é gerado e você obtém o número da versão da API Bitfinex como parâmetro.

Você pode chamar o método **Ping** para testar a conexão com o servidor.

Se o servidor enviar alguma informação, isso pode ser tratado utilizando o evento **OnBitfinexInfoMessage**, onde um Code e uma Message são parâmetros com informações sobre a mensagem enviada pelo servidor. Exemplos de códigos:

```
20051 : Parar/Reiniciar o WebSocket Server (por favor, reconecte)
20060 : Entrando em modo de Manutenção. Por favor, pause qualquer atividade e retome após receber a
mensagem de informação 20061 (deve levar no máximo 120 segundos).
20061 : Manutenção encerrada. Você pode retomar a atividade normal. Aconselha-se cancelar a inscrição/
inscrever-se novamente em todos os canais.
```

Em caso de erro, **OnBitfinexError** será gerado, e informações sobre o erro fornecidas. Exemplos de códigos de erro:

```
10000 : Evento desconhecido
10001 : Par desconhecido
```

Para alterar a configuração, chame o método **Configuration** e passe como parâmetro um dos seguintes flags:

```
CS_DEC_S = 8; // Enable all decimal as strings.
CS_TIME_S = 32; // Habilita todos os horários como strings de data.
CS_SEQ_ALL = 65536; // Habilita sequenciamento RECURSO BETA
CHECKSUM = 131072; // Habilita o checksum para cada iteração do book. Verifica as 25 entradas do topo
de cada lado do book. O checksum é um int com sinal.
```

Inscrever-se em Canais Públicos

Há canais que são públicos e não há necessidade de autenticar contra o servidor. Todas as mensagens são geradas no evento **OnBitfinexUpdate**.

SubscribeTicker

O ticker é uma visão geral de alto nível do estado do mercado. Ele mostra o melhor bid e ask atuais, bem como o último preço de trade. Ele também inclui informações como o volume diário e o quanto o preço se moveu no último dia.


```
// 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

Este canal envia uma mensagem de trade sempre que uma negociação ocorre na Bitfinex. Ela inclui todos os detalhes pertinentes da negociação, como preço, tamanho e horário.

```
// on trading pairs (ex. tBTCUSD)
[
  CHANNEL_ID,
  [
    [
      ID,
      MTS,
      AMOUNT,
      PRICE
    ],
    ...
  ]
]
// on funding currencies (ex. fUSD)
[
  CHANNEL_ID,
  [
    [
      ID,
      MTS,
      AMOUNT,
      RATE,
      PERIOD
    ],
    ...
  ]
]
```

SubscribeOrderBook

O canal Order Books permite que você acompanhe o estado do order book da Bitfinex. Ele é fornecido em base agregada por preço, com precisão personalizável. Após receber a resposta, você receberá um snapshot do book, seguido de atualizações a cada mudança no book.

```
// on trading pairs (ex. tBTCUSD)
[
  CHANNEL_ID,
  [
    [
      PRICE,
      COUNT,
      AMOUNT
    ],
    ...
  ]
]

// on funding currencies (ex. fUSD)
[
  CHANNEL_ID,
  [
    [
      RATE,
      PERIOD,
      COUNT,
      AMOUNT
    ],
    ...
  ]
]
```

SubscribeRawOrderBook

Estes são os livros mais granulares.

```
// on trading pairs (ex. tBTCUSD)
[
  CHANNEL_ID,
  [
    [
      ORDER_ID,
      PRICE,
      AMOUNT
    ],
    ...
  ]
]

// on funding currencies (ex. fUSD)
[
  CHANNEL_ID,
  [
    [
      OFFER_ID,
      PERIOD,
      RATE,
      AMOUNT
    ],
    ...
  ]
]
```

SubscribeCandles

Fornece uma forma de acessar informações de candle de gráficos. Time Frames:

1m : um minuto
 5m : cinco minutos
 15m : 15 minutos
 30m : 30 minutos
 1h : uma hora
 3h : 3 horas
 6h : 6 horas
 12h : 12 horas
 1D : um dia
 7D : uma semana
 14D : duas semanas
 1M : um mês

```
[
  CHANNEL_ID,
  [
    [
      MTS,
      OPEN,
      CLOSE,
      HIGH,
      LOW,
      VOLUME
    ],
    ...
  ]
]
```

Subscribe Authenticated Channels

Este canal permite que você se mantenha atualizado com o status da sua conta. Você pode receber atualizações sobre suas posições, seus saldos, suas ordens e seus trades.

Use o método **Authenticate** para autenticar contra o servidor e definir os parâmetros necessários.

Uma vez autenticado, você receberá atualizações de: Ordens, posições, trades, ofertas de funding, créditos de funding, empréstimos de funding, carteiras, informações de saldo, informações de margem, informações de funding, trades de funding...

Você pode requisitar o método **UnAuthenticate** se quiser fazer logoff do servidor.

API Kucoin

Kucoin

A Kucoin é uma exchange de criptomoedas internacional e multilíngue. Ela oferece algumas APIs para acessar dados da Kucoin. As seguintes APIs são suportadas:

1. **WebSocket streams:** permite que você faça subscribe a alguns métodos e obtenha dados em tempo real. Os eventos são enviados aos clientes pelo servidor para os assinantes. Usa o WebSocket como protocolo.
2. **REST API:** os clientes podem requisitar ao servidor dados de mercado e de conta. Requer uma API Key, Secret e Passphrase para autenticar e usa HTTPs como protocolo.

Properties

A API Kucoin tem 2 tipos de métodos: públicos e privados. Os métodos públicos podem ser acessados sem autenticação, exemplo: obter preços de ticker. Os métodos privados relacionados a dados do usuário exigem o uso das API keys da Kucoin.

- **ApiKey:** você pode solicitar uma nova api key na sua conta kucoin; basta copiar o valor para esta propriedade.
- **ApiSecret:** O API secret é necessário apenas para a REST_API; a api websocket requer apenas a ApiKey para alguns métodos.
- **Passphrase:** string necessária para conectar aos servidores Kucoin.
- **Sandbox:** se habilitado, conectará à Conta Demo da Kucoin (por padrão false).
 - **HTTPLogOptions:** armazena em um arquivo de texto um log das requisições HTTP
 - **Enabled:** se habilitado, armazenará todas as requisições HTTP da WebSocket API.
 - **FileName:** caminho completo do nome do arquivo onde os logs serão armazenados
 - **REST:** armazena em um arquivo de texto um log das requisições da REST API
 - **Enabled:** se habilitado, armazenará todas as requisições HTTP da REST API.
 - **FileName:** caminho completo do nome do arquivo onde os logs serão armazenados.

Most common uses

- **WebSockets API**
 - [Como Conectar à WebSocket API](#)
 - [Como Assinar um WebSocket Channel](#)
- **REST API**
 - [Como Obter Dados de Mercado](#)
 - [Como Utilizar a Private REST API](#)
 - [Como Negociar Spot](#)
 - [Private Requests Time](#)

WebSocket Feed

Para assinar as mensagens de um canal a partir de um determinado servidor, o lado do cliente deve enviar uma mensagem de assinatura ao servidor.

Se a assinatura for bem-sucedida, o sistema enviará mensagens de ack a você, quando a resposta for definida como true.

```
{
  "id": "1545910660739",
  "type": "ack"
}
```

Enquanto houver mensagens de tópico geradas, o sistema enviará as mensagens correspondentes ao lado do cliente.

Os seguintes métodos de Assinatura / Cancelamento de assinatura são suportados.

Canais Públicos

Method	Parâmetros	Descrição
SubscribeSymbolTicker	Symbol	Inscreva-se neste tópico para obter o push de alterações de BBO. Se não houver alteração dentro de um segundo, ele não será enviado. Será enviado a cada 100ms com o BBO mais recente. Se não houve alteração em relação aos últimos dados, ele não será enviado.
SubscribeAllSymbolsTicker		Assine este tópico para obter o push de todas as mudanças de BBO dos símbolos de mercado.
SubscribeSymbolSnapshot	Symbol	Assina para obter dados de snapshot de um único símbolo. Os dados de snapshot são enviados em intervalos de 2 segundos.
SubscribeMarketSnapshot	Market	Assine este tópico para obter os dados de snapshot de todo o mercado. Os dados de snapshot são enviados em intervalos de 2 segundos.
SubscribeLevel2MarketData	Symbol	Assine este tópico para obter dados do livro de ordens Level2. Quando a assinatura do websocket for bem-sucedida, o sistema enviaria a você os dados de mudança incremental enviados pelo websocket.
SubscribeLevel2_5BestAskBid	Symbol	O sistema retornará os dados das 5 melhores ordens de ask/bid, que são os dados de snapshot a cada 100 milissegundos (em outras palavras, os dados das 5 melhores ordens de ask/bid retornados a cada 100 milissegundos em tempo real).
SubscribeLevel2_50BestAskBid	Symbol	O sistema retornará os dados das 50 melhores ordens de ask/bid, que são os dados de snapshot a cada 100 milissegundos (em outras palavras, os dados das 50 melhores ordens de ask/bid retornados a cada 100 milissegundos em tempo real).
SubscribeKlines	Symbol	Assine este tópico para obter os dados de K-Line.
SubscribeMatchExecutionData	Symbol	Assine este tópico para obter o fluxo de dados de eventos de correspondência do Level 3. Para cada ordem negociada, o sistema enviará as mensagens de correspondência no formato a seguir.
SubscribeIndexPrice	Symbol	Inscreva-se neste tópico para obter o preço de índice da negociação com margem.
SubscribeMarkPrice	Symbol	Assine este tópico para obter o mark price para trading de margem.
SubscribeOrderBookChanged	Symbol	Assine este tópico para obter as alterações do order book no trade de margem.
SubscribeLevel1	Symbol	Assina os dados de melhor bid/ask de Level 1 para um símbolo.

Se o parâmetro ACK for enviado como true, após uma assinatura / cancelamento de assinatura bem-sucedido, o cliente recebe uma mensagem sobre isso.

Private Channels

Requer uma ApiKey válida obtida da sua conta Kucoin. A ApiKey, o ApiSecret e a Passphrase devem ser definidos na propriedade Kucoin do componente de API cliente.

Os dados a seguir são enviados ao cliente toda vez que há uma alteração. Não há necessidade de assinar nenhum método; isso é feito automaticamente se você definir uma ApiKey válida.

Method	Descrição
SubscribeTradeOrders	Este tópico enviará todos os eventos de alteração das suas ordens.
SubscribeAccountBalance	Você receberá esta mensagem quando o saldo de uma conta for alterado. A mensagem contém os detalhes da alteração.
SubscribePositionStatus	O sistema enviará o evento de mudança quando o status da posição mudar.
SubscribeMarginTradeOrders	O sistema enviará esta mensagem aos lenders quando a ordem entrar no order book.
SubscribeStopOrder	Quando uma stop order é recebida pelo sistema, você receberá uma mensagem com o tipo "open". Isso significa que esta ordem entrou no sistema e aguardou para ser acionada.
SubscribeTradeOrdersV2	Assine o canal de ordens de negociação V2 para notificações aprimoradas de atualização de ordem.
SubscribeCrossMarginPosition	Assine atualizações de posição de cross margin. O sistema enviará o evento de mudança quando a posição de cross margin mudar.
SubscribeIsolatedMarginPosition	Assine atualizações de posição em margem isolada. O sistema enviará o evento de alteração quando a posição em margem isolada mudar.

REST API

Todos os endpoints retornam um objeto ou array JSON.

EndPoints da API Pública

Esses endpoints podem ser acessados sem nenhuma autorização.

General EndPoints

Method	Parâmetros	Descrição
GetServiceStatus		Testa a conectividade com a Rest API e obtém o Status do Serviço
GetServerTime		Testar a conectividade com a Rest API e obter a hora atual do servidor.

Market Data EndPoints

Method	Parâmetros	Descrição
GetSymbolList	Market	Faça uma requisição através deste endpoint para obter uma lista de pares de moedas disponíveis para negociação. Se

		você quiser obter as informações de mercado do símbolo de negociação
GetTicker	Symbol	Faça a requisição por este endpoint para obter Market Data de Nível 1. O valor retornado inclui o melhor preço e tamanho de bid, o melhor preço e tamanho de ask, bem como o último preço negociado e o último tamanho negociado.
GetAllTickers		Solicita os tickers de mercado para todos os pares de negociação do mercado (incluindo volume de 24h).
Get24hrStats	Symbol	Solicite por este endpoint para obter as estatísticas do ticker especificado nas últimas 24 horas.
GetMarketList		Faça uma requisição através deste endpoint para obter a moeda de transação de todo o mercado de negociação.
GetPartOrder-Book20	Symbol	Requisita por meio deste endpoint para obter uma lista de ordens abertas de um símbolo. O order book de Level-2 inclui todos os bids e asks (agregados por preço); este nível retorna apenas um tamanho para cada preço ativo (como se houvesse apenas uma única ordem para aquele preço). O sistema retornará 20 itens de dados (dados de ask e bid) no order book.
GetPartOrder-Book100	Symbol	Faça a requisição por este endpoint para obter uma lista de ordens abertas de um símbolo. O livro de ordens de Nível 2 inclui todos os bids e asks (agregados por preço); este nível retorna apenas um tamanho para cada preço ativo (como se houvesse apenas uma única ordem para aquele preço). O sistema retornará 100 itens de dados (dados de ask e bid) no livro de ordens.
GetFullOrder-Book	Symbol	Requisite através deste endpoint para obter o order book do símbolo especificado. O order book Level 2 inclui todos os bids e asks (agregados por preço). Este nível retorna apenas um tamanho agregado para cada preço (como se houvesse apenas uma única ordem para esse preço). Esta API retornará dados com profundidade completa.
GetKLines	Symbol	Faça uma requisição por meio deste endpoint para obter a kline do símbolo especificado. Os dados são retornados em buckets agrupados com base no tipo solicitado.
GetCurrencies		Requisite através deste endpoint para obter a lista de moedas.
GetCurrency-Detail	Currency	Requisite através deste endpoint para obter os detalhes de uma moeda especificada
GetFiatPrice		Requisite através deste endpoint para obter os detalhes de uma moeda especificada
GetPartOrder-Book1	Symbol	Faça a requisição por este endpoint para obter o melhor bid/ask de Nível 1 de um símbolo.

Private API EndPoints

Requer uma APIKey e uma APISecret para ser autorizado pelo servidor.

User EndPoints

Method	Parâmetros	Descrição
GetAllSubAccounts		Você pode obter as informações de usuário de todos os sub-usuários por meio desta interface.
GetListAccounts		Obter uma lista de contas.
GetAccount	AccountId	Informações de uma única conta. Utilize este endpoint quando você conhecer o accountId.
GetAccountBalanceSubAccount	SubUserId	Este endpoint retorna as informações da conta de um sub-usuário especificado pelo subUserId.
InnerTransfer		Este endpoint da API pode ser usado para transferir fundos entre contas internamente. Os usuários podem transferir fundos entre sua conta principal, conta de negociação, conta de margem cruzada e conta de margem isolada gratuitamente. A transferência de fundos da conta principal, conta de margem cruzada e conta de negociação para a conta de futuros é suportada, mas a transferência de fundos das contas de futuros para outras contas não é suportada.
GetDepositAddresses	Currency	Obtém os endereços de depósito de uma moeda.
CreateDepositAddress	Currency	Criar um novo endereço de depósito para uma moeda.
GetDepositList		Obter histórico de depósitos.
GetAccountLedgers		Obter as entradas do ledger da conta.
GetTradeFees	Symbols	Obtém as taxas de trade para os símbolos especificados.

Endpoints de saque

Method	Parâmetros	Descrição
GetWithdrawalsList		Obtém uma lista dos Withdrawals.
GetHistoricalWithdrawalsList		Lista de saques históricos da KuCoin V1.
GetWithdrawalsQuotas	Currency	Get Withdrawals Quotas
ApplyWithdraw	Currency, Address, Amount	Cria um Withdraw
CancelWithdraw	WithdrawalId	Somente requisições de saque com status PROCESSING podem ser canceladas.

Trade Endpoints

Method	Parâmetros	Descrição
PlaceOrder		Você pode colocar dois tipos de ordens: limit e market. As ordens só podem ser colocadas se sua conta tiver fundos suficientes. Uma vez colocada uma ordem, os fundos da sua conta ficarão retidos pela duração da ordem. Quanto e

		quais fundos ficam retidos depende do tipo de ordem e dos parâmetros especificados.
PlaceMarketOrder		Coloca uma Ordem a Mercado.
PlaceLimitOrder		Coloca uma Ordem Limit.
PlaceMarginOrder		Coloca uma Margin Order.
CancelOrder		Cancela uma Ordem pelo Order Id.
CancelOrderByClientOid		Cancela uma ordem pelo Client Order Id.
CancelAllOrders		Cancela todas as ordens abertas.
ListOrders		Faça uma requisição por este endpoint para obter sua lista de ordens atual. Os itens são paginados e ordenados para mostrar os mais recentes primeiro
GetRecentOrders		Requisite por meio deste endpoint para obter 1000 ordens nas últimas 24 horas.
GetOrder		Requisita por meio deste endpoint para obter as informações de uma única ordem pelo order ID.
GetOrderByClientOid		Solicite por este endpoint para obter as informações de uma única ordem pelo Client order ID.
ListFills		Solicite por este endpoint para obter os fills recentes.
GetRecentFills		Requisita por meio deste endpoint para obter uma lista de 1000 fills nas últimas 24 horas.
PlaceStopOrder		Coloca uma Stop Order.
PlaceStopMarketOrder		Coloca uma Ordem Stop Market.
PlaceStopLimitOrder		Coloca uma ordem Stop Limit.
CancelStopOrder		Cancela uma Open Stop Order pelo Order Id
CancelStopOrderByClientOid		Cancela uma Open Stop Order pelo Client Order Id
CancelAllStopOrders		Cancel All Stop Orders
GetStopOrder		Faça uma requisição através desta interface para obter informações de uma ordem stop pelo ID da ordem.
GetStopOrderByClientOid		Requisita por meio desta interface para obter informações de uma ordem stop pelo Client order ID.
ListStopOrders		Faça uma requisição neste endpoint para obter a sua lista atual de stop orders não disparadas. Os itens são paginados e ordenados para mostrar os mais recentes primeiro.
PlaceHFOOrder		Coloca uma ordem de alta frequência.
CancelHFOOrder		Cancela uma ordem de alta frequência pelo order ID.
CancelHFOOrderByClientOid		Cancela uma ordem de alta frequência pelo ID da ordem do cliente.
CancelAllHFOOrders		Cancelar todas as ordens de alta frequência.
GetHFActiveOrders		Obter ordens ativas de alta frequência.

GetHFDoneOrders		Obter ordens de alta frequência concluídas.
GetHFOrder		Obtém uma ordem específica de alta frequência pelo ID da ordem.

Eventos

As mensagens da Kucoin são recebidas no componente `TsgcWebSocketClient`; você pode utilizar os seguintes eventos:

OnConnect

Após uma conexão bem-sucedida ao servidor Kucoin.

OnDisconnect

Após uma desconexão do servidor Kucoin

OnMessage

Mensagens enviadas pelo servidor ao cliente são tratadas neste evento.

OnError

Se houver algum erro no protocolo, este evento será chamado.

OnException

Se houver uma exceção não tratada, este evento será chamado.

Adicionalmente, há um evento específico no Componente da API Kucoin, chamado **OnKucoinHTTPException**, que é gerado toda vez que há um erro ao chamar uma Requisição HTTP (REST API ou WebSocket Feeds).

Kucoin | Connect WebSocket API

Para se conectar à WebSocket API da Kucoin, basta criar um novo cliente API da Kucoin e anexá-lo ao TsgcWebSocketClient.

Veja abaixo um exemplo:

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_Kucoin *oKucoin = new TsgcWSAPI_Kucoin();
oKucoin->Client = oClient;
oClient->Active = true;
```

Kucoin | Subscribe WebSocket Channel

A Kucoin oferece uma variedade de canais nos quais você pode se inscrever para obter atualizações em tempo real de dados de mercado, ordens... Encontre abaixo um exemplo de como se inscrever em um 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 | Get Market Data

A Kucoin oferece Market Data públicos através de endpoints REST; quando você chama um desses endpoints, obtém um snapshot dos dados de mercado solicitados.

Os Market Data Endpoints não exigem autenticação, então estão disponíveis livremente para todos os usuários.

Exemplo: para obter o snapshot do ticker BTC-USDT, faça a seguinte chamada

```
TsgcWSAPI_Kucoin *oKucoin = new TsgcWSAPI_Kucoin(this);
ShowMessage(oKucoin->REST_API->GetTicker("
BTC-USDT
"));
```

Kucoin | Private REST API

A REST API da Kucoin oferece endpoints públicos e privados. Os endpoints Privados exigem que as mensagens sejam assinadas para aumentar a segurança das transações.

Primeiro você deve fazer login na sua conta Kucoin e criar uma nova API, você obterá os seguintes valores:

- ApiKey
- ApiSecret
- Passphrase

Esses campos devem ser configurados na propriedade Kucoin do componente cliente da API Kucoin.

Uma vez configurado, você pode começar a fazer requisições privadas à REST API do Kucoin Pro

*As Private Requests exigem que sua máquina local tenha a hora local sincronizada; caso contrário, as requisições serão rejeitadas pelo servidor Kucoin. Confira o artigo a seguir sobre isso, [Kucoin Private Requests Time](#).

```
TsgcWSAPI_Kucoin *oKucoin = new TsgcWSAPI_Kucoin(this);
oKucoin->Kucoin->ApiKey = "<your api key>";
oKucoin->Kucoin->ApiSecret = "<your api secret>";
oKucoin->Kucoin->Passphrase = "<your passphrase>";
ShowMessage(oKucoin->REST_API->GetListAccounts());
```

Kucoin | Trade Spot

A Kucoin permite que você negocie no mercado spot utilizando sua REST API.

Configuração

Primeiro você deve criar uma **API Key** na sua conta Kucoin e adicionar privilégios para negociar com Spot.

Uma vez feito isso, você pode iniciar o spot trading.

Primeiro, **defina sua ApiKey, ApiSecret e Passphrase** no componente Kucoin Client, isto será utilizado para assinar as requisições enviadas ao servidor Kucoin.

Place an Order

Para colocar uma nova ordem, basta chamar o método **REST_API.PlaceOrder** do Componente Cliente Kucoin.

Dependendo do tipo da ordem (market, limit...), a API requer mais ou menos campos.

Parâmetros

Pa-ram	ty-pe	Descrição
clien-tOid	String	Id de ordem único criado pelos usuários para identificar suas ordens, por exemplo, UUID.
side	String	buy ou sell
symbol	String	um código de símbolo de negociação válido. Por exemplo, ETH-BTC
type	String	[Opcional] limit ou market (o padrão é limit)
remark	String	[Opcional] observação para a ordem, o comprimento não pode exceder 100 caracteres utf8
stp	String	[Opcional] self trade prevention , CN , CO , CB ou DC
tra-deType	String	[Opcional] O tipo de trading: TRADE (Spot Trade) , MARGIN_TRADE (Margin Trade). O padrão é TRADE . Nota: Para melhorar o desempenho do sistema e acelerar a colocação e o processamento de ordens, a KuCoin adicionou uma nova interface para a colocação de ordens de margem. Para os traders que ainda utilizam a interface atual, mude para a nova o mais rápido possível. A atual deixará de aceitar margin orders a partir de 1º de maio de 2021 (UTC). Na ocasião, a KuCoin notificará os usuários por meio do anúncio; preste atenção a ele.

PARÂMETROS DA ORDEM LIMIT

Param	type	Descrição
price	String	preço por moeda base
size	String	quantidade de moeda base a comprar ou vender
timeInForce	String	[Opcional] GTC , GTT , IOC , ou FOK (o padrão é GTC), leia Time In Force .
cancelAfter	long	[Opcional] cancela após n segundos, requer que timeInForce seja GTT

Param	type	Descrição
postOnly	boolean	[Opcional] Flag de post only, inválido quando timeInForce é IOC ou FOK
hidden	boolean	[Opcional] A ordem não será exibida no order book
iceberg	boolean	[Opcional] Apenas uma parte da ordem é exibida no order book
visibleSize	String	[Opcional] O tamanho máximo visível de uma ordem iceberg

MARKET ORDER PARAMETERS

Param	type	Descrição
size	String	[Opcional] Quantia desejada na moeda base
funds	String	[Opcional] A quantidade desejada de moeda de cotação a usar

Ao enviar uma ordem, há 2 possibilidades:

1. **Successful:** a função PlaceOrder retorna a mensagem enviada pelo servidor Kucoin.
2. **Error:** a exceção é retornada no evento OnKucoinHTTPException.

Enviar Ordem a Mercado 1 BTC-USDT

```
TsgcWSAPI_Kucoin *oKucoin = new TsgcWSAPI_Kucoin(this);
oKucoin->Kucoin->ApiKey = "<api key>";
oKucoin->Kucoin->ApiSecret = "<api secret>";
oKucoin->Kucoin->Passphrase = "<passphrase>";
ShowMessage(oKucoin->REST_API->

PlaceMarketOrder

(
    kosBuy, "BTC-USDT", 1
));
```

Colocar Ordem Limite 1 BTC-USDT a 40000

```
TsgcWSAPI_Kucoin *oKucoin = new TsgcWSAPI_Kucoin(this);
oKucoin->Kucoin->ApiKey = "<api key>";
oKucoin->Kucoin->ApiSecret = "<api secret>";
oKucoin->Kucoin->Passphrase = "<passphrase>";
ShowMessage(oKucoin->REST_API->

PlaceLimitOrder

(
    kosBuy, "BTC-USDT", 1, 40000
));
```


Kucoin | Hora das Requisições Privadas

Quando você faz uma requisição privada à Kucoin, a mensagem é assinada para aumentar a segurança das requisições. A mensagem usa a hora local e a envia dentro da mensagem assinada; se a hora local tiver uma diferença maior que 5 segundos em relação aos servidores da Kucoin, a requisição será rejeitada. Portanto, é importante verificar se sua hora local está sincronizada, o que você pode fazer usando o método de sincronização de hora do seu SO.

Você pode verificar o horário do servidor Kucoin chamando o método **GetServerTime**, que retornará o horário do servidor Kucoin

API Kucoin Futures

Kucoin Futures

A Kucoin é uma exchange de criptomoedas internacional e multilíngue. Ela oferece algumas APIs para acessar dados da Kucoin. As seguintes APIs são suportadas:

1. **WebSocket streams:** permite que você faça subscribe a alguns métodos e obtenha dados em tempo real. Os eventos são enviados aos clientes pelo servidor para os assinantes. Usa o WebSocket como protocolo.
2. **REST API:** os clientes podem requisitar ao servidor dados de mercado e de conta. Requer uma API Key, Secret e Passphrase para autenticar e usa HTTPs como protocolo.

Properties

A API da Kucoin tem 2 tipos de métodos: públicos e privados. Os métodos públicos podem ser acessados sem autenticação, exemplo: obter preços de ticker. Os métodos privados e relacionados a dados do usuário exigem o uso de chaves de API da Kucoin.

- **ApiKey:** você pode solicitar uma nova api key na sua conta kucoin; basta copiar o valor para esta propriedade.
- **ApiSecret:** O API secret é necessário apenas para a REST_API; a api websocket requer apenas a ApiKey para alguns métodos.
- **Passphrase:** string necessária para conectar aos servidores Kucoin.
- **Sandbox:** se habilitado, conectará à Conta Demo da Kucoin (por padrão false).
 - **HTTPLogOptions:** armazena em um arquivo de texto um log das requisições HTTP
 - **Enabled:** se habilitado, armazenará todas as requisições HTTP da WebSocket API.
 - **FileName:** caminho completo do nome do arquivo onde os logs serão armazenados
 - **REST:** armazena em um arquivo de texto um log das requisições da REST API
 - **Enabled:** se habilitado, armazenará todas as requisições HTTP da REST API.
 - **FileName:** caminho completo do nome do arquivo onde os logs serão armazenados.

Most common uses

- **WebSockets API**
 - [Como Conectar à WebSocket API](#)
 - [Como Assinar um Canal WebSocket](#)
- **REST API**
 - [Como obter dados de mercado](#)
 - [Como utilizar a API REST privada](#)
 - [Como negociar Futures](#)
 - [Private Requests Time](#)

WebSocket Feed

Para assinar as mensagens de um canal a partir de um determinado servidor, o lado do cliente deve enviar uma mensagem de assinatura ao servidor.

Se a assinatura for bem-sucedida, o sistema enviará mensagens de ack a você, quando a resposta for definida como true.

```
{
  "id":"1545910660739",
  "type":"ack"
}
```

Enquanto houver mensagens de tópico geradas, o sistema enviará as mensagens correspondentes ao lado do cliente.

Os seguintes métodos de Assinatura / Cancelamento de assinatura são suportados.

Canais Públicos

Method	Parâmetros	Descrição
SubscribeSymbolTickerV2	Symbol	Assine este tópico para obter o push em tempo real das mudanças de BBO. Após a assinatura, quando houver mudanças no order book, o sistema enviará as informações do ticker symbol em tempo real para você. Recomenda-se usar o novo tópico para informações pontuais.
SubscribeSymbolTicker	Symbol	Faça subscribe a este tópico para obter o push em tempo real de mudanças de BBO. O canal ticker fornece atualizações de preço em tempo real sempre que ocorre um match. Se várias ordens forem correspondidas ao mesmo tempo, apenas o último evento de correspondência será enviado.
SubscribeLevel2MarketData	Symbol	Assine este tópico para obter dados de order book de Level 2.
SubscribeExecutionData	Symbol	Para cada ordem executada, o sistema enviará a você as mensagens de match no formato a seguir.
SubscribeLevel2_5BestAskBid	Symbol	Retornado a cada 100 milissegundos no máximo.
SubscribeLevel2_50BestAskBid	Symbol	Retornado a cada 100 milissegundos no máximo.
SubscribeContractMarketData	Symbol	Assine este tópico para obter os dados de mercado do contrato.
SubscribeSystemAnnouncements	Symbol	Assine este tópico para obter os anúncios do sistema.
SubscribeTransactionStatistics	Symbol	As estatísticas de transações serão enviadas aos usuários a cada 5 segundos.
SubscribeKlines	Symbol	Assinar dados de klines (candlestick) de contratos.
SubscribeFundingFeeSettlement	Symbol	Assine notificações de liquidação de funding fee.

Se o parâmetro ACK for enviado como true, após uma assinatura / cancelamento de assinatura bem-sucedido, o cliente recebe uma mensagem sobre isso.

Private Channels

Requer uma ApiKey válida obtida da sua conta Kucoin. A ApiKey, o ApiSecret e a Passphrase devem ser definidos na propriedade Kucoin do componente de API cliente.

Os dados a seguir são enviados ao cliente toda vez que há uma alteração. Não há necessidade de assinar nenhum método; isso é feito automaticamente se você definir uma ApiKey válida.

Method	Descrição
SubscribeTradeOrders	Este tópico enviará todos os eventos de alteração das suas ordens.
SubscribeAccountBalance	Você receberá esta mensagem quando o saldo de uma conta for alterado. A mensagem contém os detalhes da alteração.
SubscribePositionChange	O sistema enviará o evento de mudança quando o status da posição mudar.

SubscribeStopOrder	Quando uma stop order é recebida pelo sistema, você receberá uma mensagem com o tipo "open". Isso significa que esta ordem entrou no sistema e aguardou para ser acionada.
SubscribeMarginMode	Inscrite-se em mudanças de modo de margem. O sistema enviará o evento de mudança quando o modo de margem for atualizado.
SubscribeCrossMargin-Leverage	Faça subscribe a alterações de alavancagem de margem cruzada. O sistema enviará o evento de alteração quando a alavancagem de margem cruzada for atualizada.

REST API

Todos os endpoints retornam um objeto ou array JSON.

EndPoints da API Pública

Esses endpoints podem ser acessados sem nenhuma autorização.

General EndPoints

Method	Parâmetros	Descrição
GetServiceStatus		Testa a conectividade com a Rest API e obtém o Status do Serviço
GetServerTime		Testar a conectividade com a Rest API e obter a hora atual do servidor.

Market Data EndPoints

Method	Parâmetros	Descrição
GetOpenContractList		Envia uma requisição para obter as informações de todos os contratos abertos.
GetOrderInfoContract		Envia uma requisição para obter informações do contrato especificado.
GetTicker	Symbol	O ticker em tempo real inclui o último preço negociado, o último tamanho negociado, o ID da transação, o lado do tomador de liquidez, o melhor preço e tamanho de bid, o melhor preço e tamanho de ask, bem como o horário da transação das ordens. Essas mensagens também podem ser obtidas através de Websocket. O Sequence Number é usado para julgar se as mensagens enviadas pelo Websocket são contínuas.
GetPartOrderBook20	Symbol	Obtém um snapshot das ordens em aberto agregadas para um símbolo.
GetPartOrderBook100	Symbol	Obtém um snapshot das ordens em aberto agregadas para um símbolo.
GetFullOrderBook	Symbol	Obtém um snapshot das ordens em aberto agregadas para um símbolo.

GetLevel2PullingMessages	Symbol	Se as mensagens enviadas pelo Websocket não forem contínuas, você pode enviar a seguinte requisição e re-extrair os dados para garantir que a sequência não esteja faltando. Na requisição, o parâmetro start é o número de sequência da sua última mensagem recebida mais 1, e o parâmetro end é o número de sequência da sua mensagem recebida atual menos 1. Após re-extrair as mensagens e aplicá-las ao seu order book de exchange local, você pode continuar a atualizar o order book através do feed incremental do Websocket. Se a diferença entre os parâmetros end e start for maior que 500, pare de utilizar esta requisição e sugerimos que você reconstrua o orderbook Level 2.
GetTradeHistory	Symbol	Lista os últimos 100 trades de um símbolo.
GetInterestRateList	Symbol	Consulta a lista de taxas de juros.
GetIndexList	Symbol	Verificar lista de índices
GetCurrentMarkPrice	Symbol	Verificar o mark price atual.
GetPremiumIndex	Symbol	Submete uma requisição para obter o índice premium.
GetCurrentFundingRate	Symbol	Envia a requisição para verificar o mark price atual.
GetKLine	Symbol	Obtém os Dados de K Line do Contrato

Private API EndPoints

Requer uma APIKey e uma APISecret para ser autorizado pelo servidor.

User EndPoints

Method	Parâmetros	Descrição
GetAccountOverview		Get Account Overview
GetTransactionHistory		Se houver posições abertas, o status da primeira página retornada será Pending, indicando o lucro e a perda realizados no período de liquidação atual de 8 horas. Especifique o número de offset mínimo da página atual no campo offset para mudar de página.

Trade Endpoints

Method	Parâmetros	Descrição
PlaceOrder		Você pode colocar dois tipos de ordens: limit e market. As ordens só podem ser colocadas se sua conta tiver fundos suficientes. Uma vez que uma ordem é colocada, seus fundos ficarão retidos pela duração da ordem. A quantidade de fundos re-

		tidos depende do tipo de ordem e dos parâmetros especificados.
PlaceMarketOrder		Coloca uma Ordem a Mercado.
PlaceLimitOrder		Coloca uma Ordem Limit.
CancelOrder		Cancela uma Ordem pelo Order Id.
LimitOrderMassCancellation		Cancela todas as ordens abertas (excluindo stop orders). A resposta é uma lista de orderIDs das ordens canceladas.
StopOrderMassCancellation		Cancela todas as ordens stop não acionadas. A resposta é uma lista de orderIDs das ordens stop canceladas. Para cancelar ordens stop acionadas, use 'Limit Order Mass Cancellation'.
GetOrderList		Lista suas ordens atuais.
GetUntriggeredStopOrderList		Obter a lista de stop orders não acionadas.
GetListOrdersCompleted24hr		Obtém uma lista das últimas 1000 ordens nas últimas 24 horas. Se você precisar obter seu histórico recente de ordens negociadas com baixa latência, pode consultar este endpoint.
GetOrder		Obtém uma única ordem por order id (incluindo uma stop order).
GetOrderByClientOid		Obtém uma única ordem pelo client order id (incluindo uma ordem stop).
GetFills		Obtém uma lista de fills recentes.
GetRecentFills		Obtém uma lista dos 1000 fills mais recentes nas últimas 24 horas. Se você precisar obter seu histórico recente de ordens negociadas com baixa latência, pode consultar este endpoint.
ActiveOrderValueCalculation		Você pode consultar este endpoint para obter o número total e o valor de todas as suas ordens ativas.
GetPositionDetails		Obtém os detalhes da posição de uma posição especificada.
GetPositionList		Obtém os detalhes da posição de uma posição especificada.
AutoDepositMargin		Habilita/Desabilita o Auto-Deposit Margin
AddMarginManually		Adicionar Margem Manualmente
ObtainFuturesRiskLimitLevel		Esta interface pode ser usada para obter informações sobre o nível de limite de risco de um contrato específico
AdjustRiskLimitLevel		Esta interface é para o ajuste do nível de limite de risco. Ajustar o nível cancelará a ordem aberta, a resposta só pode indicar se o envio da requisição de ajuste foi bem-sucedido ou não.
GetFundingHistory		Envia uma requisição para obter o histórico de funding.
GetMaxOpenSize		Obter o tamanho máximo de posição aberta para um contrato.
SwitchMarginMode		Alterna entre os modos de margem cruzada e isolada.
GetMarginMode		Obter o modo de margem atual de um contrato.

Eventos

As mensagens da Kucoin são recebidas no componente `TsgcWebSocketClient`; você pode utilizar os seguintes eventos:

OnConnect

Após uma conexão bem-sucedida ao servidor Kucoin.

OnDisconnect

Após uma desconexão do servidor Kucoin

OnMessage

Mensagens enviadas pelo servidor ao cliente são tratadas neste evento.

OnError

Se houver algum erro no protocolo, este evento será chamado.

OnException

Se houver uma exceção não tratada, este evento será chamado.

Adicionalmente, há um evento específico no Componente da API Kucoin, chamado **OnKucoinHTTPException**, que é gerado toda vez que há um erro ao chamar uma Requisição HTTP (REST API ou WebSocket Feeds).

Kucoin | Futures Connect WebSocket API

Para se conectar à WebSocket API da Kucoin, basta criar um novo cliente API da Kucoin e anexá-lo ao TsgcWebSocketClient.

Veja abaixo um exemplo:

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();  
TsgcWSAPI_Kucoin_Futures  
*oKucoin = new  
TsgcWSAPI_Kucoin_Futures  
();  
oKucoin->Client = oClient;  
oClient->Active = true;
```


Kucoin | Assinar Canal WebSocket de Futuros

A Kucoin oferece uma variedade de canais nos quais você pode se inscrever para obter atualizações em tempo real de dados de mercado, ordens... Encontre abaixo um exemplo de como se inscrever em um Ticker:

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_Kucoin_Futures
*oKucoin = new
TsgcWSAPI_Kucoin_Futures
();
oKucoin->Client = oClient;
oKucoin->SubscribeSymbolTickerV2("XBTUSD");
void OnMessage(TsgcWSConnection *Connection, const string aText)
{
// here you will receive the ticker updates
}
```

Kucoin | Futures Get Market Data

A Kucoin oferece Market Data públicos através de endpoints REST; quando você chama um desses endpoints, obtém um snapshot dos dados de mercado solicitados.

Os Market Data Endpoints não exigem autenticação, então estão disponíveis livremente para todos os usuários.

Exemplo: para obter o snapshot do ticker BTC-USDT, faça a seguinte chamada

```
TsgcWSAPI_Kucoin_Futures
*oKucoin = new
TsgcWSAPI_Kucoin_Futures
(this);
ShowMessage(oKucoin->REST_API->GetTicker("
XBTUSDM
"));
```

Kucoin | Futures Private REST API

A REST API da Kucoin oferece endpoints públicos e privados. Os endpoints Privados exigem que as mensagens sejam assinadas para aumentar a segurança das transações.

Primeiro você deve fazer login na sua conta Kucoin e criar uma nova API, você obterá os seguintes valores:

- ApiKey
- ApiSecret
- Passphrase

Esses campos devem ser configurados na propriedade Kucoin do componente cliente da API Kucoin.

Uma vez configurado, você pode começar a fazer requisições privadas à REST API do Kucoin Pro

*As Private Requests exigem que sua máquina local tenha o horário local sincronizado, caso contrário, as requisições serão rejeitadas pelo servidor Kucoin. Confira o artigo a seguir sobre isso, [Kucoin Private Requests Time](#).

```
TsgcWSAPI_Kucoin_Futures  
  
*oKucoin = new  
TsgcWSAPI_Kucoin_Futures  
(this);  
oKucoin->Kucoin->ApiKey = "<your api key>";  
oKucoin->Kucoin->ApiSecret = "<your api secret>";  
oKucoin->Kucoin->Passphrase = "<your passphrase>";  
ShowMessage(oKucoin->REST_API->GetAccountOverview());
```

Kucoin | Negociação de Futuros

A Kucoin permite que você negocie Futures usando sua REST API.

Configuração

Primeiro você deve criar uma **API Key** em sua conta Kucoin e adicionar privilégios para negociar com Futures. Uma vez feito isto, você pode começar a negociar futures.

Primeiro, **defina sua ApiKey, ApiSecret e Passphrase** no componente Kucoin Client, isto será utilizado para assinar as requisições enviadas ao servidor Kucoin.

Place an Order

Para colocar uma nova ordem, basta chamar o método **REST_API.PlaceOrder** do Componente Cliente Kucoin.

Dependendo do tipo da ordem (market, limit...), a API requer mais ou menos campos.

Parâmetros

Pa-ram	ty-pe	Descrição
clientOid	String	Id de ordem único criado pelos usuários para identificar suas ordens, por exemplo UUID. Permite apenas números, caracteres, underline(_) e separador(-)
side	String	buy ou sell
symbol	String	um código de contrato válido. por exemplo XBTUSDM
type	String	[opcional] limit ou market
leverage	String	Alavancagem da ordem
remark	String	[opcional] observação para a ordem, o comprimento não pode exceder 100 caracteres utf8
stop	String	[opcional] down ou up . Requer que stopPrice e stopPriceType estejam definidos
stopPriceType	String	[opcional] Seja TP , IP ou MP , precisa ser definido se stop for especificado.
stopPrice	String	[opcional] Precisa ser definido se stop for especificado.
reduceOnly	boolean	[opcional] Uma marca para reduzir somente o tamanho da posição. Definido como false por padrão. É necessário definir o tamanho da posição quando reduceOnly é true.
closeOrder	boolean	[opcional] Uma marca para fechar a posição. Definida como false por padrão. Fechará todas as posições quando closeOrder for true.
forceHold	boolean	[opcional] Uma marca para reter os fundos de uma ordem à força, mesmo sendo uma ordem para reduzir o tamanho da posição. Isso ajuda a ordem a permanecer no order book e a não ser cancelada quando o tamanho da posição muda. Definida como false por padrão.

PARÂMETROS DA ORDEM LIMIT

Param	type	Descrição
price	String	Limit price
size	Integer	Tamanho da ordem. Deve ser um número positivo
timeIn-Force	String	[opcional] GTC, IOC (o padrão é GTC), leia Time In Force
postOnly	boolean	[opcional] Flag Post only, inválida quando timeInForce é IOC . Quando postOnly é escolhido, não é permitido escolher hidden ou iceberg.
hidden	boolean	[opcional] Ordens não exibidas no order book. Quando hidden é escolhido, não é permitido escolher postOnly.
iceberg	boolean	[opcional] Apenas a parte visível da ordem é exibida no order book. Quando iceberg é escolhido, não é permitido escolher postOnly.
visibleSize	Integer	[opcional] O tamanho máximo visível de uma ordem iceberg

MARKET ORDER PARAMETERS

Param	type	Descrição
size	Integer	[opcional] quantidade de contrato a comprar ou vender

Ao enviar uma ordem, há 2 possibilidades:

1. **Successful:** a função PlaceOrder retorna a mensagem enviada pelo servidor Kucoin.
2. **Error:** a exceção é retornada no evento OnKucointHTTPException.

Place Market Order 1 XBTUSD

```

TsgcWSAPI_Kucoin_Futures
*oKucoin = new
TsgcWSAPI_Kucoin_Futures
(this);
oKucoin->Kucoin->ApiKey = "<api key>";
oKucoin->Kucoin->ApiSecret = "<api secret>";
oKucoin->Kucoin->Passphrase = "<passphrase>";
ShowMessage(oKucoin->REST_API->
PlaceMarketOrder
(
kosBuy, "XBTUSD", 1
));

```

Place Limit Order 1 XBTUSD at 40000

```
TsgcWSAPI_Kucoin_Futures

*oKucoin = new

TsgcWSAPI_Kucoin_Futures

(this);
oKucoin->Kucoin->ApiKey = "<api key>";
oKucoin->Kucoin->ApiSecret = "<api secret>";
oKucoin->Kucoin->Passphrase = "<passphrase>";
ShowMessage(oKucoin->REST_API->

PlaceLimitOrder

(

kosBuy, "XBTUSD", 1, 40000

));
```

Kucoin | Futures Private Requests Time

Quando você faz uma requisição privada à Kucoin, a mensagem é assinada para aumentar a segurança das requisições. A mensagem usa a hora local e a envia dentro da mensagem assinada; se a hora local tiver uma diferença maior que 5 segundos em relação aos servidores da Kucoin, a requisição será rejeitada. Portanto, é importante verificar se sua hora local está sincronizada, o que você pode fazer usando o método de sincronização de hora do seu SO.

Você pode verificar o horário do servidor Kucoin chamando o método **GetServerTime**, que retornará o horário do servidor Kucoin

API 3Commas

3Commas

APIs suportadas

- **WebSockets API:** conecta a um servidor websocket público e fornece atualizações de dados de mercado em tempo real.
- **REST API:** A REST API possui endpoints para gerenciamento de contas e ordens, bem como dados públicos de mercado.

WebSockets API

O feed websocket fornece atualizações de dados de mercado em tempo real para Trades e Deals

Você pode assinar os seguintes **canais públicos**:

Method	Arguments	Descrição
SubscribeSmartTrades		
SubscribeDeals		

Esses canais exigem **Autenticação** nos servidores 3Commas. Portanto, primeiro solicite suas chaves de API na sua Conta 3Commas e, em seguida, defina os valores na propriedade ThreeComas do componente:

- ApiKey
- ApiSecret

Se a subscrição for bem-sucedida, o evento **OnThreeCommasConfirmSubscription** será chamado. Se não, o evento **OnThreeCommasRejectSubscription** é chamado; você pode obter o motivo da rejeição usando o parâmetro **aRawMessage**.

REST API

Test Connectivity

Method	Arguments	Descrição
GetPing		
GetServerTime		Retorna o horário do servidor

Account

Method	Arguments	Descrição
GetAccounts		Lista de exchanges conectadas pelo usuário
GetMarketList		Lista de mercados suportados
GetMarketPairs	aMarketCode: código do mercado	Todos os pares de mercado
GetCurrencyRatesWithLeverageData	aMarketCode: código do mercado aPair: nome do par	Taxas e limites de moedas com dados de alavancagem
GetCurrencyRates	aMarketCode: código do mercado aPair: nome do par	Taxas e limites de moeda
GetBalances	aAccountId: id da conta	Faz balanceamento de carga para a exchange especificada
GetAccountTableData	aAccountId: id da conta	Informações sobre todos os saldos do usuário em um exchange especificado
GetAccountLeverage	aAccountId: id da conta aPair: nome do par	Informações sobre a alavancagem da conta
GetAccountInfo	aAccountId: id da conta	Single Account Info

Smart Trades

Method	Arguments	Descrição
GetSmartTradeHistory		Obter o Histórico de Negociações
PlaceMarketOrder	aAccountId: id da conta aOrderSide: buy ou sell aPair: nome do par aQuantity: quantidade	Coloca uma Ordem de Mercado
PlaceLimitOrder	aAccountId: id da conta aOrderSide: buy ou sell aPair: nome do par aQuantity: quantidade aPrice: preço limit	Coloca uma Ordem Limit
GetSmartTrade	aid: id da negociação	Obtém um Smart Trade pelo Id da Negociação

CancelSmartTrade	ald: id da negociação	Cancela um Smart Trade pelo Id do Trade
CloseByMarketSmartTrade	ald: id da negociação	
EditSmartTrade	ald: id da negociação	Edita um Smart Trade existente
ForceStartSmartTrade	ald: id da negociação	Forçar o início de um Smart Trade
AddFundsSmartTrade	ald: id da negociação	Adicionar fundos a um Smart Trade
GetSmartTradeTrades	ald: id da negociação	Obtém as negociações (trades) de um Smart Trade

DCA Bot

Method	Arguments	Descrição
CreateDCABot		Criar um novo DCA Bot
GetDCABot	ald: id do bot	Obter um DCA Bot por Id
GetDCABots		Obter todos os DCA Bots
EnableDCABot	ald: id do bot	Enable a DCA Bot
DisableDCABot	ald: id do bot	Disable a DCA Bot
DeleteDCABot	ald: id do bot	Excluir um DCA Bot
CancelDCABot	ald: id do bot	Cancelar um DCA Bot
GetDCABotStats		Get DCA Bot statistics
GetAvailableStrategyList		Obtém a lista de estratégias disponíveis
GetBlacklistPairs		Obter pares de blacklist
AddBlacklistPairs		Adicionar pares à blacklist

Deals

Method	Arguments	Descrição
GetDeals		Get all deals
GetDeal	aid: id do negócio (deal)	Obtém um deal por Id
Update-Deal	aid: id do negócio (deal)	Atualizar um deal
CancelDeal	aid: id do negócio (deal)	Cancelar um deal
CloseAt-Market-Deal	aid: id do negócio (deal)	Fechar um deal a preço de mercado

Grid Bot

Method	Arguments	Descrição
Create-GridBot		Cria um novo Grid Bot
GetGrid-Bot	aid: id do bot	Get a Grid Bot by Id
GetGrid-Bots		Obter todos os Grid Bots
Enable-GridBot	aid: id do bot	Habilitar um Grid Bot
Disable-GridBot	aid: id do bot	Desabilitar um Grid Bot
Delete-GridBot	aid: id do bot	Delete a Grid Bot

Eventos

OnConnect

Quando uma nova conexão WebSocket é aberta

OnDisconnect

Quando uma conexão WebSocket é fechada

OnThreeCommasConnect

Quando o cliente recebe uma mensagem Welcome do servidor 3Commas, significa que a conexão está pronta.

OnThreeCommasConfirmSubscription

Confirma uma assinatura anterior enviada pelo cliente.

OnThreeCommasRejectSubscription

Há um erro ao tentar assinar um canal do 3Commas

OnThreeCommasMessage

Aqui o cliente recebe os dados enviados pelo servidor relacionados aos canais assinados

OnThreeCommasPing

Ping enviado pelo servidor ao cliente.

OnThreeCommasHTTPException

Se houver qualquer erro ao chamar métodos REST HTTP, este evento será chamado.

API OKX

OKX

APIs suportadas

- **WebSockets API:** conecta a um servidor websocket e fornece atualizações de dados de mercado em tempo real, alterações de conta e colocação de ordens de negociação.

Properties

Os canais WebSocket são divididos em duas categorias: canais públicos e privados.

- **Canais públicos:** incluem o canal de tickers, o canal K-Line, o canal de preço limite, o canal do livro de ordens e o canal de mark price, etc. — não exigem login.
- **Private channels:** incluindo account channel, order channel e position channel, etc -- requerem login.

Você pode configurar as seguintes propriedades na propriedade OKX.

- **ApiKey:** você pode solicitar uma nova api key na sua conta OKX, basta copiar o valor para esta propriedade.
- **ApiSecret:** é o valor secreto da api.
- **Passphrase:** é a string personalizada definida ao criar uma nova api key.
- **IsDemo:** se habilitado, conectará à conta Demo da OKX (desabilitado por padrão).
- **IsPrivate:** se habilitado, você poderá conectar a canais privados (desabilitado por padrão).
- **IsBusiness:** se habilitado, conectará ao endpoint Business WebSocket (wss://ws.okx.com:8443/ws/v5/business). Utilize isto para canais de candle, canais de algo order e outros canais avançados (desabilitado por padrão).

Conexão

Quando o cliente conecta com sucesso aos servidores OKX, o evento **OnOKXConnect** é disparado. Se houver algum erro ao tentar conectar, o evento **OnOKXError** será disparado com os detalhes do erro.

Depois que o evento **OnOKXConnect** é disparado, então você pode começar a **enviar e receber mensagens** dos servidores OKX.

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_OKX *oOKX = new TsgcWSAPI_OKX(NULL);
oOKX->Client = oClient;
oOKX->OKX->ApiKey = "alsdj23kandfnasbdfdkjhdsf";
oOKX->OKX->ApiSecret = "aldskjfk3jkadknfajndsxfj23j";
oOKX->OKX->Passphrase = "secret_passphrase";
oClient->Active = true;
void OnOKXConnect(TObject *Sender, string aMessage, string aCode, string aRawMessage)
{
    DoLog("#OKX Connected");
}
void OnOKXError(TObject *Sender, string aCode, string aMessage, string aRawMessage)
{
    DoLog("#error: " + aMessage);
}
```

Canais Públicos

O feed do websocket fornece atualizações de dados de mercado em tempo real para ordens e negociações. O feed do websocket tem alguns canais públicos como ticker, trades...

Você pode assinar os seguintes **canais públicos**:

Method	Descrição
SubscribeInstruments	A lista completa de instrumentos será enviada pela primeira vez após a assinatura. Posteriormente, os instrumentos serão enviados se houver qualquer mudança no estado do instrumento (como entrega de FUTURES, exercício de OPTION, listagem de novos contratos / pares de negociação, suspensão de negociação, etc.).
SubscribeTicker	Recupera o último preço negociado, preço de bid, preço de ask e volume de negociação de 24 horas dos instrumentos. Os dados serão enviados a cada 100 ms.
SubscribeOpenInterest	Recupera o open interest. Os dados serão enviados a cada 3 segundos.
SubscribeCandlestick	Recupera os dados de candlesticks de um instrumento. A frequência de push é o intervalo mais rápido, enviando os dados a cada 500ms.
SubscribeTrades	Recupera os dados de trades recentes. Os dados serão enviados sempre que houver um trade.
SubscribeEstimatedPrices	Recupera o preço estimado de entrega/exercício de contratos FUTURES e OPTION. Apenas o preço estimado de entrega/exercício será enviado uma hora antes da entrega/exercício, e será enviado se houver qualquer mudança de preço.
SubscribeMarkPrice	Recupera o mark price. Os dados serão enviados a cada 200 ms quando o mark price mudar, e serão enviados a cada 10 segundos quando o mark price não mudar.
SubscribeMarkPrice-Candlestick	Recupera os dados de candlesticks do mark price. Os dados serão enviados a cada 500 ms.
SubscribePriceLimit	Recupera o preço máximo de compra e o preço mínimo de venda do instrumento. Os dados serão enviados a cada 5 segundos quando houver alterações nos limites, e não serão enviados quando não houver alterações no limite.
SubscribeOrderBook	Recupera dados do livro de ofertas. Use books para 400 níveis de profundidade, book5 para 5 níveis de profundidade, bbo-tbt tick a tick com 1 nível de profundidade, books50-l2-tbt tick a tick com 50 níveis de profundidade, e books-l2-tbt para tick a tick com 400 níveis de profundidade. • books: 400 níveis de profundidade serão enviados no snapshot completo inicial. Dados incrementais serão enviados a cada 100 ms quando houver mudança no order book. • books5: 5 níveis de profundidade serão enviados a cada vez. Os dados serão enviados a cada 100 ms quando houver mudança no order book. • bbo-tbt: 1 nível de profundidade será enviado a cada vez. Os dados serão enviados a cada 10 ms quando houver mudança no order book. • books-l2-tbt: 400 níveis de profundidade serão enviados no snapshot completo inicial. Dados incrementais serão enviados a cada 10 ms quando houver mudança no order book. • books50-l2-tbt: 50 níveis de profundidade serão enviados no snapshot completo inicial. Dados incrementais serão enviados a cada 10 ms quando houver mudança no order book. Se asks ou bids for um array vazio, significa que há mudanças na profundidade de 400, em vez de 50. Se você mantém os dados do order book localmente, ignore asks e bids vazios.
SubscribeOptionSummary	Recupera informações detalhadas de preços de todos os contratos OPTION. Os dados serão enviados (push) de uma só vez.
SubscribeFundingRate	Recupera a taxa de funding. Os dados serão enviados em 30s a 90s.
SubscribeIndexCandlestick	Recuperar os dados de candlesticks do índice. Os dados serão enviados a cada 500 ms.
SubscribeIndexTicker	Recupera os dados de tickers de índice
SubscribeStatus	Obtém o status da manutenção do sistema e envia quando o status da manutenção do sistema muda. Primeira assinatura: "Envia os dados de alteração

	mais recentes"; toda vez que há uma mudança de estado, envia o conteúdo alterado
SubscribePublicStructureBlockTrades	Os dados serão enviados sempre que houver uma negociação em bloco (block trade).
SubscribeBlockTickers	Recupera o volume de negociação em bloco mais recente nas últimas 24 horas. Os dados serão enviados quando acionados por um evento de execução de transação. Além disso, também serão enviados em intervalos de 5 minutos de acordo com a granularidade da assinatura.
SubscribeAllTrades	Recuperar todos os dados de negociações. Os dados serão enviados sempre que houver uma negociação.
SubscribeLiquidationOrders	Recuperar ordens de liquidação. Os dados serão enviados quando houver uma ordem de liquidação.
SubscribeADLWarning	Recupera dados de aviso de ADL. Os dados serão enviados (push) quando o risco de auto-deleveraging aumentar.
SubscribeEconomicCalendar	Recuperar os eventos do calendário econômico. Os dados serão enviados quando houver atualizações nos eventos econômicos.
SubscribePublicBlockTrades	Recuperar block trades públicos. Os dados serão enviados sempre que houver um block trade.
SubscribeOptionTrades	Recupera dados de negociações de opções. Os dados serão enviados sempre que houver uma negociação de opção.
SubscribeCallAuctionDetails	Recupera detalhes do leilão de chamada. Os dados serão enviados quando houver atualizações nas informações do leilão de chamada.

```

TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_OKX *oOKX = new TsgcWSAPI_OKX(NULL);
oOKX->Client = oClient;
oOKX->OKX->ApiKey = "alsdjk23kandfnasbdfdkjhdsf";
oOKX->OKX->ApiSecret = "aldskjfk3jkadknfajndsxfj23j";
oOKX->OKX->Passphrase = "secret_passphrase";
oClient->Active = true;
void OnOKXConnect(TObject *Sender, string aMessage, string aCode, string aRawMessage)
{
    oOKX->SubscribeInstruments(okxitFutures);
}

```

Private Channels

Incluindo account channel, order channel e position channel, etc -- requerem login.

Você pode se inscrever nos seguintes **canais Privados**:

Method	Descrição
SubscribeAccount	Recupera informações da conta. Os dados serão enviados quando acionados por eventos como colocação de ordem, cancelamento de ordem, execução de transação etc. Também serão enviados em intervalos regulares de acordo com a granularidade da inscrição.
SubscribePositions	Recupera informações de posição. O snapshot inicial será enviado de acordo com a granularidade da assinatura. Os dados serão enviados quando disparados por eventos como colocação/cancelamento de ordem, e também serão enviados em intervalo regular de acordo com a granularidade da assinatura.
SubscribeBalanceAndPosition	Recupera informações de saldo da conta e de posição. Os dados serão enviados quando disparados por eventos como ordem preenchida, transferência de funding.

SubscribeOrders	Recupera informações da ordem. Os dados não serão enviados na primeira assinatura. Os dados só serão enviados quando acionados por eventos como colocação/cancelamento de ordem.
SubscribeOrdersAlgo	Recupera ordens algo (inclui trigger order, oco order, conditional order). Os dados não serão enviados quando inscritos pela primeira vez. Os dados só serão enviados quando acionados por eventos como colocação/cancelamento de ordem.
SubscribeAdvanceAlgo	Recupera ordens algo avançadas (incluindo ordem Iceberg, ordem TWAP, ordem Trailing). Os dados serão enviados quando a inscrição for feita pela primeira vez. Os dados serão enviados quando acionados por eventos como a colocação/cancelamento de ordem.
SubscribePositionRisk	Este canal de push é utilizado apenas como aviso de risco e não é recomendado como julgamento de risco para negociação estratégica No caso de o mercado não estar se movendo violentamente, pode haver a possibilidade de a posição ter sido liquidada no mesmo momento em que esta mensagem é enviada.
SubscribeAccountGreeks	Recupera informações de greeks da conta. Os dados serão enviados quando acionados por eventos como aumento/diminuição de posições ou saldo de caixa na conta, e também serão enviados em intervalos regulares de acordo com a granularidade da assinatura.
SubscribeRfqs	Recupera os Rfqs.
SubscribeQuotes	Recupera as Quotes.
SubscribePrivateStructureBlockTrades	Recupera Block Trades de Estrutura.
SubscribeSpotGridAlgoOrders	Recupera ordens de spot grid algo. Os dados serão enviados na primeira assinatura. Os dados serão enviados quando acionados por eventos como registro/cancelamento de ordem.
SubscribeContactGridAlgoOrders	Recupera ordens de grid de contrato. Os dados serão enviados quando inscritos pela primeira vez. Os dados serão enviados quando acionados por eventos como colocação/cancelamento de ordem.
SubscribeGridPositions	Recupera posições de grid. Os dados serão enviados quando inscritos pela primeira vez. Os dados serão enviados quando acionados por eventos como colocação/cancelamento de ordem.
SubscribeGridSubOrders	Recupera sub-ordens de grid. Os dados serão enviados quando inscritos pela primeira vez. Os dados serão enviados quando acionados por eventos como colocação de ordem.
SubscribeFills	Recupera dados de ordens preenchidas. Os dados serão enviados quando uma ordem for preenchida.
SubscribeDepositInfo	Recupera informações de depósito. Os dados serão enviados quando houver uma atualização de status de depósito.
SubscribeWithdrawalInfo	Recupera informações de saque. Os dados serão enviados quando houver uma atualização de status de saque.

```

TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_OKX *oOKX = new TsgcWSAPI_OKX(NULL);
oOKX->Client = oClient;
oOKX->OKX->ApiKey = "alsdk23kandfnasbdfdkjhdsdf";
oOKX->OKX->ApiSecret = "alsdkjfk3jkadknfajndsxfj23j";
oOKX->OKX->Passphrase = "secret_passphrase";
oClient->Active = true;
void OnOKXConnect(TObject *Sender, string aMessage, string aCode, string aRawMessage)
{
    oOKX->SubscribeOrders(okxItFutures, "BTC-USD", "BTC-USD-200329");
}

```


Trading

O WebSocket Trade requer **Authentication**.

Você só pode fazer uma ordem se tiver fundos suficientes. Veja abaixo uma tabela com os parâmetros da requisição:

Parameter	Type	Obrigatório	Descrição
id	String	Sim	Identificador único da mensagem Fornecido pelo cliente. Será retornado na mensagem de resposta para identificar a requisição correspondente. Uma combinação de caracteres alfanuméricos sensíveis a maiúsculas/minúsculas, todos números ou todas letras, com até 32 caracteres.
> instId	String	Sim	Instrument ID, por exemplo <code>BTC-USD-190927-5000-C</code>
> tdMode	String	Sim	Modo de negociação Modo de margem <code>isolated</code> <code>cross</code> Modo sem margem <code>cash</code>
> ccy	String	No	Moeda da margem Aplicável somente a <code>cross</code> <code>MARGIN</code> ordens em <code>Single-currency margin</code> .
> clOrdId	String	No	ID de ordem fornecido pelo cliente Uma combinação de caracteres alfanuméricos sensíveis a maiúsculas e minúsculas, todos números ou todas letras, de até 32 caracteres.
> tag	String	No	Tag da ordem Uma combinação de caracteres alfanuméricos sensíveis a maiúsculas e minúsculas, todos números ou todas letras, de até 16 caracteres.
> side	String	Sim	Lado da ordem, <code>buy</code> <code>sell</code>
> posSide	String	Optional	Lado da posição O padrão é <code>net</code> no <code>net</code> modo É obrigatório no modo <code>long/short</code> e só pode ser <code>long</code> ou <code>short</code> . Aplicável apenas a <code>FUTURES/SWAP</code> .
> ordType	String	Sim	Tipo de ordem <code>market</code> : ordem a mercado <code>limit</code> : ordem limitada <code>post_only</code> : ordem Post-only <code>fok</code> : ordem Fill-or-kill <code>ioc</code> : ordem Immediate-or-cancel <code>optimal_limit_ioc</code> : ordem a mercado com immediate-or-cancel
> sz	String	Sim	Quantidade a comprar ou vender.
> px	String	Optional	Price Aplicável apenas a ordens <code>limit</code> , <code>post_only</code> , <code>fok</code> , <code>ioc</code> .
> reduce-Only	Boolean	No	Se deve reduzir somente a posição ou não, <code>true</code> <code>false</code> , o padrão é <code>false</code> . Aplicável somente a ordens <code>MARGIN</code> , e ordens <code>FUTURES/SWAP</code> em modo <code>net</code> Aplicável somente a <code>Single-currency margin</code> e <code>Multi-currency margin</code>
> tgtCcy	String	No	Tipo de quantidade <code>base_ccy</code> : Moeda base, <code>quote_ccy</code> : Moeda de cotação Aplicável somente a <code>SPOT</code> negociado com ordem a Mercado O padrão é <code>quote_ccy</code> para compra, <code>base_ccy</code> para venda

Parameter	Type	Obrigatório	Descrição
> banAmend	Boolean	No	Se deve banir a modificação de ordens spot ou não, true ou false, o padrão é false. Falhará ao colocar ordens se o saldo não for suficiente quando banAmend for true. Aplicável apenas a ordens SPOT market

Exemplo de Place Order

Você só pode colocar uma ordem se tiver fundos suficientes.

```
// Place Market Order
TsgcWSAPI_OKX1->PlaceMarketOrder(okxosBuy, "ETH-BTC", 1);
// Place Limit Order
TsgcWSAPI_OKX1->PlaceLimitOrder(okxosBuy, "ETH-BTC", 1, 0.25);
```

Exemplo de Cancelamento de Ordem

Cancelar uma ordem incompleta

```
TsgcWSAPI_OKX1->CancelOrder("
ETH-BTC
", "457589362405027840");
```

Amend Order

Modifica uma ordem incompleta.

```
TsgcWSAPI_OKX1->AmendOrder("ETH-BTC", "457589362405027840", "", 2);
```

Operações de trade em batch

A WebSocket Trade API também suporta operações em lote para colocar, cancelar e modificar múltiplas ordens de uma vez. Essas operações exigem **Authentication**.

Method	Descrição
BatchPlaceOrders	Coloca várias ordens em uma única requisição. No máximo 20 ordens podem ser colocadas de uma vez.
BatchCancelOrders	Cancela múltiplas ordens em uma única requisição. No máximo 20 ordens podem ser canceladas por vez.
BatchAmendOrders	Alterar múltiplas ordens incompletas em uma única requisição. No máximo 20 ordens podem ser alteradas por vez.
MassCancelOrders	Cancela em massa todas as ordens pendentes para um tipo de instrumento específico.

API XTB

XTB

APIs suportadas

- **WebSockets API:** conecta a um servidor websocket e fornece atualizações de dados de mercado em tempo real, mudanças de conta e colocação de ordens de trading.

Properties

O protocolo WebSocket permite 2 tipos de requisições: **Streaming commands** (receber atualizações ao vivo) e **Retrieve Trading Data** (enviar uma requisição ao servidor recuperando alguma informação).

Você pode configurar as seguintes propriedades na propriedade XTB.

- **User:** o nome de usuário que identifica a conexão.
- **Password:** é o valor secreto do usuário.
- **Demo:** se habilitado, conectará à conta XTB Demo (desabilitado por padrão).

Conexão

Quando o cliente conecta com sucesso aos servidores XTB, o evento **OnXTBConnect** é disparado. Se houver qualquer erro ao tentar conectar, o evento **OnXTBError** será disparado com os detalhes do erro.

Após o evento **OnXTBConnect** ser disparado, você pode começar a **enviar e receber mensagens** dos servidores XTB.

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_XTB *oXTB = new TsgcWSAPI_XTB(NULL);
oXTB->Client = oClient;
oXTB->XTB->User = "user_0001";
oXTB->XTB->Password = "secret_0001";
oClient->Active = true;
void OnXTBConnect(TObject *Sender, string aStreamSessionId)
{
    DoLog("#XTB Connected");
}
void OnXTBError(TObject *Sender, string aCode, string aDescription, string aRawMessage)
{
    DoLog("#error: " + aDescription);
}
```

Comandos de Conexão

Method	Descrição
Login	Para realizar qualquer ação, a aplicação cliente precisa realizar o processo de login. Nenhuma funcionalidade está disponível antes do processo de login adequado. O método de login é chamado automaticamente após o cliente se conectar ao servidor websocket e os valores de User/Password serem definidos.
Logout	

Comandos de Streaming

Você pode se inscrever nos seguintes canais:

Method	Descrição
SubscribeBalance	Permite obter os valores atuais dos indicadores da conta em tempo real, assim que estiverem disponíveis no sistema.
SubscribeCandles	Assina e cancela a assinatura de candles de gráfico da API. O intervalo de cada candle é de 1 minuto. Um novo candle chega a cada minuto.
SubscribeKeepAlive	Assina e cancela a assinatura de mensagens 'keep alive'. Uma nova mensagem 'keep alive' é enviada pela API a cada 3 segundos.
SubscribeNews	Assina e cancela a assinatura de notícias.
SubscribeProfits	Faz subscribe e unsubscribe de profits.
SubscribeTickPrices	Estabelece a assinatura para cotações e permite que você obtenha as informações relevantes em tempo real, assim que estiverem disponíveis no sistema. O comando getTickPrices pode ser invocado muitas vezes para o mesmo símbolo, mas apenas uma assinatura para um determinado símbolo será criada. Esteja ciente de que, quando vários registros estão disponíveis, a ordem em que são recebidos não é garantida.
SubscribeTrades	Estabelece a assinatura para os dados de status de negociação do usuário e permite obter as informações relevantes em tempo real, assim que estiverem disponíveis no sistema. Esteja ciente de que, quando vários registros estão disponíveis, a ordem em que são recebidos não é garantida.
SubscribeTradeStatus	Permite obter o status das requisições de negociação enviadas em tempo real, assim que estiver disponível no sistema. Atenção: quando múltiplos registros estão disponíveis, a ordem em que são recebidos não é garantida.
SubscribePing	Chamar regularmente esta função é suficiente para atualizar o estado interno de todos os componentes do sistema. A conexão de streaming, quando nenhum comando é enviado pelo cliente na sessão, gera apenas tráfego de rede unidirecional. Recomenda-se que qualquer aplicação que não execute outros comandos chame este comando pelo menos uma vez a cada 10 minutos.

Recuperando Dados de Negociação

Você pode enviar as seguintes Requisições:

Method	Descrição
GetAllSymbols	Retorna um array de todos os símbolos disponíveis para o usuário.
GetCalendar	Retorna o calendário com eventos de mercado.
GetChartLastRequest	Observe que esta função geralmente pode ser substituída por seu equivalente de streaming getCandles, que é a forma preferida de recuperar dados de candle atuais. Retorna informações do gráfico, da data de início até o horário atual. Se o período escolhido de CHART_LAST_INFO_RECORD for maior que 1 minuto, o último candle retornado pela API pode mudar até o fim do período (o candle é atualizado automaticamente a cada minuto).

GetChartRangeRequest	Observe que esta função geralmente pode ser substituída por seu equivalente de streaming getCandles, que é a forma preferida de recuperar dados de velas atuais. Retorna informações de gráfico com dados entre as datas de início e fim fornecidas.
GetCommissionDef	Retorna o cálculo da comissão e da taxa de câmbio. O valor é calculado como valor esperado e, portanto, pode não ser perfeitamente preciso.
GetCurrentUserData	Retorna informações sobre a moeda da conta e a alavancagem da conta.
GetIbsHistory	Retorna dados de IBs do intervalo de tempo fornecido.
GetMarginLevel	Observe que esta função geralmente pode ser substituída pelo seu equivalente de streaming getBalance, que é a forma preferida de recuperar indicadores da conta. Retorna diversos indicadores da conta
GetMarginTrade	Retorna a margem esperada para um determinado instrumento e volume. O valor é calculado como valor de margem esperado e, portanto, pode não ser perfeitamente exato.
GetNews	Observe que esta função geralmente pode ser substituída por seu equivalente de streaming getNews, que é a forma preferida de recuperar dados de notícias. Retorna notícias do servidor de negociação que foram enviadas dentro de um período de tempo especificado.
GetProfitCalculation	Calcula o lucro estimado para os dados de negociação fornecidos. Deve ser usado apenas para aplicativos do tipo calculadora. O lucro de transações abertas deve ser obtido do servidor, devido à maior precisão do cálculo do servidor
GetServerTime	Retorna o horário atual no servidor de negociação
GetStepRules	Retorna uma lista de step rules para DMAs
GetSymbol	Retorna informações sobre o símbolo disponível para o usuário
GetTickPrices	Observe que esta função pode normalmente ser substituída por seu equivalente de streaming getTickPrices, que é a forma preferida de recuperar dados de ticks. Retorna um array de cotações atuais para os símbolos fornecidos; apenas as cotações que mudaram a partir do timestamp fornecido são retornadas. O novo timestamp obtido da saída será utilizado como argumento da próxima chamada deste comando.
GetTradeRecords	Retorna o array de negociações listadas no argumento orders
GetTrades	Observe que esta função geralmente pode ser substituída por seu equivalente em streaming getTrades, que é a forma preferida de recuperar dados de negociações. Retorna um array das negociações do usuário.
GetTradesHistory	Observe que esta função geralmente pode ser substituída por seu equivalente de streaming getTrades, que é a forma preferida de recuperar dados de negociações. Retorna um array das negociações do usuário que foram fechadas dentro de um período de tempo especificado.
GetTradingHours	Retorna cotações e horários de negociação.
GetVersion	Retorna a versão atual da API.
Ping	Chamar esta função regularmente é suficiente para atualizar o estado interno de todos os componentes do sistema. É recomendado que qualquer aplicação que não execute outros comandos chame este comando pelo menos uma vez a cada 10 minutos. Observe que a contraparte de streaming desta função é a combinação de ping e getKeepAlive
TradeTransaction	Inicia a transação de negociação. tradeTransaction envia as informações principais da transação ao servidor.
TradeTransactionStatus	Observe que esta função pode normalmente ser substituída por seu equivalente de streaming getTradeStatus, que é a forma preferida de recuperar da-

dos de status de transação. Retorna o status atual da transação. A qualquer momento do processamento da transação, o cliente pode verificar o status da transação no lado do servidor. Para fazer isso, o cliente deve fornecer a ordem única obtida da invocação `tradeTransaction`.

API Bybit

Bybit

APIs suportadas

- **WebSocket API:** conecta a um servidor websocket e fornece atualizações de dados de mercado em tempo real, alterações de conta e muito mais.
- **REST API:** envia requisições HTTP para obter dados de mercado, registrar ordens, dados de conta...

Atualmente, a versão da API suportada é a V5. A API V5 traz uniformidade e eficiência às linhas de produtos da Bybit, unificando Spot, Derivativos e Opções em um único conjunto de especificações.

OpenAPI Version	Tipo de Conta	Linear			Inverse		Spot	Options
		USDT Perpetual	USDC Perpetual	USDC Futures	Perpetual	Futures		
V5	Unified trading account	✓	✓	✓	veja a nota		✓	✓
	Conta clássica	✓			✓	✓	✓	
V3	Unified trading account	✓	✓					✓
	Conta clássica	✓			✓	✓	✓	

*Nota: a conta Unified suporta negociação inversa. No entanto, a margem usada é da carteira de derivativos inversos em vez da carteira unified.

Properties

Você pode configurar as seguintes propriedades na propriedade Bybit.

- **ApiKey:** você pode solicitar uma nova api key em sua conta Bybit, basta copiar o valor para esta propriedade. Se a APIKey estiver definida, o cliente conectará ao servidor websocket privado. Se estiver vazia, conectará ao servidor WebSocket público.
- **ApiSecret:** é o valor secreto da api.
- **SignatureExpires:** número de segundos após o qual a assinatura expira (por padrão 10 segundos).
- **TestNet:** se habilitado, conectará à conta Demo da Bybit TestNet (desabilitado por padrão).

Conexão

Quando o cliente se conecta com sucesso aos servidores da Bybit, o evento **OnConnect** é disparado. Após o evento **OnConnect** ser disparado, você pode então começar a **enviar e receber mensagens** de/para os servidores da Bybit. Se você estiver se conectando ao canal websocket privado, deve aguardar até que o evento **OnBybitAuthentication** seja disparado e verificar se o parâmetro **success** é **true**, antes de assinar qualquer canal.

O cliente suporta várias APIs, portanto utilize a propriedade **BybitClient** para definir qual API você quer utilizar:

- **bybSpot**
- **byblInverse**
- **bybLinear**
- **bybPerpetual**

Veja abaixo um exemplo de conexão à WebSocket Spot Private API.

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bybit *oBybit = new TsgcWSAPI_Bybit(NULL);
oBybit->Client = oClient;
oBybit->Bybit->ApiKey = "a1sdjk23kandfnasbdfdkjhdsf";
oBybit->Bybit->ApiSecret = "aldskjfk3jkadknfajnds fj23j";
oBybit->BybitClient = bybSpot;
oClient->Active = true;
void OnConnect(TsgcWSConnection *Connection)
{
    DoLog("#Bybit Connected");
}
```

Após uma conexão bem-sucedida com o Spot WebSocket Server, você pode começar a assinar canais WebSocket; basta acessar a propriedade **SPOT** e então chamar qualquer um dos métodos **subscribe/unsubscribe** disponíveis.

Eventos

O cliente bybit implementa os seguintes eventos para controlar o fluxo da conexão e obter dados enviados pelo servidor WebSocket:

- **OnConnect:** disparado quando o cliente websocket conecta ao Servidor WebSocket.
- **OnDisconnect:** disparado quando o cliente websocket se desconecta do WebSocket Server.
- **OnBybitAuthentication:** disparado quando o cliente se autentica no servidor WebSocket privado.
- **OnBybitSubscribe:** quando o cliente assina um canal websocket.
- **OnBybitUnSubscribe:** quando o cliente cancela a assinatura de um canal websocket.
- **OnBybitData:** quando o cliente recebe dados do servidor.
- **OnBybitError:** quando há algum erro durante a conexão websocket da bybit.
- **OnBybitHTTPException:** quando há algum erro durante a requisição REST.

WebSocket API

O feed do websocket fornece atualizações de dados de mercado em tempo real para ordens e negociações. O feed do websocket tem alguns canais públicos como ticker, trades...

Você pode se inscrever nos seguintes **canais**:

Method	Público ou privado	Descrição
SubscribeOrderBook	Public	Assinar o stream do orderbook. Suporta diferentes profundidades.
SubscribeTrade	Public	Assine o stream de trades recentes.

SubscribeTicker	Public	Assinar o stream de ticker.
SubscribeKLine	Public	Assina o stream de klines.
SubscribeLiquidation	Public	Assine o stream de liquidação
SubscribeLT_KLine	Public	Assine o stream de kline do leveraged token.
SubscribeLT_Ticker	Public	Assina o stream de ticker de leveraged token.
SubscribeLT_Nav	Public	Assina o stream de ticker de leveraged token.
SubscribePosition	Private	Assine o stream de nav do leveraged token.
SubscribeExecution	Private	Assinar
SubscribeOrder	Private	Assinar
SubscribeWallet	Private	Assinar
SubscribeGreek	Private	Assinar
SubscribeDcp	Private	Assinar
SubscribeInsurance	Public	Assina o stream do insurance fund.
SubscribeOrderPriceLimit	Public	Assina o stream de order price limit.
SubscribeADLAlert	Public	Assina o stream de alerta de auto-desalavancagem.
SubscribeFastExecution	Private	Assina o stream de execução rápida.

Veja abaixo um exemplo de assinatura de canais websocket privados após uma autenticação bem-sucedida.

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bybit *oBybit = new TsgcWSAPI_Bybit(NULL);
oBybit->Client = oClient;
oBybit->Bybit->ApiKey = "alsdj23kandfnasbdfdkjhdsf";
oBybit->Bybit->ApiSecret = "aldskjfk3jkadknfajndsrfj23j";
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

A API REST tem uma lista de métodos públicos e privados para solicitar dados de: mercados, conta privada e carteira. Veja abaixo uma lista dos métodos disponíveis.

Method	Público / Privado
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	Private
PlaceMarketOrder	Private
PlaceLimitOrder	Private
AmendOrder	Private
CancelOrder	Private
GetOpenOrders	Private
CancelAllOrders	Private
GetOrderHistory	Private
GetPositionInfo	Private
SetLeverage	Private
SwitchCrossIsolatedMargin	Private
SetTPSLMode	Private
SwitchPositionMode	Private
SetRiskLimit	Private
SetTradingStop	Private
SetAutoAddMargin	Private
AddOrReduceMargin	Private
GetExecution	Private
GetClosedPNL	Private
ConfirmNewRiskLimit	Private
GetWalletBalance	Private
GetAccountInfo	Private
GetTransactionLog	Private
BatchPlaceOrder	Private
BatchAmendOrder	Private
BatchCancelOrder	Private
SetDCP	Private
GetFeeRate	Private
GetCollateralInfo	Private
SetMarginMode	Private
GetBorrowHistory	Private
GetCoinGreeks	Private
GetCoinInfo	Private
GetAllCoinsBalance	Private
CreateInternalTransfer	Private
GetInternalTransferList	Private
GetDepositRecords	Private
GetDepositAddress	Private
CreateWithdrawal	Private
CancelWithdrawal	Private
GetWithdrawalRecords	Private

Encontre abaixo um exemplo de como obter as ordens abertas.

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bybit *oBybit = new TsgcWSAPI_Bybit(NULL);
oBybit->Client = oClient;
oBybit->Bybit->ApiKey = "a1sdjk23kandfnasbdfdkjhdsdf";
oBybit->Bybit->ApiSecret = "a1dskjfk3jkadknfajndsrfj23j";
```

```
oBybit->BybitClient = bybSpot;  
oBybit->REST_API->GetAccountInfo();
```

API Forex

[Forex.com](#) (StoneX / CityIndex)

O Forex.com é um ambiente de negociação de varejo operado pela StoneX. O componente envolve a Trading REST API e o serviço de streaming baseado em Lightstreamer usado pelo Forex.com, StoneX e CityIndex (os três compartilham a mesma plataforma). As seguintes APIs são suportadas:

1. **Streaming API:** Lightstreamer TLCP 2.5 sobre WebSocket. Price ticks, atualizações de ordens, atualizações de posições, margem da conta e cotações são enviados em tempo real após um LogOn REST bem-sucedido.
2. **REST API:** login de sessão (UserName + Password + AppKey), gerenciamento de ordens trade e stop-limit, consultas de conta e dados de mercado. Utiliza HTTPS.

Componentes

- **TsgcWSAPI_Forex:** cliente principal. Mantém **Credentials**, expõe **REST_API**, o cliente de streaming **Lightstreamer** interno e os eventos **OnForex***.
- **TsgcHTTP_API_Forex:** componente REST, acessível através da propriedade **REST_API** do cliente principal.
- **TsgcWSPClient_Lightstreamer:** cliente interno do subprotocolo TLCP 2.5 usado para subscrições de streaming (PRICES / ORDERS / QUOTES / CLIENTACCOUNTMARGIN / TRADEMARGIN).

Published Properties

- **Credentials:** grupo de campos necessários para fazer login na REST API.
 - **UserName:** seu nome de usuário da conta Forex.com / StoneX / CityIndex.
 - **Password:** senha da conta.
 - **AppKey:** chave de aplicação emitida pela venue.
 - **AppVersion:** string opcional de versão do cliente.
 - **AppComments:** texto livre registrado no login.
 - **StreamingHost:** host do Lightstreamer (padroniza para o host de streaming público).
- **AutoReconnect:** quando habilitado, encaminha para o WatchDog do cliente WebSocket subjacente, de modo que a sessão de streaming reconecta automaticamente após uma queda de rede (por padrão true).
- **REST_API:** referência ao componente **TsgcHTTP_API_Forex**.
- **Lightstreamer:** referência ao cliente de subprotocolo interno **TsgcWSPClient_Lightstreamer** (AdapterSet é fixo em STREAMINGALL).
- **SessionToken:** somente leitura. Token de sessão retornado pela chamada REST LogOn; usado como *LS_password* para o Lightstreamer.
- **ClientAccountId / TradingAccountId:** somente leitura. Preenchido após um LogOn bem-sucedido.

Eventos Publicados (Published)

OnForexConnect

Disparado quando a sessão de streaming foi criada e as inscrições podem começar.

OnForexDisconnect

Disparado quando a sessão de streaming termina (fechamento do servidor, queda de rede ou Disconnect explícito).

OnForexException

Gerado em erros de REST (incluindo falha de LogOn) e em falhas de sessão de streaming.

OnForexPriceTick

Disparado para cada atualização de PRICES: Bid, Offer, Price, Change, Direction, AuditId, TickDate.

OnForexOrderUpdate

Disparado para cada atualização de ORDERS (nova, preenchida, cancelada, rejeitada).

OnForexPositionUpdate

Disparado para cada atualização de posição aberta na TradingAccount ativa.

OnForexAccountMargin

Disparado para cada atualização CLIENTACCOUNTMARGIN / TRADEMARGIN (equity, margem, margem livre, moeda).

OnForexQuote

Disparado para cada atualização de QUOTES (fluxo de request-for-quote).

OnForexSubscribe

Disparado cada vez que o servidor confirma uma assinatura Lightstreamer (SUBOK / SUBCMD). Parâmetros: *aAdapter* (PRICES, ORDERS, QUOTES, CLIENTACCOUNTMARGIN ou TRADEMARGIN), *aGroup* (grupo de itens, por exemplo *ID.<ClientAccountId>*), *aItems*, *aFields*.

OnForexUnsubscribe

Disparado quando uma subscrição termina — seja via **UnwatchMarket**, na desconexão do streaming, ou quando o servidor envia um frame UNSUB. Parâmetros: *aAdapter*, *aGroup*.

Most common uses

- **REST API**
 - [Como conectar à REST API](#)
 - [Como colocar ordens de Trade e Stop-Limit](#)
 - [Como consultar contas, posições e histórico de ordens](#)
 - [Como obter dados de mercado](#)
- **WebSockets API**
 - [Como Conectar à WebSocket API](#)
 - [Como Fazer Subscribe a Streaming Adapters](#)

Exemplo

Crie o cliente, defina as credenciais, conecte, faça subscribe a um stream de preços de mercado e desconecte.

```
TsgcWSAPI_Forex *oForex = new TsgcWSAPI_Forex(NULL);
oForex->Credentials->UserName = "<username>";
oForex->Credentials->Password = "<password>";
oForex->Credentials->AppKey = "<appkey>";
oForex->OnForexPriceTick = ForexPriceTick;
oForex->Connect();
oForex->WatchMarket(401484830);
oForex->Disconnect();
```

Forex | Connect REST API

Cada chamada à Trading API da Forex.com / StoneX / CityIndex requer uma sessão ativa. O estado da sessão é obtido com **LogOn**, encerrado com **LogOff** e mantido ativo com **Ping**.

Credenciais

Defina as credenciais no componente cliente principal. Elas são copiadas para a sessão REST na primeira chamada.

- **UserName:** nome de usuário da conta Forex.com / StoneX / CityIndex.
- **Password:** senha da conta.
- **AppKey:** chave de aplicação emitida pela venue.
- **AppVersion / AppComments:** strings opcionais registradas no login.

Métodos de Session

Method	Parâmetros	Descrição
LogOn	UserName, Password, App-Key	Abre uma sessão REST. Retorna a resposta JSON completa e armazena SessionToken, ClientAccountId e TradingAccountId no cliente principal. Todas as chamadas REST e de streaming posteriores se autenticam com esse token.
LogOff		Invalida a sessão no servidor e limpa o token.
Ping		Mantém a sessão ativa. A sessão do servidor expira após um período de inatividade; emita Ping periodicamente quando nenhuma outra chamada REST estiver sendo feita.
GetServiceStatus		Reporta o status atual do serviço (nenhuma autenticação necessária).

Token Flow

O token retornado por **LogOn** é anexado a cada requisição REST subsequente como o cabeçalho HTTP *User-Name*. O mesmo token é reutilizado pelo cliente de streaming como o campo *LS_password* quando a sessão Lightstreamer é criada. Não há necessidade de chamar LogOn duas vezes: chamar **Connect** no cliente principal realiza o LogOn REST primeiro e só então abre o socket de streaming.

Exemplo

Abrir uma sessão REST e chamar Ping

```
TsgcWSAPI_Forex *oForex = new TsgcWSAPI_Forex(NULL);
oForex->Credentials->UserName = "<username>";
oForex->Credentials->Password = "<password>";
oForex->Credentials->AppKey = "<appkey>";
ShowMessage(oForex->REST_API->LogOn(oForex->Credentials->UserName,
                                   oForex->Credentials->Password,
                                   oForex->Credentials->AppKey));
ShowMessage(oForex->REST_API->Ping());
oForex->REST_API->LogOff();
```

Forex | Trade and Stop-Limit Orders

A REST API expõe cinco endpoints de gerenciamento de ordens. Cada endpoint é fornecido em três formas de sobrecarga para que o chamador possa escolher o nível de controle necessário.

Endpoints

Method	Descrição
NewTradeOrder	Coloca uma ordem de trade market / at-best em um mercado de CFD.
UpdateTradeOrder	Anexa ou altera a proteção stop / limit anexada a uma negociação aberta existente.
NewStopLimitOrder	Coloca uma ordem stop ou limit pendente (ainda não no mercado).
UpdateStopLimitOrder	Modifica uma ordem stop ou limitada pendente existente.
CancelOrder	Cancela uma ordem stop ou limit pendente.

Overload Tiers

Os métodos de ordem estão disponíveis em duas sobrecargas:

1. **Sobrecarga com parâmetros escalares** (apenas NewTradeOrder e CancelOrder) — recebe os campos mais utilizados (MarketId, TradingAccountId, Direction, Quantity, BidPrice / OfferPrice, AuditId para NewTradeOrder; OrderId, TradingAccountId, MarketId para CancelOrder). O helper constrói a requisição internamente. Útil para ordens simples, sem proteção.
2. **Sobrecarga com objeto de ordem tipado** — recebe um **TsgcHTTPForexTradeOrder**, **TsgcHTTPForexStopLimitOrder** ou **TsgcHTTPForexCancelOrder**. Esses objetos expõem cada campo documentado, incluindo **IfDone** (brackets de stop / limit anexados), arrays **Close** (semântica de fechamento por order-id), **PositionMethodId**, **Reference** e flags de preenchimento parcial. Este é o caminho recomendado para qualquer coisa além de uma simples ordem de mercado; a convenção de construção de objetos é compartilhada com outras exchanges, como a Kucoin.

Campos de TsgcHTTPForexTradeOrder

- **MarketId**: id numérico de mercado retornado por ListCfdMarkets ou FullSearchWithTags.
- **TradingAccountId**: a conta de trading ativa. Leia-a a partir do cliente principal após LogOn.
- **Direction**: *buy* ou *sell*.
- **Quantity**: tamanho do trade na unidade nativa do mercado.
- **BidPrice / OfferPrice**: preços observados pelo cliente quando a ordem foi enviada (a Forex.com os utiliza para detectar cotações desatualizadas).
- **AuditId**: id de auditoria de preço do último tick de preço.
- **PositionMethodId**: comportamento de netting / hedging (1 = *Netting*, 2 = *Hedging*).
- **IfDone**: lista de ordens anexadas (stop, limit, guaranteed stop, trailing stop) que são ativadas quando a ordem-pai é preenchida.
- **Close**: array de ids de ordens a fechar quando esta ordem for executada (fechamento parcial).
- **Reference**: referência de cliente fornecida pelo chamador, ecoada de volta nas atualizações de ordem.

Exemplo

Comprar 1000 EUR/USD a mercado com take-profit e stop-loss anexados

```
TsgcWSAPI_Forex *oForex = new TsgcWSAPI_Forex(NULL);
oForex->Credentials->UserName = "<username>";
oForex->Credentials->Password = "<password>";
oForex->Credentials->AppKey = "<appkey>";
oForex->Connect();

TsgcHTTPForexTradeOrder *oOrder = new TsgcHTTPForexTradeOrder();
oOrder->MarketId = 401484830;
oOrder->TradingAccountId = oForex->TradingAccountId;
oOrder->Direction = fdBuy;
oOrder->Quantity = 1000;
```

```
oOrder->BidPrice      = 1.09120;  
oOrder->OfferPrice     = 1.09130;  
ShowMessage(oForex->NewTradeOrder(oOrder));  
delete oOrder;
```


Forex | Account Queries

Esses endpoints leem o estado da conta e o histórico de ordens. Eles requerem uma sessão ativa (consulte [Connect REST API](#)).

Endpoints

Method	Parâmetros	Descrição
GetClientAndTradingAccount		Retorna a conta do cliente e a lista de contas de negociação vinculadas ao usuário autenticado.
ListOpenPositions	TradingAccountId	Retorna todas as posições atualmente abertas em uma conta de negociação.
ListActiveStopLimitOrders	TradingAccountId	Retorna todas as stop e limit orders pendentes.
GetOrder	OrderId	Retorna uma única ordem por id, incluindo o histórico de execução e as ordens IfDone anexadas.
ListTradeHistory	TradingAccountId, Max-Results	Retorna o histórico de trade-order da conta, mais recentes primeiro.
ListStopLimitOrderHistory	TradingAccountId, Max-Results	Retorna o histórico de ordens stop-limit da conta, mais recentes primeiro.
SimulateTrade	JSON / scalar / order-object	Executa uma trade order pelo validador do servidor sem submetê-la. Útil para verificações de pré-voo de margem e tolerância de preço. Aceita os mesmos três níveis de sobrecarga que NewTrade-Order.

Endpoints Descontinuados

A referência oficial da Forex.com / StoneX / CityIndex ainda documenta alguns endpoints que o servidor ativo não serve mais. Estes intencionalmente não são expostos pelo componente:

- **GetSystemLookup** — a referência descreve um endpoint de lookup genérico, mas o servidor responde com HTTP 404 para cada chave de lookup conhecida. Use **GetClientAndTradingAccount** e **ListCfdMarkets** em vez disso.
- **/news/headlines/*** — a família news-feed ainda está listada na referência, mas retorna HTTP 500 no servidor ativo. Nenhuma substituição está disponível através da Trading API; as notícias são servidas somente através do front end web.

Exemplo

Lista as posições abertas e as últimas 50 negociações

```
TsgcWSAPI_Forex *oForex = new TsgcWSAPI_Forex(NULL);
oForex->Credentials->UserName = "<username>";
oForex->Credentials->Password = "<password>";
oForex->Credentials->AppKey = "<appkey>";
oForex->Connect();
ShowMessage(oForex->REST_API->ListOpenPositions(oForex->TradingAccountId));
ShowMessage(oForex->REST_API->ListTradeHistory(oForex->TradingAccountId, 50));
```

Forex | Market Data

Metadados estáticos de mercado e preços históricos. Esses endpoints são usados para descobrir valores de **MarketId** que são então fornecidos aos endpoints de negociação e aos adaptadores de streaming.

Endpoints

Method	Parâmetros	Descrição
GetMarketInformation	MarketId	Retorna o descritor completo do mercado (símbolo, casas decimais, tamanho do pip, fator de margem, horários de negociação).
ListCfdMarkets	SearchByMarketName, ClientAccountId, MaxResults	Lista os mercados CFD que correspondem a um prefixo de nome.
FullSearchWithTags	Query, Tags, MaxResults	Busca em texto livre em todos os mercados com filtros de tag opcionais (FX, Indices, Shares, Commodities).
GetPriceBars	MarketId, Interval, Span, PriceBars	Barras OHLC históricas. O Interval é um dos seguintes: MINUTE, HOUR, DAY, WEEK; o Span é o número de intervalos por barra.
GetPriceTicks	MarketId, PriceTicks	Snapshot do tick mais recente de um mercado (bid / ask / mid).

Exemplo

Pesquisa EUR/USD e lê as últimas 100 barras de 1 minuto

```
TsgcWSAPI_Forex *oForex = new TsgcWSAPI_Forex(NULL);
oForex->Credentials->UserName = "<username>";
oForex->Credentials->Password = "<password>";
oForex->Credentials->AppKey = "<appkey>";
oForex->Connect();
ShowMessage(oForex->REST_API->ListCfdMarkets("EUR/USD", 0, 10));
ShowMessage(oForex->REST_API->GetPriceBars(401484830, "MINUTE", 1, 100));
```

Forex | Connect WebSocket API

Forex.com / StoneX / CityIndex transmite dados de mercado e atualizações de conta por meio do Lightstreamer. O componente fala o protocolo nativo Lightstreamer TLCP 2.5 sobre um WebSocket; nenhuma biblioteca nativa do Lightstreamer é necessária.

Transport

- **Subprotocolo WebSocket:** *TLCP-2.5.0.lightstreamer.com*. Anunciado no cabeçalho de requisição *Sec-WebSocket-Protocol*.
- **Conjunto de adaptadores:** *STREAMINGALL*. Configurado no cliente de subprotocolo **Lightstreamer** interno; todos os adaptadores de dados da Forex.com (PRICES, ORDERS, QUOTES, CLIENTACCOUNTMARGIN, TRADEMARGIN) ficam sob esse conjunto de adaptadores.
- **Authentication:** o token de sessão obtido da chamada REST **LogOn** é enviado como *LS_password* ao criar a sessão Lightstreamer, junto com *LS_user* definido como o nome de usuário da conta.

Reconexão

Quando **AutoReconnect** está habilitado (o padrão), o WatchDog do cliente WebSocket subjacente reabre o socket após erros de rede. O próprio protocolo Lightstreamer sinaliza frames *LOOP* quando o servidor quer que o cliente vá para uma URL diferente ou refaça o bind da sessão; o componente trata isso de forma transparente e só gera **OnForexDisconnect** / **OnForexConnect** em torno de quebras completas de sessão.

Quando a sessão REST expira (sem Ping por muito tempo), o servidor de streaming fecha a sessão. O componente chama **LogOn** novamente usando as credenciais armazenadas, atualiza **SessionToken** e reinicia a sessão Lightstreamer com o novo token.

Exemplo

Abre a sessão de streaming

```
TsgcWSAPI_Forex *oForex = new TsgcWSAPI_Forex(NULL);
oForex->Credentials->UserName = "<username>";
oForex->Credentials->Password = "<password>";
oForex->Credentials->AppKey = "<appkey>";
oForex->OnForexConnect = ForexConnected;
oForex->OnForexDisconnect = ForexDisconnected;
oForex->OnForexException = ForexException;
oForex->AutoReconnect = true;
oForex->Connect();
oForex->Disconnect();
```

Forex | Assinar Adaptadores de Streaming

Os dados de streaming são entregues por meio de cinco data adapters do Lightstreamer sob o conjunto de adapters *STREAMINGALL*. Cada adapter tem sua própria convenção de nomenclatura de item e modo de assinatura — **MERGE** para estado no estilo snapshot (preço atual, margem atual da conta) e **RAW** para canais de stream de eventos (atualizações de ordens, atualizações de posições, eventos de cotação).

Adaptadores

DataAdapter	Item	Mode	Evento	Fields
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, MarginIndicator, NetEquity, OpenTradeEquity, TotalMarginRequirement, TradeableFunds, CurrencyId, CurrencyISOCode
TRADEMARGIN	TRADEMARGIN.{ClientAccountId}	RAW	OnForexPositionUpdate	OrderId, ClientAccountId, MarketId, MarketTypeId, DirectionId, Direction, Quantity, OpenPrice, CurrentPrice, Multiplier, MarginRequirement, Currency, OpenTradeEquity

Notas:

- PRICES e CLIENTACCOUNTMARGIN utilizam a forma de item unificada *ID.{numericId}* e transmitem um único registro mutável por item (MERGE).
- ORDERS, QUOTES e TRADEMARGIN usam o prefixo legado por adaptador (*ORDERS.{id}*, *QUOTES.{id}*, *TRADEMARGIN.{id}*) e fazem stream de uma enxurrada de eventos (RAW). Um item carrega cada ordem / quote / negociação para o **ClientAccountId** logado; os registros se identificam por **OrderId** / **QuoteId**.
- O registro TRADEMARGIN corresponde ao *TradeMarginDTO* oficial do SDK legado da CityIndex. **DirectionId** chega como um código numérico (1 = buy, 2 = sell); a biblioteca também expõe **Direction** como a string "buy" / "sell". **OpenPrice** é o preço de entrada do trade (campo do servidor *PriceTakenAt*), **CurrentPrice** o preço de valoração mais recente (*PriceCalculatedAt*), **OpenTradeEquity** o P&L flutuante na moeda da conta (*OTEConverted*).
- O campo **Direction** de ORDERS é o código numérico publicado pelo servidor (1 / 2); **Status** é uma string (por exemplo, "Accepted", "Open", "Cancelled"); **OrderType** é uma string ("Market" / "Stop" / "Limit").
- O feed TRADEMARGIN não publica **TradingAccountId**, **MarketName** nem um timestamp por trade; um item carrega cada posição do **ClientAccountId** conectado. Chame **REST_API.ListOpenPositions** para esses campos. O feed ORDERS não publica **TriggerPrice** nem **Applicability**; chame **REST_API.GetOrder** quando esses forem necessários.

Métodos Auxiliares

O cliente principal encapsula a infraestrutura SUB / UNSUB do Lightstreamer. As mais comuns são:

- **WatchMarket(aMarketId)**: inscreve-se em PRICES de um mercado. As atualizações chegam em **OnForexPriceTick**.
- **UnwatchMarket(aMarketId)**: remove a assinatura.
- **WatchAccount**: assina ORDERS, CLIENTACCOUNTMARGIN e TRADEMARGIN para as contas logadas. As atualizações chegam nos eventos correspondentes.
- **WatchQuotes**: assina o adaptador QUOTES para fluxos de request-for-quote.

Eventos do Ciclo de Vida de Subscrição

O cliente principal gera dois eventos de ciclo de vida que espelham os frames TLCP SUBOK / UNSUB subjacentes:

- **OnForexSubscribe(Sender, aAdapter, aGroup, altems, aFields)**: disparado quando o servidor confirma qualquer assinatura Lightstreamer. *aAdapter* é o nome do data adapter (PRICES, ORDERS, QUOTES, CLIENTACCOUNTMARGIN, TRADEMARGIN); *aGroup* é o grupo de itens (por exemplo, *ID.<ClientAccountId>* ou *ORDERS.<ClientAccountId>*); *altems* e *aFields* são as contagens reportadas pelo servidor no frame SUBOK / SUBCMD.
- **OnForexUnsubscribe(Sender, aAdapter, aGroup)**: disparado quando uma assinatura termina — via **UnwatchMarket**, na desconexão do streaming, ou quando o servidor envia um frame UNSUB.

Exemplo

Assinar ticks de preço de EUR/USD

```
TsgcWSAPI_Forex *oForex = new TsgcWSAPI_Forex(NULL);
oForex->Credentials->UserName = "<username>";
oForex->Credentials->Password = "<password>";
oForex->Credentials->AppKey = "<appkey>";
oForex->OnForexPriceTick = ForexPriceTick;
oForex->Connect();
oForex->WatchMarket(401484830);
```

API Blockchain

Blockchain

A Blockchain WebSocket API permite que os desenvolvedores recebam notificações em tempo real sobre novas transações e blocos.

Uma vez que o WebSocket está aberto, você pode assinar um canal:

- **SubscribeTransactions:** Assine notificações de todas as novas transações bitcoin.
- **UnsubscribeTransactions:** Cancela a assinatura das notificações de todas as novas transações bitcoin.
- **SubscribeAddress:** Receba novas transações para um endereço bitcoin específico.
- **UnsubscribeAddress:** Para de receber novas transações para um endereço bitcoin específico.

As transações são recebidas no Evento **OnNewTransaction**:

```
{
  "op": "utx",
  "x": {
    "lock_time": 0,
    "ver": 1,
    "size": 192,
    "inputs": [
      {
        "sequence": 4294967295,
        "prev_out": {
          "spent": true,
          "tx_index": 99005468,
          "type": 0,
          "addr": "1BwGf3z7n2fHk6NoVJNkV32qwyAYsMhkWf",
          "value": 65574000,
          "n": 0,
          "script": "76a91477f4c9ee75e449a74c21a4decfb50519cbc245b388ac"
        },
        "script": "483045022100e4ff962c292705f051c2c2fc519fa775a4d8955bce1a3e29884b2785277999ed02200b537e"
      }
    ],
    "time": 1440086763,
    "tx_index": 99006637,
    "vin_sz": 1,
    "hash": "0857b9de1884eec314ecf67c040a2657b8e083e1f95e31d0b5ba3d328841fc7f",
    "vout_sz": 1,
    "relayed_by": "127.0.0.1",
    "out": [
      {
        "spent": false,
        "tx_index": 99006637,
        "type": 0,
        "addr": "1A828tTnkVFJfSvLCqF42ohZ51ksS3jJgX",
        "value": 65564000,
        "n": 0,
        "script": "76a914640cfd7b79d94d1c980133e3587bd6053f091f388ac"
      }
    ]
  }
}
```

- **SubscribeBlocks:** Receber notificações quando um novo bloco for encontrado. Nota: se a chain se dividir, você receberá mais de uma notificação para uma altura de bloco específica.
- **UnsubscribeBlocks:** Para de receber notificações quando um novo bloco é encontrado. Nota: se a chain se dividir, você receberá mais de uma notificação para uma altura de bloco específica.

Os blocos são recebidos no evento **OnNewBlock**:

```
{
  "op": "block",
  "x": {
```

```
    "txIndexes": [
      3187871,
      3187868
    ],
    "nTx": 0,
    "totalBTCSent": 0,
    "estimatedBTCSent": 0,
    "reward": 0,
    "size": 0,
    "blockIndex": 190460,
    "prevBlockIndex": 190457,
    "height": 170359,
    "hash": "00000000000006436073c07dfa188a8fa54fefadf571fd774863cda1b884b90f",
    "mrklRoot": "94e51495e0e8a0c3b78dac1220b2f35ceda8799b0a20cfa68601ed28126cfcc2",
    "version": 1,
    "time": 1331301261,
    "bits": 436942092,
    "nonce": 758889471
  }
}
```

API Cex

Cex

A WebSocket API permite obter notificações em tempo real sem enviar requisições extras, tornando-a uma forma mais rápida de obter dados da exchange

O componente Cex tem uma propriedade chamada Cex onde você pode preencher as API Keys fornecidas pela Cex para obter acesso aos dados da sua conta.

Message encoding

- Todas as mensagens são codificadas no formato JSON.
- Os preços são apresentados como strings para evitar erros de arredondamento ao fazer o parsing do JSON no lado do cliente.
- A compressão de frames WebSocket não é suportada pelo servidor.
- O tempo é apresentado como um timestamp UNIX inteiro em segundos.

Autenticação

Para obter acesso aos dados WebSocket da CEX.IO, você deve estar autorizado.

- Faça login na conta CEX.IO.
- Vá para a página <https://cex.io/trade/profile#/api>.
- Selecione o tipo de permissões necessárias.
- Clique no botão "Generate Key" e salve sua chave secreta, pois ela se tornará inacessível após a ativação.
- Ative sua chave.

Conectividade

- Se um Socket conectado ficar inativo por 15 segundos, o servidor CEX.IO enviará uma mensagem PING.
- Apenas o servidor pode ser um Iniciador de requisição PING.
- O servidor envia ping apenas ao usuário autenticado.
- O usuário tem que responder com uma mensagem PONG. Caso contrário, o WebSocket será DESCONECTADO. Isso é tratado automaticamente pela biblioteca.
- Para o usuário autenticado, caso não haja notificação ou ping do servidor dentro de 15 segundos, seria mais seguro enviar uma requisição como 'ticker' ou 'get-balance' e receber uma resposta, para garantir a conectividade e a autenticação.

Canais Públicos

Esses canais não requerem autenticação. As respostas do servidor são recebidas através do evento OnCexMessage.

- **SubscribeTickers:** Feed de ticker apenas com o preço da transação realizada em todos os pares (descontinuado)

```
{
  "e": "tick",
  "data": {
    "symbol1": "BTC",
    "symbol2": "USD",
    "price": "428.0123"
  }
}
```



```
}
}
```

- **SubscribeChart:** feeds de gráfico OHLCV com números de Open, High, Low, Close, Volume (descontinuado)

```
{
  'e': 'ohlcv24',
  'pair': 'BTC:USD',
  'data': [
    '418.2936',
    '420.277',
    '412.09',
    '416.9778',
    '201451078368'
  ]
}
```

- **Subscribe Pair:** feed de Market Depth (descontinuado)

```
{
  '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
    }
  }
}
```

- **Subscribe Pair:** feed do Order Book (obsoleto)

```
{
  '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
    ]
  }
}
```

Private Channels

Para acessar esses canais, primeiro chame o método `Authenticate`. As respostas do servidor são recebidas por meio do 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

No sistema CEX.IO, as ordens podem estar presentes no trade engine ou em um banco de dados de arquivo. Pode haver períodos de tempo (~2 segundos ou mais) em que a ordem está concluída/cancelada, mas ainda não movida para o banco de dados de arquivo. Isso significa que você não pode vê-la utilizando as chamadas: `archived-orders/open-orders`. Esta chamada permite obter informações da ordem em qualquer caso. As respostas podem ter formato diferente dependendo da localização das ordens.

```

{
  "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": "3057.5333333",
        "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

APIs suportadas

- **WebSockets API:** conecta a um servidor websocket público e fornece atualizações de dados de mercado em tempo real.

WebSockets API

O WebSocket é um protocolo de comunicação full-duplex baseado em TCP. Full-duplex significa que ambas as partes podem enviar mensagens umas às outras de forma assíncrona usando o mesmo canal de comunicação. Esta seção descreve quais mensagens o Exchange Plus e o Cliente devem enviar um ao outro. Todas as mensagens devem ser objetos JSON válidos.

A WebSocket API é usada principalmente para obter informações ou realizar ações que não estão disponíveis ou não são fáceis de fazer usando a REST API. No entanto, algumas requisições ou ações são possíveis de fazer tanto na REST API quanto na WebSocket API. O Exchange Plus envia mensagens ao Cliente como resposta a uma requisição enviada anteriormente pelo Cliente, ou como notificação sobre algum evento (sem requisição prévia do Cliente).

Public API Calls

O limite de taxa da API pública é imposto para proteger o sistema de ataques DDoS e garantir que todos os Clientes tenham o mesmo nível de acesso estável aos endpoints da Exchange Plus API. As requisições públicas são limitadas pelo endereço IP a partir do qual as requisições da API pública são feitas. Os limites de requisição são determinados a partir do custo associado a cada chamada da API pública. Por padrão, cada requisição pública tem um custo de 1 ponto, mas para algumas requisições específicas esse custo pode ser maior. Consulte as informações atualizadas de custo do limite de taxa de requisição na especificação de cada método.

O Exchange Plus limita as chamadas à Public API a no máximo 100 pontos por minuto, considerando que cada chamada à Public API tem seu custo (veja abaixo). Se o limite de taxa de requisições for atingido, o Exchange Plus responde com erro, envia o evento de desconexão ao Cliente e, em seguida, fecha a conexão WS. O Exchange Plus continuará a atender o Cliente a partir do próximo minuto do calendário. No exemplo a seguir, o contador de requisições será zerado às 11:02:00.000.

Method	Descrição
GetTicker	Este método foi projetado para obter informações atuais sobre o Ticker, incluindo dados sobre preços atuais, mudanças de preço & volume em 24h, último evento de negociação, etc. de certos ativos.
GetOrderBook	Este método permite que o Cliente receba o snapshot atual do order book para um par de negociação específico.
GetCandles	Utilizando o método Candles, o Cliente pode receber candles OHLCV históricos de diferentes resoluções e tipos de dados. O Cliente pode indicar filtros adicionais de timeframe e limit para tornar a resposta mais precisa aos requisitos do Cliente.
GetTradeHistory	Este método permite que o Cliente obtenha dados históricos sobre as negociações ocorridas no par de negociação solicitado. O Cliente pode complementar a requisição Trade History com parâmetros de filtro adicionais, como período de tempo, intervalo de tradelds, side etc. para receber negociações que correspondam aos parâmetros da requisição.

GetServerTime	Este método é usado para obter o horário atual no servidor Exchange Plus. Pode ser útil para aplicações que precisam estar sincronizadas com o horário do servidor.
GetPairsInfo	O método Pair Info permite que o Cliente receba os parâmetros de todos os pares de negociação suportados.
GetCurrenciesInfo	O método Currencies Info permite que o Cliente receba os parâmetros de todas as moedas configuradas no Exchange Plus, bem como a disponibilidade de depósito e saque entre o Exchange Plus e a CEX.IO Wallet.
GetProcessingInfo	Esta requisição permite que o Cliente receba informações detalhadas sobre as opções disponíveis para fazer depósitos a partir de carteiras externas e saques para carteiras externas, para cada criptomoeda suportada, incluindo o nome da criptomoeda e os blockchains disponíveis para depósito\saque. Além disso, para cada blockchain suportado, são indicados o tipo de criptomoeda no blockchain indicado, a disponibilidade atual de depósito\saque, os valores mínimos para depósitos\saques e as taxas de saque externo. As informações de processamento tornam o Cliente mais flexível na escolha do blockchain desejado para receber o endereço de Depósito e iniciar saques externos por meio de um determinado blockchain, de modo que o Cliente use a forma mais conveniente de transferir seus ativos de cripto de ou para o Ecossistema CEX.IO.
SubscribeOrderBook	O Cliente, ao assinar via WebSocket, pode assinar o feed do livro de ordens do par de negociação solicitado. Em resposta à requisição Order Book Subscribe, o Cliente receberá o snapshot atual (inicial) do livro de ordens do par solicitado com o número seqId indicado. Para acompanhar as atualizações seguintes do Livro de Ordens, o Cliente precisa assinar via WebSocket as mensagens "order_book_increment", que conteriam o nome do par de negociação, o número seqId e os deltas dos níveis de preço de Bids e Asks.
UnSubscribeOrderBook	Cancela a assinatura do canal de order book.
SubscribeTrade	Ao usar o método Trade Subscribe, o Cliente pode assinar via WebSocket o feed ao vivo de eventos de negociação que ocorrem no par de negociação solicitado. Em resposta à requisição Trade Subscribe, o Cliente receberá um identificador único da assinatura de negociação, que deve ser usado posteriormente para cancelamento quando a assinatura de negociação não for mais necessária para o Cliente. O Cliente deve assinar via WebSocket as mensagens "tradeHistorySnapshot" e "tradeUpdate" para receber os snapshots iniciais e periódicos do Histórico de Negociações, e os eventos de negociação ao vivo do par de negociação solicitado.
UnSubscribeTrade	Cancela a assinatura do canal de trade.

Exemplo: obter o ticker mais recente do par BTC-USD

```
TsgcWebSocketClient* oClient;
TsgcWSAPI_CexPlus* oCexPlus;

void __fastcall OnCexPlusConnectEvent(System::TObject* Sender)
{
    oCexPlus->GetTicker("BTC-USD");
}

void __fastcall OnCexPlusMessageEvent(System::TObject* Sender, UnicodeString Event, UnicodeString Msg)
{
    ShowMessage("Ticker data: " + Msg);
}

void SetupCexPlus()
{
    oClient = new TsgcWebSocketClient(NULL);
    oCexPlus = new TsgcWSAPI_CexPlus(NULL);
    oClient->Active = true;
    oCexPlus->Client = oClient;
    oCexPlus->OnConnect = OnCexPlusConnectEvent;
    oCexPlus->OnMessage = OnCexPlusMessageEvent;
}
```

Chamadas de API Privada

O Exchange Plus utiliza chaves de API para permitir o acesso às Private APIs.

O cliente pode gerar, configurar e gerenciar api keys, definir níveis de permissão, IPs em whitelist para a API key etc. por meio do Web Terminal da Exchange Plus na seção API Keys Management Profile.

Limite de API Keys: Por padrão, o Cliente pode ter até 5 API Keys.

Para restringir o acesso a determinadas funcionalidades ao usar API Keys, deve haver um conjunto específico de permissões definido para cada API Key. O conjunto de permissões definido pode ser editado posteriormente, se necessário.

Os seguintes níveis de permissão estão disponíveis para as API Keys:

- **Read:** nível de permissão para visualização de dados relacionados à conta, recebimento de relatórios, assinatura de dados de mercado etc.
- **Trade:** nível de permissão que permite colocar e cancelar ordens em nome da conta.
- **Funds Internal:** nível de permissão que permite transferir fundos entre contas (entre sub-contas ou conta principal e sub-contas) do Portfólio CEX.IO Exchange Plus.
- **Funds Wallet:** nível de permissão que permite transferir fundos das contas CEX.IO Exchange Plus Portfolio (conta principal e sub-contas) para a CEX.IO Wallet e vice-versa.

Method	Descrição
GetCurrentFee	Este método indica as taxas atuais em um momento específico, considerando o volume atualizado de 30d do Cliente e o dia da semana (as taxas podem ser diferentes, por exemplo, nos fins de semana).
GetFeeStrategy	A Fee Strategy retorna todas as opções de taxa que podem ser aplicadas ao Cliente, considerando o volume de trading do Cliente, dia da semana, pares, grupo de pares etc. Este método fornece informações sobre a estratégia geral de taxas, que inclui todos os possíveis valores de taxa de trading que podem ser aplicados ao Cliente. Para receber as taxas de trading atuais, com base no volume de trading dos últimos 30 dias do Cliente, o Cliente deve utilizar o método [Current Fee]. Para receber o volume de trading atual dos últimos 30 dias, o Cliente deve utilizar o método [Volume].
GetVolume	Esta requisição permite que o Cliente receba seu volume de negociação dos últimos 30 dias em equivalente em USD.
CreateAccount	Esta requisição permite que o Cliente crie uma nova subconta. Por padrão, o Cliente pode ter até 5 subcontas, incluindo a conta principal.
GetAccountStatus	Ao utilizar o método Account Status V3, o Cliente pode descobrir o saldo atual e seu equivalente indicativo em moeda convertida (por padrão "USD"), os valores bloqueados em ordens abertas (ativas) referentes a cada sub-account e moeda. Se o trading fee balance estiver disponível para o Cliente, então a resposta também conterá dados gerais de trading fee balance, como nome do promo, nome da moeda, saldo total e data de expiração deste promo no Trading Fee Balance. É responsabilidade do Cliente rastrear o saldo de trading disponível de suas sub-accounts, já que o saldo atual da sub-account é reduzido pelo valor de saldo bloqueado em ordens abertas (ativas) na sub-account.
GetOrders	Esta requisição permite que o Cliente descubra informações sobre suas ordens.
NewOrder	O cliente pode enviar novas ordens via WebSocket API utilizando a Do My New Order Request. Junto com a resposta a essa requisição, a Exchange Plus envia mensagens Account Event e Execution Report ao cliente se a requisição for bem-sucedida. A mensagem de resposta indica o status mais recente da ordem disponível no sistema no momento do envio da resposta. Se o cliente não recebeu uma mensagem de resposta à Do My New Order Request, o cliente pode consultar o status atual da ordem utilizando a Get My Orders Request com o parâmetro clientOrderId. Ao enviar uma requisição de nova ordem, é altamente reco-

	mendável utilizar o parâmetro <code>clientOrderId</code> , que corresponde à requisição de nova ordem específica do lado do cliente. A Exchange Plus evita o registro múltiplo de ordens com o mesmo <code>clientOrderId</code> . Se mais de uma nova ordem com <code>clientOrderId</code> idêntico e outros parâmetros de ordem idênticos forem identificadas, a Exchange Plus registra apenas a primeira ordem e retorna o status dessa ordem ao cliente em resposta à segunda e às subsequentes requisições de nova ordem com os mesmos parâmetros. Se mais de uma nova ordem com <code>clientOrderId</code> idêntico, mas com outros parâmetros de ordem diferentes, forem identificadas, a Exchange Plus processa apenas a primeira ordem e rejeita a segunda e as subsequentes requisições de nova ordem com o mesmo <code>clientOrderId</code> , mas com outros parâmetros de ordem diferentes.
NewMarketOrder	Coloca uma nova ordem a mercado.
NewLimitOrder	Coloca uma nova limit order.
CancelOrder	O cliente pode cancelar ordens. Junto com uma resposta a esta requisição, o Exchange Plus envia mensagens Account Event e Execution Report ao Cliente se esta requisição for bem-sucedida. Além disso, se a requisição para cancelar uma ordem for recusada, o Exchange Plus envia a mensagem Order Cancellation Rejection.
CancelAllOrders	O Cliente pode cancelar todas as ordens abertas via WebSocket API. Junto com uma resposta a esta requisição, o Exchange Plus iniciará o processo de cancelamento de todas as ordens abertas e enviará as mensagens correspondentes de Account Event e Execution Report ao Cliente.
GetTransactionHistory	Esta requisição permite que o Cliente descubra suas transações financeiras (depósitos, saques, transferências internas, comissões ou negociações).
GetFundingHistory	Esta requisição permite que o Cliente encontre suas transações de depósito e saque.
InternalTransfer	O Cliente pode solicitar a transferência de dinheiro entre suas subcontas ou entre sua conta principal e subconta. O Exchange Plus não cobra do Cliente nenhuma comissão pela transferência de fundos entre suas contas. Junto com a resposta a esta requisição, o Exchange Plus envia mensagens de Account Event ao Cliente se esta requisição for bem-sucedida.
GetDepositAddress	Este método pode ser utilizado pelo Cliente para receber um endereço cripto para depositar criptomoeda. O endereço de depósito pode ser gerado para contas principais e subcontas. A lista de blockchains disponíveis para gerar o endereço de depósito pode ser recebida pelo Cliente através da requisição Get Processing Info.
FundsDeposit-FromWallet	O Cliente pode depositar fundos da CEX.IO Wallet para a conta Exchange Plus. O sistema evita o processamento de múltiplas requisições de depósito com o mesmo <code>clientTxId</code> . Se múltiplas requisições de depósito com <code>clientTxId</code> idêntico forem recebidas, o sistema processa apenas a primeira requisição de depósito e rejeita a segunda e as subsequentes requisições de depósito com o mesmo <code>clientTxId</code> .
FundsWithdrawal-ToWallet	O Cliente pode sacar fundos da conta Exchange Plus para a CEX.IO Wallet. O sistema evita múltiplas requisições de saque com o mesmo <code>clientTxId</code> . Se múltiplas requisições de saque com <code>clientTxId</code> idênticos forem recebidas, o sistema processa apenas a primeira requisição de saque e rejeita a segunda e as subsequentes com o mesmo <code>clientTxId</code> .
GetWalletBalance	Recupera as informações de saldo da Carteira CEX.IO.
SubscribeAccountEvents	Assina notificações de eventos de conta em tempo real (mudanças de saldo, preenchimentos de ordem).
UnSubscribeAccountEvents	Cancela a assinatura das notificações de eventos da conta.

Exemplo: obter as ordens.

```
TsgcWebSocketClient* oClient;
TsgcWSAPI_CexPlus* oCexPlus;
```

```

void __fastcall OnCexPlusAuthenticatedEvent(System::TObject* Sender)
{
    oCexPlus->GetOrders();
}

void __fastcall OnCexPlusMessageEvent(System::TObject* Sender, UnicodeString Event, UnicodeString Msg)
{
    ShowMessage("Orders: " + Msg);
}

void SetupCexPlus()
{
    oClient = new TsgcWebSocketClient(NULL);
    oCexPlus = new TsgcWSAPI_CexPlus(NULL);
    oClient->Active = true;
    oCexPlus->Client = oClient;
    oCexPlus->CexPlus->ApiKey = "your-api-key";
    oCexPlus->CexPlus->ApiSecret = "your-api-secret";
    oCexPlus->OnCexPlusAuthenticated = OnCexPlusAuthenticatedEvent;
    oCexPlus->OnMessage = OnCexPlusMessageEvent;
}

```

API Discord

Discord

Nota: A partir da API v10, o domínio da Discord API mudou de `discordapp.com` para `discord.com`. O componente foi atualizado para usar a versão 10 da API e o novo domínio.

Os Gateways são a forma de comunicação em tempo real do Discord sobre WebSockets seguros. Os clientes receberão eventos e dados pelo gateway ao qual estão conectados e enviarão dados pela REST API.

Autorização

Primeiro você deve gerar um novo Bot e copiar o Bot Token, que será utilizado para autenticar por meio da API. Em seguida, defina este token no Componente da API.

```
TsgcWSAPI_Discord1->DiscordOptions->BotOptions->Token = "...bot token here...";
```

Intents

Manter uma aplicação com estado pode ser difícil quando se trata da quantidade de dados que você precisa processar, especialmente em escala. Gateway Intents são um sistema para ajudar a reduzir essa carga computacional.

Ao se identificar para o gateway, você pode especificar um parâmetro intents que permite assinar condicionalmente "intents" predefinidos, grupos de eventos definidos pelo Discord. Se você não especificar um determinado intent, não receberá nenhum dos eventos de gateway agrupados nesse grupo. Os intents válidos são (o valor zero significa que todos os eventos são recebidos):

```
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

Os heartbeats são tratados automaticamente pelo componente, portanto você não precisa se preocupar com eles. Quando o cliente conecta ao servidor, o servidor envia uma resposta HELLO com um intervalo de heartbeat, e o componente lê a resposta e ajusta automaticamente o heartbeat para enviar um ping a cada x segundos. Às vezes, o servidor pode enviar um ping ao cliente; isso também é tratado automaticamente pelo cliente.

Conexão Pronta

Quando a conexão está pronta, após um login bem-sucedido e a autorização pelo servidor, o evento **OnDiscordReady** é gerado e então você pode começar a receber atualizações do servidor.

Connection Resume

Se a conexão fechar inesperadamente, quando o cliente tenta reconectar, ele chama o evento **OnDiscordBeforeReconnect**. O componente salva automaticamente todos os dados necessários para fazer um resume bem-sucedido, mas os parâmetros podem ser alterados se necessário. Se você não quiser reconectar e iniciar uma nova sessão limpa, basta definir Reconnect como False.

Se a sessão for retomada, o evento **OnDiscordResumed** é disparado. Se for uma nova sessão, **OnDiscordReady** será disparado.

Despachar eventos

Os eventos são despachados por meio de **OnDiscordDispatch**, então aqui você pode ler os eventos enviados pelo servidor ao cliente.

```
void OnDiscordDispatch(TObject *Sender, const string aEvent, const string RawData)
{
    DoLog("#discord dispatch: " + aEvent + " " + RawData);
}
```

O parâmetro **aEvent** contém o nome do evento.

RawData contém a mensagem JSON completa.

HTTP Requests

Para solicitar informações sobre guilds, usuários ou atualizar dados... em vez de usar mensagens websocket do gateway, o Discord exige o uso de requisições HTTP. Veja abaixo todos os métodos disponíveis para fazer uma requisição 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;
```

Exemplo: obter informações do usuário atual

```
result = GET_Request("/users/@me");
```

exemplo de resposta do servidor:

```
{
  "id": "637423922035480852",
  "username": "test",
  "avatar": null,
  "discriminator": "5125",
  "bot": true,
  "email": null,
  "verified": true,
  "locale": "en-US",
  "mfa_enabled": false,
  "flags": 0
}
```

API | OpenAI

A OpenAI Realtime API permite interações multimodais de baixa latência, incluindo experiências conversacionais speech-to-speech e transcrição em tempo real.

O componente **TsgcWSAPI_OpenAI** implementa a RealTime API da OpenAI.

Configuração

Utilize a propriedade **method** para selecionar **Conversation** ou **Transcription**, atualmente apenas o modo Transcription é suportado.

A propriedade **InputAudio** permite que você personalize os seguintes dados:

- **Language:** exemplo inglês (value = 'en').
- **Model:** qual modelo será usado, exemplo: whisper-1
- **Prompt:** prompt opcional para fornecer instruções ao modelo.

OpenAI

A API da OpenAI utiliza chaves de API para autenticação. Acesse sua página de [API Keys](#) para obter a chave de API que você utilizará em suas requisições.

Lembre-se de que sua chave de API é um segredo! Não a compartilhe com outras pessoas nem a exponha em qualquer código do lado do cliente (navegadores, aplicativos). As requisições de produção devem ser roteadas por meio do seu próprio servidor de backend, onde sua chave de API pode ser carregada com segurança a partir de uma variável de ambiente ou de um serviço de gerenciamento de chaves.

Esta **API Key** deve ser configurada na propriedade **OpenAIOptions.ApiKey** do componente. Opcionalmente, para usuários que pertencem a múltiplas organizações, você pode definir sua Organization na propriedade **OpenAIOptions.Organization** se sua conta pertencer a uma organização.

Uma vez que a API Key esteja configurada, veja abaixo uma lista de funções disponíveis para interagir com a API da OpenAI.

Azure

O cliente suporta o Microsoft Azure OpenAI Services, portanto você pode utilizar sua conta Azure para interagir com a Azure OpenAI API também. Para configurar o cliente para trabalhar com o Azure, siga os próximos passos:

1. Configure a propriedade **OpenAIOptions.Provider** = oapvAzure
2. Defina os valores de ResourceName e DeploymentId (esses valores podem ser localizados na sua Conta Azure)
 1. **OpenAIOptions.AzureOptions.ResourceName** = <your resource name>.
 2. **OpenAIOptions.AzureOptions.DeploymentId** = <your deployment id>.
3. Defina a API Key da sua Conta Azure
 1. **OpenAIOptions.ApiKey** = <azure api key>.

Exemplo de Transcrição

Encontre abaixo um exemplo de transcrição em tempo real usando a API da openAI

```
TsgcWebSocketClient *WClient = new TsgcWebSocketClient(NULL);
TsgcAudioRecorderWave *oAudio = new TsgcAudioRecorderWave(NULL);
TsgcWSAPI_OpenAI *OpenAI = new TsgcWSAPI_OpenAI(NULL);
OpenAI->Client = WClient;
OpenAI->AudioRecorder = oAudio;

OpenAI->OpenAIOptions->APIKey = "your-api-key-here";
```



```
OpenAI->OpenAIOptions->Method = sgcoaimTranscription;
OpenAI->OpenAIOptions->Provider = sgcoaipOpenAI;
OpenAI->InputAudio->Language = "en";
OpenAI->InputAudio->Model = "whisper-1";

void __fastcall TForm1::OnOpenAIAudioTranscriptionCompleted(TObject *Sender, TsgcWSOpenAIConversation_Item_Comple
    Log("#transcription_completed: " + aItem->Transcript);
}
```

API MEXC

MEXC é um exchange global de criptomoedas que expõe dados de mercado e de conta em streaming sobre conexões WebSocket seguras.

A WebSocket Spot API segue as especificações publicadas na documentação [Market Streams](#) e [User Data Streams](#). Os canais de mercado são entregues como frames Protocol Buffers, enquanto os streams privados utilizam JSON. O componente **TsgcWSAPI_MEXC** gerencia o ciclo de vida da conexão, a autenticação e a manutenção da listen key, de modo que as aplicações possam se concentrar no processamento dos dados enviados.

Além de streaming sobre WebSockets, o toolkit também expõe a Spot HTTP API completa. O componente **TsgcHTTP_API_MEXC_Spot** (unit **sgcHTTP_API_MEXC**) envolve cada endpoint REST oficial, de modo que as aplicações possam consultar dados de mercado ou negociar com requisições assinadas.

Configuração do componente

O endpoint WebSocket base é **wss://wbs-api.mexc.com/ws**. Quando **MEXCUserDataStreams.UserStream** está habilitado e uma chave de API válida é fornecida, o componente solicita automaticamente uma listen key e a anexa à URL de conexão.

Solte um componente **TsgcWebSocketClient** e um **TsgcWSAPI_MEXC** no formulário (ou crie-os em código), atribua a instância do cliente ao componente da API e ative o cliente para iniciar a sessão.

```
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("BTCUSD");
}
```

Properties

- **Client:** referência ao transporte **TsgcWebSocketClient**. O componente habilita o heartbeat do cliente (15 segundos) para manter a sessão ativa.
- **MEXCAPI.ApiKey / ApiSecret:** Credenciais da Spot API. A key é necessária para canais privados; o secret é necessário para assinar requisições de listen key.
- **MEXCUserDataStreams.UserStream:** quando **True** (padrão), o componente recupera uma listen key antes de conectar e a renova a cada 10 minutos. Desabilite-o para trabalhar somente com feeds públicos.
- **MEXCUserDataStreams.ListenKeyOnDisconnect:** ação executada quando o WebSocket desconecta enquanto uma listen key está ativa:
 - **blkmxodDeleteListenKey:** revoga a listen key via REST (padrão).
 - **blkmxodClearListenKey:** mantém a listen key no lado do servidor e limpa o valor em cache.
 - **blkmxodDoNothing:** reutiliza a mesma listen key na próxima conexão.
- **ListenKey:** propriedade somente leitura que expõe a listen key ativa retornada pela REST API.
- **RawMessages:** se habilitado, os frames de texto de entrada não são analisados e são encaminhados como JSON simples por meio dos eventos WebSocket herdados.

Eventos

- **OnConnect / OnDisconnect:** gerados quando a sessão WebSocket subjacente é estabelecida ou encerrada.

- **OnMEXCMarketStream**: disparado para cada frame de mercado Protobuf. O manipulador recebe um **TsgcMEXCSpotProtoMessage** cuja propriedade **MessageType** identifica o payload (Trade, KLine, Diff-Depth, LimitDepth, BookTicker, BookTickerBatch, MiniTicker ou MiniTickers). Utilize as classes auxiliares em **sgcWebSocket_API_MEXC_Proto** para inspecionar a estrutura decodificada.
- **OnMEXCResponse**: recebe confirmações de inscrição e respostas de ping/pong em formato JSON (**Id**, **Code**, **Message**).
- **OnMEXCUserDataStream**: disparado quando chegam mensagens privadas da conta (ordens, deals, atualizações de saldo). O nome do canal e o payload JSON bruto são fornecidos.
- **OnMEXCException**: captura exceções REST ou WebSocket (erros de listen key, falhas HTTP, problemas de análise, etc.).

O exemplo a seguir demonstra como decodificar um quadro de negociação:

Métodos de assinatura WebSocket

O componente expõe auxiliares para cada canal Spot público. Invoque o método **Subscribe*** para começar a transmitir dados e o correspondente **UnSubscribe*** para pará-lo.

Canais públicos

Method	Parâmetros	Descrição
SubscribeTrade	Symbol, Interval (ms)	Execuções de negociação agregadas entregues como Protobuf publicdeals .
SubscribeKline	Symbol, Interval	Atualizações de candlestick para intervalos configuráveis (1 minuto a 1 mês).
SubscribeDiffDepth	Symbol, Interval (ms)	Deltas incrementais de profundidade do order book para manutenção do book local.
SubscribeBookDepth	Symbol, Levels	Snapshots periódicos do livro de ordens limitadas com a profundidade solicitada (padrão 5).
SubscribeBookTicker	Symbol, Interval (ms)	Atualizações de melhor bid/ask para um único par de negociação.
SubscribeBookTicker-Batch	flag UTC	Atualizações agregadas de book ticker para múltiplos símbolos.
SubscribeMiniTickers	flag UTC	Estatísticas do mini ticker móvel de 24h para todos os instrumentos ativos.
SubscribeMiniTicker	Symbol, flag UTC	Mini ticker para um único símbolo.

Canais privados

Os tópicos privados exigem uma chave de API válida. O componente se inscreve usando a listen key ativa e a re-nova periodicamente.

Method	Descrição
SubscribeAccountUpdate	Mudanças de saldo e posição da conta.
SubscribeAccountDeals	Relatórios de execução para ordens preenchidas.
SubscribeAccountOrders	Atualizações de ciclo de vida de ordens (new, cancelled, rejected, etc.).

Utilize os métodos **UnSubscribe*** correspondentes para encerrar um stream. O componente envia automaticamente comandos **PING** e valida o **PONG** retornado para monitorar a conectividade.

Endpoints REST públicos

Métodos REST disponíveis publicamente.

Method	Descrição
Ping	Verificação de conectividade que chama /api/v3/ping e retorna True quando a exchange responde com um objeto vazio.
GetServerTime	Recupera o timestamp atual da exchange de /api/v3/time .
GetExchangeInformation	Retorna as regras de negociação e os metadados de símbolo de /api/v3/exchangeInfo .
GetOrderBook	Baixa snapshots do order book de /api/v3/depth ; defina o parâmetro opcional limit para controlar o número de níveis (padrão 100).
GetTrades	Recupera negociações públicas recentes de /api/v3/trades com um limit opcional (padrão 100).
GetHistoricalTrades	Fornece dados históricos de negociação com filtros opcionais de limit e fromId .
GetAggregateTrades	Retorna negociações comprimidas/agregadas via /api/v3/aggTrades , suportando os filtros opcionais limit , fromId , startTime e endTime .
GetKlines	Baixa candlesticks de /api/v3/klines com suporte a filtros de intervalo de tempo e limite.
GetAveragePrice	Retorna o preço médio ponderado atual de /api/v3/avgPrice .
Get24hrTicker	Recupera as estatísticas do ticker de 24 horas para um ou todos os símbolos via /api/v3/ticker/24hr .
GetPriceTicker	Busca o preço mais recente utilizando /api/v3/ticker/price ; passe um símbolo vazio para recuperar os preços de todos os pares de negociação.
GetBookTicker	Obtém as melhores cotações de bid/ask de /api/v3/ticker/bookTicker com suporte a requisições de símbolo único ou de toda a exchange.

Endpoints REST privados

Os endpoints privados requerem API keys válidas e uma query string assinada. O componente anexa automaticamente o timestamp, o recvWindow e a assinatura HMAC.

Method	Descrição
GetAccountInformation	Retorna saldos e permissões de /api/v3/account .
GetOpenOrders	Lista as ordens abertas atuais (opcionalmente filtradas por símbolo).
GetAllOrders	Recupera ordens históricas com filtros opcionais de tempo e limite.
GetOrder	Consulta o status de uma ordem específica fornecendo orderId ou origClientOrderId .
GetMyTrades	Lista execuções privadas de negociação com os filtros opcionais limit , fromId , startTime e endTime .
NewOrder	Coloca uma ordem ao vivo em /api/v3/order (market, limit, stop, etc.) com timeInForce , quantity , quoteOrderQty , price , newClientOrderId , stopPrice , icebergQty opcionais e parâmetros extras.
TestNewOrder	Envia uma requisição apenas de validação para /api/v3/order/test aceitando o mesmo conjunto de parâmetros de NewOrder .

CancelOrder	Cancela uma ordem específica em /api/v3/order utilizando orderId ou origClientId .
CancelAllOrders	Cancela em massa todas as ordens abertas de um símbolo via /api/v3/openOrders .
GetSubAccounts	Lista as sub-contas gerenciadas (/api/v3/sub-account/list).
GetSubAccountAssets	Retorna os saldos de uma subconta específica.
TransferSubAccount	Transfere ativos entre subcontas por meio de /api/v3/sub-account/transfer ; forneça o valor e, opcionalmente, o type da transferência.
GetDepositAddress	Solicita endereços de depósito com um filtro opcional de network .
GetDepositHistory	Busca registros de depósito com filtros opcionais coin , status , startTime e endTime .
GetWithdrawHistory	Retorna o histórico de saques filtrado por coin , status e intervalo de tempo opcional.
Withdraw	Submete uma requisição de saque via /api/v3/capital/withdraw/apply incluindo os obrigatórios coin , address e amount além dos opcionais network , addressTag , withdrawOrderId e parâmetros extras.
BatchOrders	Envia ordens em batch para múltiplos símbolos em uma única requisição.
GetTradeFee	Obtém a taxa de comissão de negociação para um símbolo específico.
GetDepositHistory	Busca registros de depósito com filtros opcionais coin , status , startTime e endTime .
CancelWithdraw	Cancelar uma requisição de saque pendente pelo ID do saque.
CreateInternalTransfer	Transferir ativos entre contas internamente.
GetTransferHistory	Obtém o histórico de transferências internas entre contas.

API MEXC Futures

Os contratos perpétuos e de entrega da [MEXC](#) expõem uma API de streaming dedicada documentada na [especificação oficial Futures WebSocket](#). O componente **TsgcWSAPI_MEXC_Futures** encapsula a conexão, o handshake de login e o gerenciamento de tópicos necessários para consumir dados de derivativos em tempo real.

A MEXC também mantém uma interface REST para trading de derivativos. O componente **TsgcHTTP_API_MEXC_Futures** contido em **sgcHTTP_API_MEXC** complementa os feeds WebSocket com auxiliares HTTP que mapeiam um-para-um aos endpoints oficiais.

Configuração do componente

O endpoint WebSocket de Futures é **wss://contract.mexc.com/edge**. As mensagens são codificadas como objetos JSON e a exchange exige frames **ping/pong** periódicos, que são tratados automaticamente pelo componente.

Para receber notificações privadas (dados de ordem e de conta), você deve autenticar. Defina **MEXCAPI.ApiKey** e **MEXCAPI.ApiSecret** antes de ativar o cliente WebSocket. Quando as credenciais estão presentes, o componente assina uma requisição de login (HMAC SHA256) assim que o socket conecta.

```
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);
}
```

Properties

- **Client:** transporte WebSocket utilizado pelo componente da API. O suporte a Heartbeat é habilitado para manter o socket ativo.
- **MEXCAPI.ApiKey / ApiSecret:** necessários para assinar a requisição de login. Sem credenciais, apenas canais públicos são acessíveis.
- **RawMessages:** ignora o parser JSON quando habilitado, para que os quadros de texto bruto sejam expostos por meio dos eventos base.

Eventos

- **OnConnect / OnDisconnect:** notificações de ciclo de vida do WebSocket.
- **OnMEXCLogin:** gerado após um login autenticado bem-sucedido.
- **OnMEXCSubscribed / OnMEXCUnsubscribed:** confirmação de mudanças de subscrição. O evento expõe o nome do canal retornado pela MEXC.
- **OnMEXCMessage:** entrega cada atualização de mercado ou de conta junto com o identificador do canal.
- **OnMEXCError:** acionado quando o servidor responde com um payload **rs.error** (parâmetros inválidos, falha de autenticação, etc.).
- **OnMEXCException:** reporta exceções locais geradas ao processar dados JSON ou ao realizar operações de rede.

O seguinte manipulador registra confirmações de assinatura e imprime snapshots de profundidade recebidos:

Métodos de assinatura WebSocket

Cada helper envolve uma requisição **sub/unsub** conforme descrito pela API oficial. Utilize o método **UnSubscribe*** correspondente para cancelar o stream.

Method	Parâmetros	Descrição
SubscribeDeal	Symbol	Negociações executadas no contrato (canal: deal).
SubscribeTickers	-	Estatísticas globais de ticker para todos os contratos.
SubscribeTicker	Symbol	Resumo do ticker para um único instrumento.
SubscribeDepth	Symbol, Compress	Atualizações incrementais do order book
SubscribeDepthFull	Symbol, Level	Snapshots de profundidade completa com número configurável de níveis (padrão 20).
SubscribeKline	Symbol, Interval	Dados de candlestick para os timeframes suportados (Min1, Min5, Min15, Min30, Min60, Hour4, Hour12, Day1, Week1, Month1).
SubscribeFundingRate	Symbol	Anúncios mais recentes da taxa de funding.
SubscribeIndexPrice	Symbol	Stream subjacente do preço de índice.
SubscribeFairPrice	Symbol	Atualizações de preço Mark (justo) enviadas pela exchange.

Quando o servidor confirma uma requisição, o evento **OnMEXCSubscribed** é disparado com o nome do canal (por exemplo **rs.sub.depth.BTC_USDT**). Erros são encaminhados através de **OnMEXCError**, incluindo a mensagem do servidor para solução de problemas.

Canais privados

Os canais privados exigem uma API key válida. O componente autentica automaticamente quando as credenciais são fornecidas.

Method	Descrição
SubscribePersonalOrder	Atualizações de ordens pessoais.
SubscribePersonalOrderDeal	Atualizações de execução (fill) de ordens pessoais.
SubscribePersonalPosition	Atualizações de posição pessoal.
SubscribePersonalPlanOrder	Atualizações de ordem do plano pessoal (trigger).
SubscribePersonalStopOrder	Atualizações de stop orders pessoais.
SubscribePersonalStopPlanOrder	Atualizações de ordens de plano de stop pessoal.
SubscribePersonalRiskLimit	Atualizações do limite de risco pessoal.
SubscribePersonalADLLevel	Atualizações de nível pessoal de ADL (auto-deleveraging).
SubscribePersonalAsset	Atualizações de ativos pessoais.

Endpoints REST públicos

Métodos REST disponíveis publicamente.

Method	Descrição
GetPing	Verifica a acessibilidade da API via /api/v1/ping .
GetServerTime	Retorna o timestamp do servidor de /api/v1/time .

GetContracts	Fornece a lista de contratos disponíveis (/api/v1/contract/detail).
GetDepth	Baixa a profundidade do order book de /api/v1/contract/depth; use o parâmetro opcional limit (padrão 50) para selecionar o número de níveis.
GetDeals	Recupera as negociações recentes usando /api/v1/contract/deals com um limit opcional (padrão 100).
GetKlines	Retorna dados de candlestick por meio de /api/v1/contract/kline com filtros opcionais startTime , endTime e limit (padrão 200).
GetIndexPrice	Obtém o preço de índice subjacente para o símbolo solicitado (/api/v1/contract/indexPrice).
GetFairPrice	Retorna o fair (mark) price de um contrato de /api/v1/contract/fairPrice.
GetFundingRate	Reporta a taxa de funding mais recente de um símbolo utilizando /api/v1/contract/fundingRate.
GetAllTickers	Obtém todos os tickers de contratos de futuros com resumos de preços em tempo real.
GetFundingRateHistory	Obtém registros históricos de funding rate de um contrato.
GetFairPriceKline	Obtém dados de kline/candlestick do preço fair (mark) para um contrato.
GetIndexPriceKline	Obtém dados de kline/candlestick do preço de índice para um contrato.

Endpoints REST privados

Os endpoints privados de derivativos exigem credenciais de API. O componente assina cada requisição com o timestamp e o recvWindow automaticamente.

Method	Descrição
GetAccountAssets	Retorna os saldos de margin de /api/v1/private/account/assets.
GetPositionList	Lista as posições atuais via /api/v1/private/position/list, opcionalmente filtradas por símbolo.
SetPositionLeverage	Atualiza a alavancagem de um símbolo e, se fornecido, o modo de margem.
PlaceOrder	Coloca uma ordem de futures em /api/v1/private/order fornecendo symbol , side , positionSide , type e quantity além de price opcional, clientOrderId e parâmetros extras.
CancelOrder	Cancela uma ordem específica utilizando /api/v1/private/order/cancel por orderId ou clientOrderId .
CancelAllOrders	Cancela todas as ordens abertas de um símbolo via /api/v1/private/order/cancel-all.
GetOpenOrders	Lista as ordens abertas atuais através de /api/v1/private/order/list/open com um filtro de símbolo opcional.
GetOrderHistory	Recupera o histórico de ordens (/api/v1/private/order/list/history) com parâmetros opcionais de símbolo, intervalo de tempo e limite.
GetFundingHistory	Retorna pagamentos de funding passados de /api/v1/private/account/funding com os filtros opcionais startTime , endTime e limit .
GetOpenPositions	Obtém todas as posições atualmente abertas.
ChangeMargin	Altera o valor da margem de uma posição.
GetPositionMode	Obter o modo de posição atual (one-way ou hedge mode).
ChangePositionMode	Altera o modo de posição entre one-way e hedge mode.
PlaceBatchOrder	Coloca várias ordens de futuros em uma única requisição em lote.
GetOrderDetail	Obter informações detalhadas de uma ordem específica.
GetOrderDealDetails	Obtém os detalhes de fill/deal de uma ordem específica.

PlaceTriggerOrder	Coloca uma ordem trigger (plan) que é executada quando as condições são atendidas.
CancelTriggerOrder	Cancelar uma ordem de trigger específica.
CancelAllTriggerOrders	Cancelar todas as ordens trigger abertas.
GetTriggerOrders	Obter a lista de ordens de gatilho (plano).
GetStopOrders	Obtém a lista de ordens stop.
CancelStopOrder	Cancela uma ordem stop específica.
CancelAllStopOrders	Cancela todas as ordens stop abertas.

API Bitget

Bitget

APIs suportadas

- **WebSocket API:** conecta a um servidor websocket e fornece atualizações de dados de mercado em tempo real, alterações de conta e muito mais.
- **REST API:** envia requisições HTTP para obter dados de mercado, registrar ordens, dados de conta...

Properties

Use a propriedade **BitgetChannel** para selecionar o tipo de canal:

- **bgcSpot:** conecta à WebSocket API do Spot.
- **bgcFutures:** conecta à Futures WebSocket API.

Você pode configurar as seguintes propriedades na propriedade Bitget.

- **ApiKey:** você pode solicitar uma nova chave de api na sua conta Bitget, basta copiar o valor para esta propriedade. Se a APIKey for definida, o cliente se conectará ao servidor websocket privado. Se estiver vazia, conectará ao servidor WebSocket público.
- **ApiSecret:** é o valor secreto da api.
- **Passphrase:** é a string personalizada definida ao criar uma nova api key.

Conexão

Quando o cliente se conecta com sucesso aos servidores da Bitget, o evento **OnConnect** é disparado. Após o evento **OnConnect** ser disparado, você pode começar a **enviar e receber mensagens** de/para os servidores da Bitget. Se você estiver se conectando ao canal websocket privado, deve aguardar até que o evento **OnBitgetAuthentication** seja disparado e verificar se o parâmetro **success** é **true**, antes de assinar qualquer canal.

```
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");
}
```

Eventos

O cliente Bitget implementa os seguintes eventos para controlar o fluxo de conexão e obter os dados enviados pelo servidor WebSocket:

- **OnConnect:** disparado quando o cliente websocket conecta ao Servidor WebSocket.
- **OnDisconnect:** disparado quando o cliente websocket se desconecta do WebSocket Server.
- **OnBitgetAuthentication:** disparado quando o cliente se autentica contra o Servidor WebSocket Privado.
- **OnBitgetSubscribe:** quando o cliente assina um canal websocket.
- **OnBitgetUnSubscribe:** quando o cliente cancela a assinatura de um canal websocket.
- **OnBitgetData:** quando o cliente recebe dados do servidor.
- **OnBitgetError:** quando há algum erro durante a conexão websocket da Bitget.
- **OnBitgetHTTPException:** quando há algum erro durante a requisição REST.

WebSocket API

O feed websocket fornece atualizações de market data em tempo real para ordens e negociações.

Você pode se inscrever nos seguintes **canais**:

Method	Público ou privado	Descrição
SubscribeTicker	Public	Assine o stream de ticker para um instrumento.
SubscribeTrade	Public	Assine o stream de trades recentes.
SubscribeCandle	Public	Assine o stream de candlestick. Suporta múltiplos intervalos (por exemplo candle1m, candle5m, candle1H).
SubscribeOrderBook	Public	Assina o stream do order book. Suporta diferentes níveis de profundidade.
SubscribeOrders	Private	Assina atualizações de ordens. Requer autenticação.
SubscribePositions	Private	Assina atualizações de posição. Requer autenticação.
SubscribeAccount	Private	Assinar atualizações de saldo da conta. Requer autenticação.

Veja abaixo um exemplo de assinatura de canais websocket privados após uma autenticação bem-sucedida.

```
void OnBitgetAuthentication(TObject *Sender, bool aSuccess, const string aError, const string aRawMessage)
{
    if (aSuccess == true)
    {
        oBitget->SubscribeOrders("BTCUSD");
        oBitget->SubscribeAccount();
    }
}
```

REST API

A REST API fornece métodos Públicos e Privados para solicitar dados de mercados e de contas privadas. Acesse a REST API através da propriedade **REST_API** do componente.

Method	Público / Privado	Descrição
GetServerTime	Public	Obter o horário do servidor.
GetTickers	Public	Obtém informações de ticker para um ou todos os símbolos.
GetOrderBook	Public	Obtém o livro de ordens de um símbolo.
GetCandles	Public	Obtém dados de candlestick/kline para um símbolo.
GetRecentTrades	Public	Obter as negociações mais recentes de um símbolo.
PlaceOrder	Private	Coloca uma nova ordem.
CancelOrder	Private	Cancela uma ordem existente.
GetOpenOrders	Private	Obter a lista de ordens abertas.
GetOrderDetail	Private	Obter detalhes de uma ordem específica.
GetOrderHistory	Private	Obtém o histórico de ordens.
GetAccountAssets	Private	Obtém as informações de ativos da conta.
GetAccountInfo	Private	Obtém as informações da conta.

Encontre abaixo um exemplo de uso da 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("BTCUSD", "buy", "market", "normal", "0.01");
```

API GateIO

[Gate.io](https://gate.io)

APIs suportadas

- **WebSocket API:** conecta a um servidor websocket e fornece atualizações de dados de mercado em tempo real, alterações de conta e muito mais.
- **REST API:** envia requisições HTTP para obter dados de mercado, registrar ordens, dados de conta...

Properties

Utilize a propriedade **GateIOChannel** para selecionar o tipo de canal:

- **giocSpot:** conecta à Spot WebSocket API.
- **giocFutures:** conecta-se à Futures WebSocket API.

Você pode configurar as seguintes propriedades na propriedade GateIO.

- **ApiKey:** você pode requisitar uma nova api key em sua conta da Gate.io; basta copiar o valor para esta propriedade. Se a APIKey for definida, o cliente se autenticará contra o servidor privado. Se estiver vazia, apenas canais públicos estarão disponíveis.
- **ApiSecret:** é o valor secreto da api.

Conexão

Quando o cliente se conecta com sucesso aos servidores da Gate.io, o evento **OnConnect** é disparado. Após o evento **OnConnect** ser disparado, você pode começar a **enviar e receber mensagens** de/para os servidores da Gate.io. Se você estiver se conectando ao canal websocket privado, deve aguardar até que o evento **OnGateIO-Authentication** seja disparado e verificar se o parâmetro success é true, antes de assinar qualquer canal privado.

```
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");
}
```

Eventos

O cliente GateIO implementa os seguintes eventos para controlar o fluxo da conexão e obter os dados enviados pelo servidor WebSocket:

- **OnConnect:** disparado quando o cliente websocket conecta ao Servidor WebSocket.
- **OnDisconnect:** disparado quando o cliente websocket se desconecta do WebSocket Server.
- **OnGateIOAuthentication:** disparado quando o cliente se autentica contra o Servidor WebSocket Privado.
- **OnGateIOSubscribe:** quando o cliente assina um canal websocket.
- **OnGateIOUnSubscribe:** quando o cliente faz unsubscribe de um canal websocket.
- **OnGateIOData:** quando o cliente recebe dados do servidor.
- **OnGateIOError:** quando há algum erro durante a conexão websocket da GateIO.
- **OnGateIOHTTPException:** quando há qualquer erro durante a requisição REST.

WebSocket API

O feed websocket fornece atualizações de market data em tempo real para ordens e negociações. Você pode se inscrever nos seguintes **canais**:

Method	Público ou privado	Descrição
SubscribeTicker	Public	Inscreve-se no stream de ticker de um símbolo.
SubscribeTrades	Public	Assine o stream de trades recentes.
SubscribeCandlesticks	Public	Assine o stream de candlestick. Suporta múltiplos intervalos (por exemplo 1m, 5m, 15m, 30m, 1h).
SubscribeOrderBook	Public	Assina o stream do livro de ordens. Suporta níveis de profundidade configuráveis e intervalos de atualização.
SubscribeOrders	Private	Assina atualizações de ordens. Requer autenticação.
SubscribeBalances	Private	Assinar atualizações de saldo. Requer autenticação.

Veja abaixo um exemplo de assinatura de canais websocket públicos.

```
void OnConnect(TsgcWSConnection *Connection)
{
    oGateIO->SubscribeTicker("BTC_USDT");
    oGateIO->SubscribeOrderBook("BTC_USDT", "20", "100ms");
}
```

REST API

A REST API fornece métodos Públicos e Privados para solicitar dados de mercados e de contas privadas. Acesse a REST API através da propriedade **REST_API** do componente.

Method	Público / Privado	Descrição
GetCurrencyPairs	Public	Obter informações sobre os pares de moedas disponíveis.
GetTickers	Public	Obtém informações de ticker de um ou de todos os pares de moedas.
GetOrderBook	Public	Obter o order book de um par de moedas.
GetCandlesticks	Public	Obtém dados de candlestick/kline para um par de moedas.
GetTrades	Public	Obtém os trades mais recentes de um par de moedas.
PlaceOrder	Private	Coloca uma nova ordem. Suporta ordens limit e market.
CancelOrder	Private	Cancela uma ordem existente.
GetOrder	Private	Obter detalhes de uma ordem específica.
GetOpenOrders	Private	Obter a lista de ordens abertas.
GetSpotAccounts	Private	Obter informações da conta spot.

Encontre abaixo um exemplo de uso da 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 é uma exchange de derivativos de criptomoedas que oferece negociação de futuros e opções em Bitcoin e Ethereum.

APIs suportadas

- **WebSocket API:** conecta a um servidor websocket e fornece atualizações de dados de mercado em tempo real, alterações de conta e muito mais.
- **REST API:** envia requisições HTTP para obter dados de mercado, registrar ordens, dados de conta...

Properties

Você pode configurar as seguintes propriedades na propriedade Deribit.

- **ApiKey:** você pode solicitar uma nova api key na sua conta Deribit, basta copiar o valor para esta propriedade. Se a APIKey estiver definida, o cliente se autenticará contra o servidor privado. Se estiver vazia, apenas os canais públicos estarão disponíveis.
- **ApiSecret:** é o valor secreto da api.
- **TestNet:** se habilitado, conectará ao ambiente TestNet da Deribit (desabilitado por padrão). Útil para testes sem arriscar fundos reais.

Conexão

Quando o cliente se conecta com sucesso aos servidores da Deribit, o evento **OnConnect** é disparado. Depois que o evento **OnConnect** é disparado, você pode começar a **enviar e receber mensagens** de/para os servidores da Deribit. Se você estiver se conectando ao canal websocket privado, deve aguardar até que o evento **OnDeribitAuthentication** seja disparado e verificar se o parâmetro **success** é **true**, antes de assinar qualquer canal privado.

```
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");
}
```

Eventos

O cliente Deribit implementa os seguintes eventos para controlar o fluxo de conexão e obter os dados enviados pelo servidor WebSocket:

- **OnConnect:** disparado quando o cliente websocket conecta ao Servidor WebSocket.
- **OnDisconnect:** disparado quando o cliente websocket se desconecta do WebSocket Server.
- **OnDeribitAuthentication:** disparado quando o cliente se autentica no Servidor WebSocket Privado.
- **OnDeribitSubscribe:** quando o cliente se inscreve em um canal websocket.
- **OnDeribitUnSubscribe:** quando o cliente cancela a assinatura de um canal websocket.
- **OnDeribitData:** quando o cliente recebe dados do servidor.
- **OnDeribitError:** quando há algum erro durante a conexão websocket Deribit.
- **OnDeribitHTTPException:** quando há algum erro durante a requisição REST.

WebSocket API

O feed websocket fornece atualizações de market data em tempo real para ordens e negociações.

Você pode assinar os seguintes **canais públicos**:

Method	Descrição
SubscribeTicker	Assina o stream de ticker de um instrumento. Fornece dados de preço e volume em tempo real.
SubscribeTrades	Inscribe-se no stream de negociações recentes de um instrumento.
SubscribeOrderBook	Inscribe-se no stream do order book. Suporta agrupamento configurável, níveis de profundidade e intervalos de atualização.
SubscribeCandle	Assina o stream de candlestick. Suporta múltiplas resoluções (por exemplo 1, 3, 5, 10, 15, 30, 60, 120, 180, 360, 720, 1D).
SubscribePerpetual	Assina os dados de contrato perpétuo de um instrumento. Fornece funding rate e outras informações específicas de perpétuos.
SubscribeBookChange	Assine as notificações de alteração do order book de um instrumento.

Você pode assinar os seguintes **canais privados** (requerem autenticação):

Method	Descrição
SubscribeOrders	Assina atualizações de ordens para um instrumento.
SubscribePositions	Inscribe-se em atualizações de posição. Suporta filtragem por moeda e tipo de instrumento.
SubscribeAccount-Changes	Assine as alterações de saldo da conta para uma moeda.
SubscribeUserTrades	Assina atualizações de negociações do usuário para um instrumento.

Encontre abaixo um exemplo de inscrição em canais websocket.

```
void OnDeribitAuthentication(TObject *Sender, bool aSuccess, const string aError, const string aRawMessage)
{
    if (aSuccess == true)
    {
        oDeribit->SubscribeOrders("BTC-PERPETUAL");
        oDeribit->SubscribePositions("BTC");
    }
}
```

REST API

A REST API fornece métodos Públicos e Privados para solicitar dados de mercados, negociação, conta e carteira. Acesse a REST API por meio da propriedade **REST_API** do componente.

Market Data (Public)

Method	Descrição
GetInstruments	Obtém os instrumentos disponíveis para uma moeda.
GetTicker	Obtém informações do ticker para um instrumento.
GetOrderBook	Obter o order book de um instrumento.
GetTrades	Obtém as negociações mais recentes de um instrumento.
GetCurrencies	Obtém a lista de moedas suportadas.
GetIndexPrice	Obtém o preço de índice atual para um nome de índice.
GetFundingRateHistory	Obtém o histórico da taxa de funding de um instrumento.
GetFundingRateValue	Obtém o valor atual da funding rate para um instrumento.
GetBookSummaryBy-Currency	Obtém o resumo do book para todos os instrumentos de uma moeda.
GetBookSummaryByInstrument	Obtém o resumo do book para um instrumento específico.

Trading (Private)

Method	Descrição
Buy	Coloca uma ordem de compra. Suporta os tipos de ordem market e limit.
Sell	Coloca uma ordem de venda. Suporta tipos de ordem a mercado e limitada.
CancelOrder	Cancela uma ordem existente pelo ID da ordem.
CancelAllOrders	Cancela todas as ordens abertas.
CancelAllByInstrument	Cancela todas as ordens abertas de um instrumento específico.
EditOrder	Edita uma ordem existente (altera o valor ou o preço).
GetOpenOrders	Obtém a lista de ordens abertas para uma moeda.
GetOrderState	Obtém o estado de uma ordem específica.
GetOrderHistory	Obter o histórico de ordens de uma moeda.

Account (Private)

Method	Descrição
GetPositions	Obtém as posições de uma moeda.
GetAccountSummary	Obtém o resumo da conta para uma moeda.
GetSubAccounts	Obter a lista de subcontas.
GetTransactionLog	Obtém o log de transações de uma moeda.

Wallet (Privada)

Method	Descrição
GetDeposits	Obtém o histórico de depósitos de uma moeda.
GetWithdrawals	Obtém o histórico de saques de uma moeda.
GetTransfers	Obter o histórico de transferências de uma moeda.

Encontre abaixo um exemplo de uso da 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](#)

APIs suportadas

- **WebSocket API:** conecta a um servidor websocket e fornece atualizações de dados de mercado em tempo real, alterações de conta e muito mais.
- **REST API:** envia requisições HTTP para obter dados de mercado, registrar ordens, dados de conta...

Properties

Utilize a propriedade **Channel** para selecionar o tipo de canal:

- **cccMarket:** conecta à API WebSocket de Mercado para dados públicos de mercado.
- **cccUser:** conecta-se à User WebSocket API para dados privados da conta (requer autenticação).

Você pode configurar as seguintes propriedades na propriedade CryptoCom.

- **ApiKey:** você pode solicitar uma nova chave de api na sua conta Crypto.com, basta copiar o valor para esta propriedade. Obrigatório para canais privados e chamadas autenticadas à REST API.
- **ApiSecret:** é o valor secreto da api.

Conexão

Quando o cliente conecta com sucesso aos servidores Crypto.com, o evento **OnConnect** é disparado. Após o evento **OnConnect** ser disparado, você pode começar a **enviar e receber mensagens** de/para os servidores Crypto.com. Se você está conectando ao canal User, deve aguardar até que o evento **OnCryptoComAuthentication** seja disparado e verificar se o parâmetro **success** é **true**, antes de assinar qualquer canal privado.

```
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");
}
```

Eventos

O cliente Crypto.com implementa os seguintes eventos para controlar o fluxo de conexão e obter os dados enviados pelo servidor WebSocket:

- **OnConnect:** disparado quando o cliente websocket conecta ao Servidor WebSocket.
- **OnDisconnect:** disparado quando o cliente websocket se desconecta do WebSocket Server.
- **OnCryptoComAuthentication:** disparado quando o cliente se autentica contra o Servidor WebSocket Privado.
- **OnCryptoComSubscribe:** quando o cliente assina um canal websocket.
- **OnCryptoComUnSubscribe:** quando o cliente cancela a assinatura de um canal websocket.
- **OnCryptoComData:** quando o cliente recebe dados do servidor.
- **OnCryptoComError:** quando há algum erro durante a conexão websocket da Crypto.com.
- **OnCryptoComHTTPException:** quando ocorre algum erro durante a requisição REST.

WebSocket API

O feed websocket fornece atualizações de market data em tempo real para ordens e negociações. Você pode assinar os seguintes **canais de Mercado** (públicos):

Method	Descrição

SubscribeTicker	Assina o stream de ticker de um instrumento. Fornece dados de preço e volume em tempo real.
SubscribeTrade	Inscribe-se no stream de negociações recentes de um instrumento.
SubscribeCandlestick	Assina o stream de candlesticks. Suporta múltiplos intervalos (por exemplo, 5m, 15m, 30m, 1h, 4h, 1D).
SubscribeBook	Assina o stream do livro de ofertas. Suporta níveis de profundidade configuráveis (por exemplo, 10, 50).

Você pode assinar os seguintes **canais de Usuário** (privados, requerem autenticação):

Method	Descrição
SubscribeOrders	Assina atualizações de ordens. Opcionalmente filtra por nome do instrumento.
SubscribeUserTrades	Assine as atualizações de negociações do usuário. Opcionalmente filtre por nome do instrumento.
SubscribeBalance	Inscribe-se nas atualizações de saldo da sua conta.

Veja abaixo um exemplo de assinatura de canais de mercado.

```
void OnConnect(TsgcWSConnection *Connection)
{
    oCryptoCom->SubscribeTicker("BTC_USDT");
    oCryptoCom->SubscribeBook("BTC_USDT", "10");
}
```

REST API

A REST API fornece métodos Públicos e Privados para solicitar dados de mercados e de contas privadas. Acesse a REST API através da propriedade **REST_API** do componente.

Method	Público / Privado	Descrição
GetInstruments	Public	Obtém a lista de instrumentos disponíveis.
GetTicker	Public	Obtém informações de ticker para um ou todos os instrumentos.
GetOrderBook	Public	Obter o order book de um instrumento.
GetTrades	Public	Obtém as negociações mais recentes de um instrumento.
GetCandlestick	Public	Obtém dados de candlestick/kline para um instrumento.
CreateOrder	Private	Cria uma nova ordem. Suporta ordens limitadas e a mercado.
CancelOrder	Private	Cancela uma ordem existente.
CancelAllOrders	Private	Cancela todas as ordens abertas de um instrumento.
GetOpenOrders	Private	Obter a lista de ordens abertas.
GetOrderDetail	Private	Obter detalhes de uma ordem específica.
GetOrderHistory	Private	Obtém o histórico de ordens.
GetAccountSummary	Private	Obter informações de resumo da conta.

Encontre abaixo um exemplo de uso da 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

A HTX (anteriormente Huobi) é uma exchange de criptomoedas internacional e multilíngue. O componente da HTX API fornece uma HTTP REST API para complementar a Huobi WebSocket API existente.

APIs suportadas

- **WebSocket API:** conecta a um servidor websocket e fornece atualizações de dados de mercado em tempo real, alterações de conta e mais. Consulte a documentação da Huobi API para os métodos de assinatura WebSocket.
- **REST API:** envia requisições HTTP para obter dados de mercado, registrar ordens, dados de conta...

Properties

Você pode configurar as seguintes propriedades na propriedade Huobi (para WebSocket) ou na propriedade HTXOptions (para REST API).

- **ApiKey:** você pode solicitar uma nova api key na sua conta HTX, basta copiar o valor para esta propriedade. Se a ApiKey estiver definida, o cliente se conectará ao servidor websocket privado. Se estiver vazia, conectará ao servidor WebSocket público.
- **ApiSecret:** é o valor secreto da api.

Conexão

O componente HTX estende o cliente da WebSocket API da Huobi. Quando o cliente se conecta com sucesso aos servidores HTX, o evento **OnConnect** é disparado. Se a ApiKey não estiver vazia, o cliente tentará conectar ao servidor websocket privado, de modo que apenas os métodos privados estarão disponíveis. Se a ApiKey estiver vazia, o cliente conectará ao servidor websocket público e apenas os métodos públicos estarão disponíveis.

```
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");
}
```

Eventos

O cliente HTX implementa os seguintes eventos para controlar o fluxo de conexão e obter os dados enviados pelo servidor WebSocket:

- **OnConnect:** disparado quando o cliente websocket conecta ao Servidor WebSocket.
- **OnDisconnect:** disparado quando o cliente websocket se desconecta do WebSocket Server.
- **OnHuobiSubscribed:** evento chamado após uma assinatura bem-sucedida.
- **OnHuobiUnSubscribed:** evento chamado após um cancelamento de assinatura bem-sucedido.
- **OnHuobiUpdate:** toda vez que há uma atualização nos dados (kline, market depth...) este evento é chamado.
- **OnHuobiError:** se houver um erro na API HTX, este evento fornecerá informações sobre o erro.

WebSocket API

A WebSocket API fornece atualizações de dados de mercado em tempo real. Consulte a documentação da Huobi API para a lista completa de métodos de assinatura WebSocket, incluindo canais públicos (SubscribeKLine, SubscribeMarketDepth, SubscribeTradeDetail, etc.) e canais privados (SubscribeOrderUpdates, SubscribeTradeClearing, SubscribeAccountChange).

REST API

A REST API fornece métodos Públicos e Privados para solicitar dados de mercados e contas privadas. A REST API é acessada através do componente **TsgcHTTP_API_HTX**.

Market Data (Public)

Method	Descrição
GetServerTime	Obter o horário do servidor.
GetSymbols	Obtém a lista de símbolos de negociação disponíveis.
GetCurrencys	Obtém a lista de moedas suportadas.
GetMarketTickers	Obter tickers para todos os símbolos de negociação.
GetMarketDetail	Obtém os detalhes de mercado de 24 horas para um símbolo.
GetMarketDepth	Obtém a profundidade do livro de ordens para um símbolo. Suporta tipo de profundidade e níveis de profundidade configuráveis.
GetMarketTrade	Obtém a negociação mais recente de um símbolo.
GetMarketHistoryTrade	Obtém o histórico de negociações mais recente de um símbolo.
GetMarketHistoryKline	Obter dados de candlestick/kline. Suporta múltiplos períodos (1min, 5min, 15min, 30min, 60min, 4hour, 1day, 1mon, 1week, 1year).

Trading (Private)

Method	Descrição
PlaceOrder	Coloca uma nova ordem. Suporta buy-market, sell-market, buy-limit, sell-limit e outros tipos de ordem.
CancelOrder	Cancela uma ordem existente pelo ID da ordem.
GetOrder	Obter detalhes de uma ordem específica.
GetOpenOrders	Obter a lista de ordens abertas de uma conta.
GetOrderHistory	Obtém o histórico de ordens de um símbolo.

Account (Private)

Method	Descrição
GetAccounts	Obtém a lista de contas.
GetAccountBalance	Obtém o saldo de uma conta específica.
GetAccountAssetValuation	Obtém a avaliação total dos ativos para um tipo de conta (por exemplo, spot).

Encontre abaixo um exemplo de uso da 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

Envie e receba mensagens usando uma versão hospedada em nuvem da **WhatsApp Business Platform**. A **Cloud API** permite que você implemente APIs do WhatsApp Business sem o custo de hospedar seus próprios servidores e também permite escalar mais facilmente as mensagens do seu negócio. A Cloud API suporta até 80 mensagens por segundo de envio e recebimento combinados (incluindo mensagens de texto e de mídia).

A WhatsApp Business API permite que empresas de médio e grande porte se comuniquem com seus clientes em escala. Utilizando a API, as empresas podem construir sistemas que conectam milhares de clientes com agentes ou bots, habilitando tanto a comunicação programática quanto a manual. Além disso, você pode integrar a API a inúmeros sistemas de backend, como plataformas de CRM e de marketing.

Features

As empresas receberão todos os novos recursos mais rapidamente na Cloud API. Atualmente, a WhatsApp Business Cloud API vem com todos os recursos disponíveis na WhatsApp Business API.

Recursos úteis da WhatsApp Cloud API:

- **Integre** a mensageria do WhatsApp com ferramentas como **CRM**, **analytics** e apps de **terceiros**
- **Green Tick**, perfil verificado do WhatsApp Business
- WhatsApp **Broadcast & Mensagens em Massa**
- Sem app ou interface, use via BSPs ou CRM
- **Chatbot de WhatsApp & automação** de chat usando aplicativos de terceiros
- **Agende** mensagens WhatsApp em larga escala
- Os recursos de **mensagens interativas** incluem mensagens de Lista, botões de resposta e mensagens CTA

Most common uses

- **Configuração**
 - [WhatsApp Create App](#)
 - [WhatsApp Phone Number Id](#)
 - [WhatsApp Token](#)
 - [WhatsApp Webhook](#)
 - [WhatsApp Security](#)
- **Messages**
 - [WhatsApp Send Messages](#)
 - [WhatsApp Send Interactive Messages](#)
 - [WhatsApp Send Template Messages](#)
 - [WhatsApp Receive Messages and Status Notifications](#)
 - [WhatsApp Send Files](#)
 - [WhatsApp Download Media](#)

Get Started

Para enviar e receber uma primeira mensagem usando um número de teste, complete as seguintes etapas:

1. Configurar os Developer Assets e o Platform Access

- [Registre-se como Meta Developer](#)
- [Habilite a autenticação de dois fatores para sua conta](#)

- **Criar um Meta App:** Acesse developers.facebook.com > **My Apps** > **Create App**. Selecione o tipo "Business" e siga as instruções na sua tela.

No App Dashboard, clique no app que você gostaria de conectar ao WhatsApp. Role para baixo para encontrar o produto "WhatsApp" e clique em **Set up**.

Em seguida, você verá a opção de selecionar um Business Manager existente (se você tiver um) ou, se preferir, o processo de onboarding pode criar um automaticamente para você (você pode personalizar seu negócio depois, se necessário). Faça uma seleção e clique em **Continue**.

Quando você clica em **Continue**, o processo de onboarding realiza as seguintes ações:

- Seu App está associado ao Business Manager que você escolheu ou que foi criado automaticamente.
- Um número de telefone de teste do WhatsApp é adicionado ao seu negócio. Você pode utilizar este número de telefone de teste para explorar a WhatsApp Business Platform sem registrar ou migrar um número de telefone real. Os números de telefone de teste podem enviar mensagens ilimitadas para até 5 destinatários (que podem estar em qualquer lugar do mundo).

2. Enviar uma Mensagem de Teste

Agora você pode abrir sua IDE e criar um novo projeto. Coloque um componente `TsgcWhatsapp_Client` e preencha as seguintes propriedades:

- **WhatsappOptions.PhoneNumberId:** é o ID do Número de Telefone utilizado para enviar mensagens.
- **WhatsappOptions.Token:** é o Temporary Access Token válido por 24 horas.

Uma vez que essas 2 propriedades tenham sido devidamente configuradas, chame o método **SendTest** para enviar sua **Primeira mensagem** para um número de telefone usando a **WhatsApp Business Platform**.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendTest("34605889421");
```

3. Configurar um Webhook

Para ser alertado quando você recebe uma mensagem ou quando o status de uma mensagem mudou, você precisa configurar um endpoint de Webhooks para sua aplicação. Configurar os Webhooks não afeta o status do seu número de telefone e não interfere no envio ou recebimento de mensagens.

Para começar, primeiro você precisa criar o endpoint, então primeiro configure a propriedade **ServerOptions** do componente WhatsApp Client e configure as seguintes propriedades:

- **ServerOptions:** aqui você pode configurar o IP Address para vincular, a Listening Port, se está usando SSL (o WebHook deve rodar em um servidor seguro; você pode configurar seu servidor para usar SSL ou fazer proxy das requisições do WebHook para um servidor não HTTPs). O servidor é baseado em [TsgcWebSocketsHTTPServer](#).
 - **WebhookOptions:** esta propriedade permite que você defina as propriedades do Webhook que posteriormente serão configuradas em sua conta de desenvolvedor do facebook.
 - **Endpoint:** é o nome do endpoint, por padrão é /webhook. Exemplo: se seu servidor está escutando em <https://www.esegece.com>, o endpoint será "https://www.esegece.com/webhook"
 - **Token:** é uma string de segurança que pode conter qualquer valor definido por você. É usada para verificar se o registro do Webhook está correto.

Após configurar o servidor, você pode utilizar o método **StartServer** para iniciar o servidor e aceitar as requisições de entrada.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->ServerOptions->WebhookOptions->PhoneNumberId = "/webhook";
oClient->ServerOptions->WebhookOptions->Token = "MySecretToken";
oClient->StartServer();
```

Uma vez que seu endpoint esteja pronto, vá para o App Dashboard.

No Dashboard do seu App, encontre o produto WhatsApp e clique em **Configuration**. Em seguida, encontre a seção de webhooks e clique em **Configure a webhook**. Após o clique, um diálogo aparece na sua tela e solicita dois itens:

- **Callback URL**: Esta é a URL para a qual a Meta enviará os eventos.
- **Verify Token**: Esta string é configurada por você, ao criar o endpoint do seu webhook.

Após adicionar as informações, clique em **Verify and Save**.

De volta ao App Dashboard, clique em **WhatsApp > Configuration** no painel à esquerda. Em Webhooks, clique em **Manage**. Uma caixa de diálogo será aberta com todos os objetos sobre os quais você pode ser notificado. Para receber mensagens dos seus usuários, clique em **Subscribe** para **messages**.

4. Receber uma mensagem de teste

Toda vez que uma nova mensagem é recebida, o evento de cliente **OnMessageReceived** será chamado.

```
void OnMessageReceived(TObject *Sender, TsgcWhatsapp_Receive_Message *aMessage, ref bool aMarkAsRead)
{
    DoLog("Received: " + aMessage->Messages->_Message[0]->Id);
}
```

Agora que seu Webhook está configurado, envie uma mensagem ao número de teste que você utilizou. Você deve receber imediatamente uma notificação de Webhooks com o conteúdo da sua mensagem!

A WhatsApp API não permite enviar mensagens de texto livre para telefones que não entraram em contato com você antes (nas últimas 24 horas). A única maneira de enviar uma mensagem de texto a um telefone que nunca enviou mensagem para o número da sua conta de desenvolvedor é enviando um Template (previamente aprovado pela Meta). Para contornar essa limitação ao testar mensagens de texto livre, primeiro envie uma mensagem de WhatsApp do número de destino para o número da sua conta de desenvolvedor, e então você poderá enviar mensagens de texto livre por 24 horas.

Eventos

OnBeforeSendMessage

Este evento é chamado antes de a mensagem ser enviada aos servidores do WhatsApp. Você pode acessar a mensagem interna por meio do parâmetro `RawMessage`.

OnBeforeSubscribe

Este evento é chamado antes de o servidor assinar um tópico. Use o parâmetro `Accept` para permitir ou negar a assinatura. Por padrão, o servidor assinará todos os eventos solicitados.

OnRawMessage

Este evento é chamado quando o servidor recebe uma nova mensagem que ainda não foi analisada, então você obtém acesso à mensagem bruta.

OnMessageReceived

Este evento é chamado depois que o servidor recebe e faz o parsing de uma nova mensagem. Se você definir o parâmetro `MarkAsRead` como `True`, o remetente receberá uma confirmação dupla (double check).

OnMessageSent

Este evento é chamado toda vez que o servidor recebe uma nova mensagem de status sobre a mensagem enviada anteriormente. Leia a propriedade `Status` para saber se a mensagem foi enviada, entregue ou lida.

WhatsApp Create App

Acesse developers.facebook.com e **Crie um App**.

Selecione **Business Type** como o tipo de app e prossiga.

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

Forneça um nome para seu app (evite usar nomes registrados como marcas, como “WhatsApp” ou “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

Uma vez criado o app, clique no **botão WhatsApp** na próxima tela para adicionar recursos de envio do WhatsApp ao seu app.



Na próxima tela, será solicitado que você vincule seu aplicativo WhatsApp à sua conta comercial do Facebook. Você também terá a opção de criar uma nova conta comercial caso ainda não tenha uma.

WhatsApp Phone Number Id

Quando você se registra na WhatsApp Cloud API, o Facebook fornece um número de telefone de teste do WhatsApp que será o endereço de envio padrão da sua aplicação. Para os destinatários, você terá a opção de adicionar um máximo de 5 números de telefone durante a fase de desenvolvimento sem precisar fazer nenhum pagamento.

Mais tarde, você pode registrar seu próprio Número de Telefone para evitar a limitação de 5 números de telefone.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();  
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
```

WhatsApp Token

A WhatsApp Cloud API requer um token válido para enviar qualquer mensagem usando a Cloud API.

O Facebook fornece um número de telefone de Teste do WhatsApp que permite enviar mensagens para até 5 números de telefone. Você pode posteriormente substituir essa limitação registrando seu próprio número de telefone.

A API do WhatsApp fornece um **Token de Acesso Temporário** que será válido por 23 horas. Este token deve ser configurado no componente `TsgcWhatsApp_Client` para permitir o envio de mensagens.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();  
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
```

Se você precisar de um token de validade longa, pode criar (ou atualizar) um usuário de sistema e gerar um novo token com a permissão **whatsapp_business_messaging**. Isso permitirá que você envie e receba mensagens do WhatsApp sem atualizar o token a cada 23 horas.

WhatsApp Webhook

Assine os Webhooks para obter notificações sobre as mensagens que sua empresa recebe e as atualizações de perfil de cliente.

Create Endpoint

Antes de poder começar a receber notificações, você precisará criar um endpoint no seu servidor para receber notificações.

Seu endpoint deve ser capaz de processar dois tipos de requisições HTTPS: Verification Requests e Event Notifications. Como ambas as requisições utilizam HTTPS, seu servidor deve ter um certificado TLS ou SSL válido corretamente configurado e instalado. Certificados autoassinados não são suportados.

Ao configurar o Webhook nas Configurações do WhatsApp, você deve definir o endpoint onde seu servidor está escutando e um Token que pode ser qualquer valor; este token é utilizado ao registrar o endpoint do webhook e ao verificar se o assinante é válido.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->ServerOptions->WebhookOptions->PhoneNumberId = "/webhook";
oClient->ServerOptions->WebhookOptions->Token = "MySecretToken";
oClient->StartServer();
```

Uma vez configurado o Webhook, assine os Webhook Fields de **Messages** para ser notificado toda vez que uma nova mensagem for recebida.

Você pode ler mais sobre como configurar o [SSL Server](#).

WhatsApp Security

Toda vez que uma nova mensagem é recebida ou há um novo status de uma mensagem, o servidor recebe uma notificação no endpoint configurado no [Webhook](#). Para garantir que a requisição vem dos Servidores da WhatsApp Cloud API, a requisição contém um cabeçalho com uma assinatura; você pode configurar o cliente WhatsApp para verificar as assinaturas antes de processar a mensagem.

Para fazer isso, primeiro você precisa definir o `Application Secret` na propriedade **`ServerOptions.Application.Secret`** e habilitar a propriedade **`VerifySignature`**.

Uma vez configurado, toda vez que uma nova mensagem é recebida, primeiro a assinatura é verificada e, se estiver errada, retorna um erro 500 e a mensagem não é processada.

WhatsApp Send Messages

Todas as chamadas de API devem ser autenticadas com um Access Token. Os desenvolvedores podem autenticar suas chamadas de API com o access token gerado em **App Dashboard > WhatsApp > Getting Started**

As chamadas de API retornam o Message Id como uma string.

Text Messages

Chame o método **SendMessageText** e passe os seguintes parâmetros:

- **aTo:** número de telefone
- **aText:** texto da mensagem.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendMessageText("34605889421", "Hello from sgcWebSockets!!!");
```

Image Messages

Chame o método **SendMessageImage** e passe os seguintes parâmetros:

- **aTo:** número de telefone
- **aLink:** url onde está a imagem a enviar
- **aCaption:** título da imagem (opcional).

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendMessageImage("34605889421", "https://www.media.com/image.png", "logo");
```

Document Messages

Chame o método **SendMessageDocument** e passe os seguintes parâmetros:

- **aTo:** número de telefone
- **aLink:** url onde está o documento a enviar
- **aCaption:** título do documento (opcional).
- **aFileName:** nome do arquivo (opcional).

```
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");
```

Audio Messages

Chame o método **SendMessageAudio** e passe os seguintes parâmetros:

- **aTo:** número de telefone
- **aLink:** url onde está o áudio a enviar

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendMessageAudio("34605889421", "https://www.audio.com/audio.mp3");
```

Mensagens de Vídeo

Chame o método **SendMessageVideo** e passe os seguintes parâmetros:

- **aTo:** número de telefone
- **aLink:** url onde está o vídeo a ser enviado

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendMessageVideo("34605889421", "https://www.video.com/audio.mp4");
```

Mensagens de Sticker

Chame o método **SendMessageSticker** e passe os seguintes parâmetros:

- **aTo:** número de telefone
- **aLink:** url onde está o sticker a enviar

```
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

Chame o método **SendMessageLocation** e passe os seguintes parâmetros:

- **aTo:** número de telefone
- **aLongitude:** Longitude da localização.
- **aLatitude:** Latitude da localização.
- **aName:** Nome da localização.
- **aAddress:** Endereço da localização. Exibido apenas se aName for definido.

```
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");
```


Contact Messages

Chame o método **SendMessageContact** e passe os seguintes parâmetros:

- **aTo:** número de telefone
- **aName:** Nome completo, como normalmente aparece (obrigatório).
- **aPhone:** o número de telefone (opcional).
- **aEmail:** o e-mail (opcional).

```
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 Enviar mensagens interativas

As mensagens interativas dão aos seus usuários uma forma mais simples de encontrar e selecionar o que desejam do seu negócio no WhatsApp. Durante os testes, chatbots usando recursos de mensagens interativas alcançaram taxas de resposta e conversões significativamente mais altas em comparação com os baseados em texto.

As seguintes mensagens são consideradas interativas:

- **List Messages:** Mensagens que incluem um menu de até 10 opções. Este tipo de mensagem oferece uma forma mais simples e consistente para os usuários fazerem uma seleção ao interagir com uma empresa.
- **Reply Buttons:** Mensagens que incluem até 3 opções —cada opção é um botão. Este tipo de mensagem oferece uma forma mais rápida para os usuários fazerem uma seleção a partir de um menu ao interagir com uma empresa. Os reply buttons têm a mesma experiência de usuário que os templates interativos com botões.

Interactive Message Specifications

- Mensagens interativas podem ser combinadas no mesmo fluxo.
- Os usuários não podem selecionar mais de uma opção ao mesmo tempo de uma lista ou mensagem com botões, mas podem voltar e reabrir uma mensagem anterior.
- Mensagens de botão de lista ou de resposta não podem ser usadas como notificações. Atualmente, elas só podem ser enviadas dentro de 24 horas da última mensagem enviada pelo usuário. Se você tentar enviar uma mensagem fora da janela de 24 horas, receberá uma mensagem de erro.

Quando Você Deve Usá-lo

As List Messages são ideais para apresentar várias opções, como:

- Um menu de atendimento ao cliente ou FAQ
- Um cardápio para viagem
- Seleção de lojas ou locais próximos
- Horários de reserva disponíveis
- Escolhendo uma ordem recente para repetir

Os Reply Buttons são ideais para oferecer respostas rápidas a partir de um conjunto limitado de opções, como:

- Recarga de crédito de celular
- Alterando dados pessoais
- Reordenando uma ordem anterior
- Solicitando uma devolução
- Adicionar extras opcionais a um pedido de comida
- Escolhendo um método de pagamento

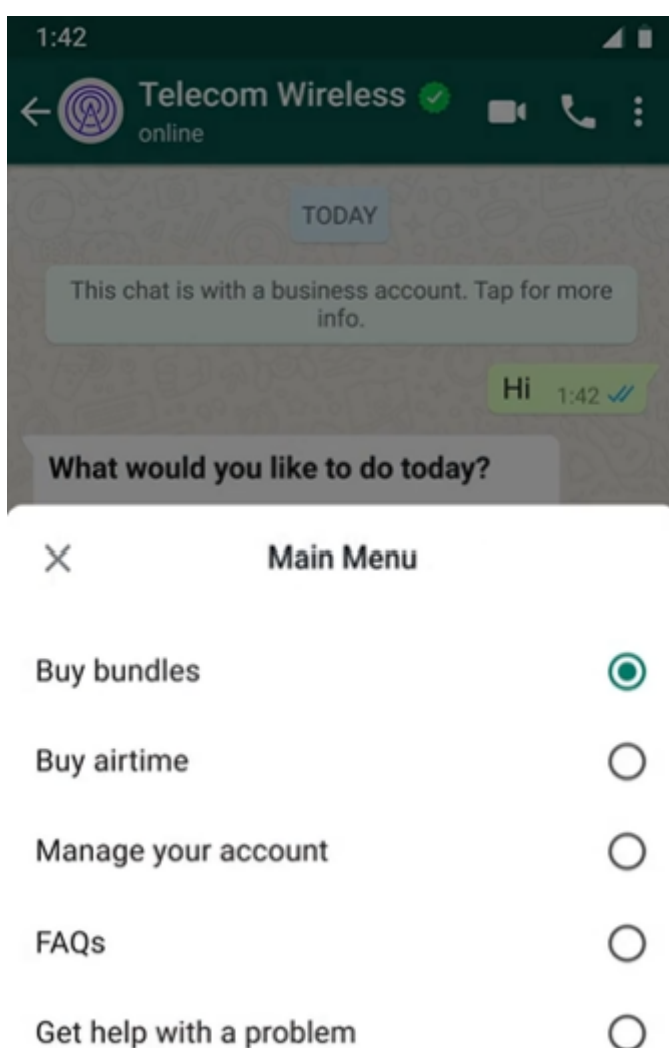
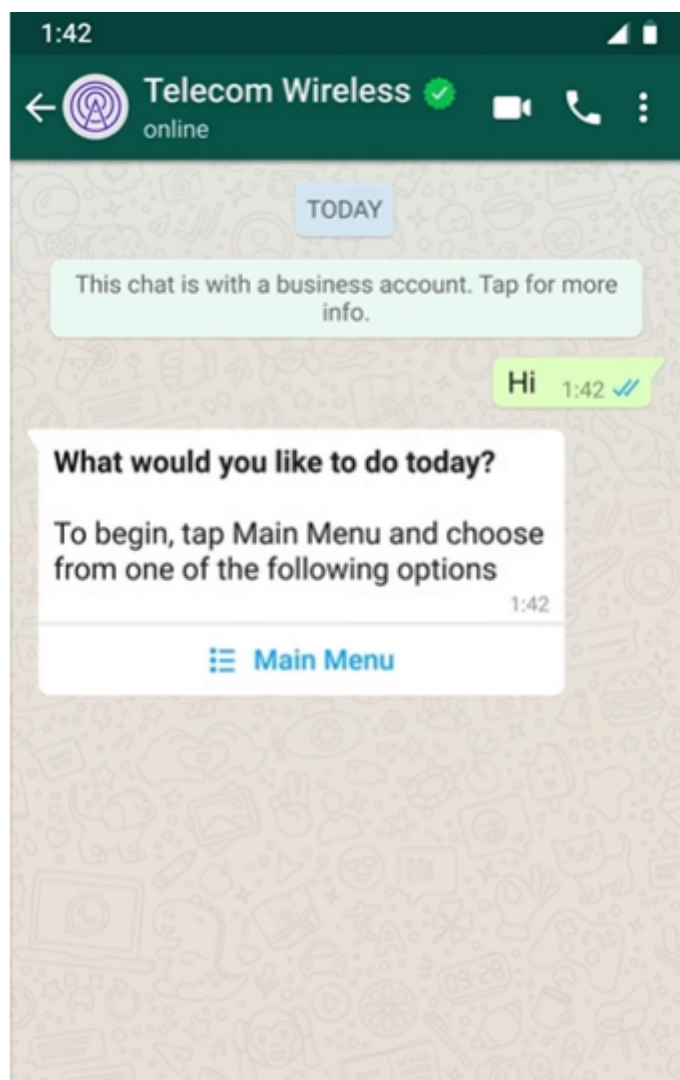
Os botões de resposta são particularmente valiosos para casos de uso 'personalizados' em que uma resposta genérica não é adequada.

Interactive List

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendMessageInteractiveList("
```

34605889421

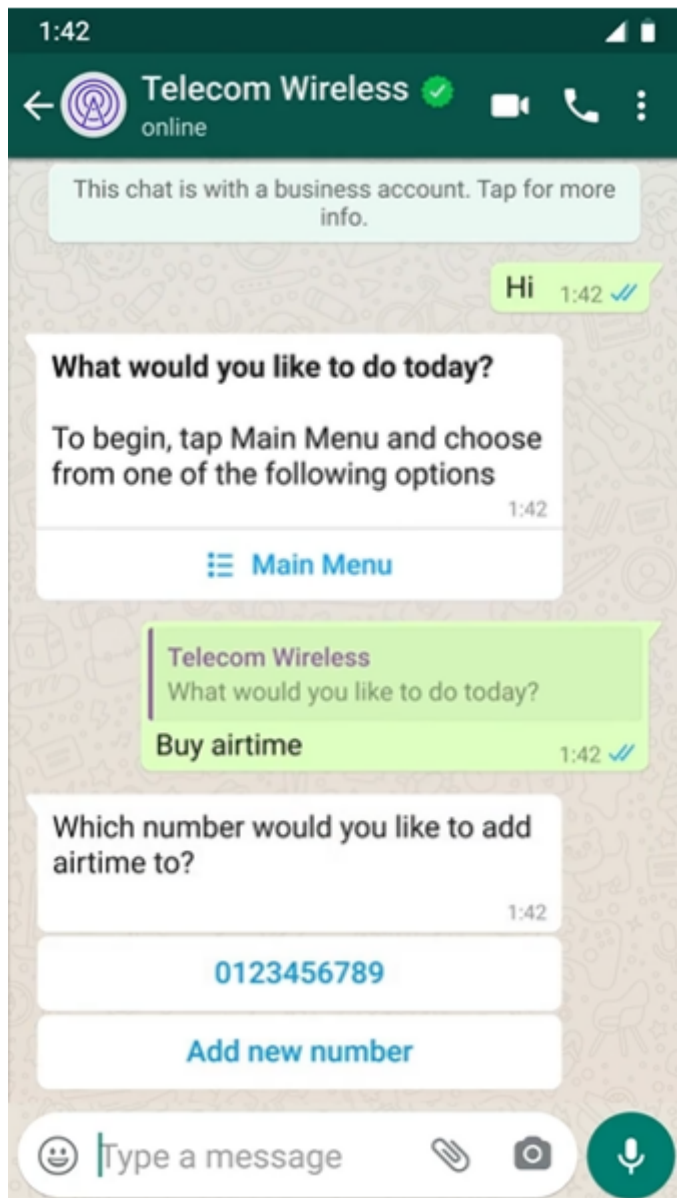
", "What Would you like to do today?", "To begin, Tap Main Menu and choose from of the following options", "", "M

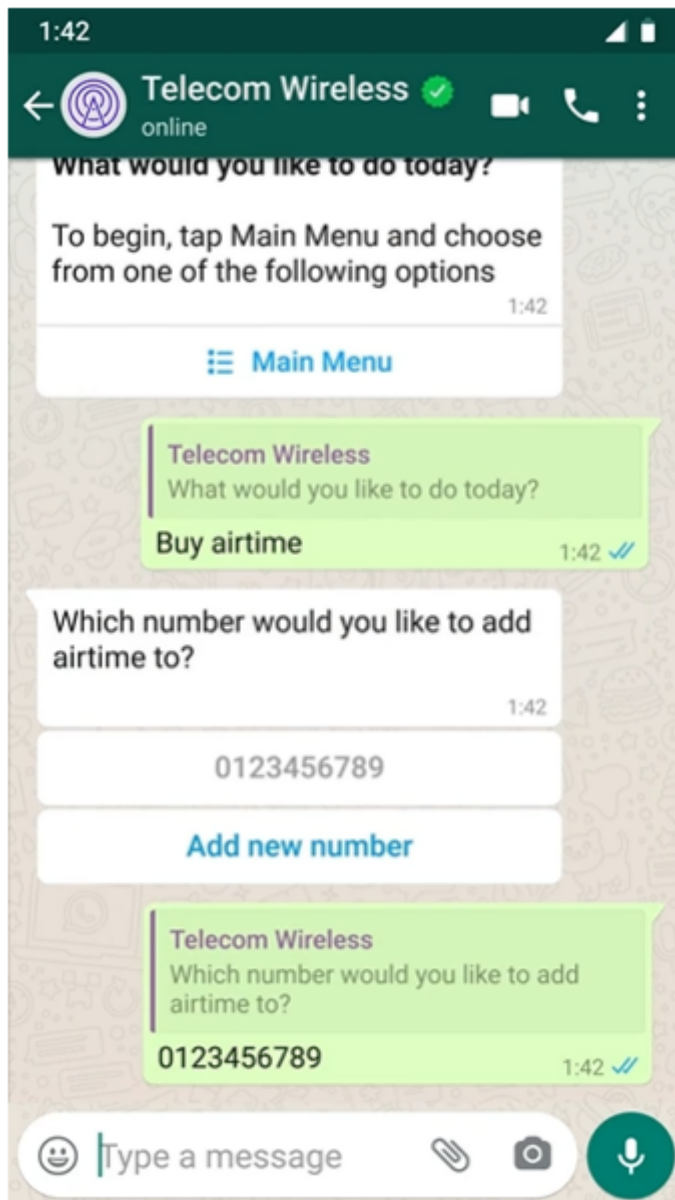


Reply Buttons

```
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 air
```





WhatsApp Send Template Messages

Chame o método **SendMessageTemplate** e passe os seguintes parâmetros:

- **aTo:** número de telefone
- **aTemplate:** identificador do template.
- **aLanguageCode:** idioma do template.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendMessageTemplate("34605889421", "hello_world", "en_US");
```

Parâmetros de Mensagem de Template

Templates podem incluir parâmetros, veja abaixo um exemplo de template padrão com parâmetros

```
void SendSamplePurchaseFeedbackTemplate(string aName)
{
    TsgcWhatsApp_Send_Message_Template *oTemplate = new TsgcWhatsApp_Send_Message_Template();
    Try
    {
        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 *oComponent = new TsgcWhatsApp_Send_Message_Template_Component();
        oComponent->_Type = wapctBody;
        oTemplate->Components->Add(oComponent);
        TsgcWhatsApp_Send_Message_Template_Parameter *oParameter = new TsgcWhatsApp_Send_Message_Template_Parameter();
        oParameter->Text = aName;
        oParameter->_Type = wapptText;
        oComponent->Parameters->Add(oParameter);
        whatsapp->SendMessageTemplate("107809351952205", oTemplate);
    }
    Finally
    {
        oTemplate->Free();
    }
    End;
}
```

Template Message Uploaded Image

Veja abaixo um exemplo de um template onde, em vez de utilizar um link para uma imagem, primeiro carrega a imagem no servidor e então define o Id do documento.

```
oClient->OnMessage(TsgcWSConnection *Connection, const string Text)
void SendSamplePurchaseFeedbackTemplate(string aName)
{
    TsgcWhatsApp_Send_Message_Template *oTemplate = new TsgcWhatsApp_Send_Message_Template();
    Try
    {
        // ... first upload the file
        string vId = whatsapp->UploadMedia("c:\\images\\file.png", "image/png");

        // ... send message
        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 *oComponent = new TsgcWhatsApp_Send_Message_Template_Component()
oComponent->_Type = wapctBody;
oTemplate->Components->Add(oComponent);
TsgcWhatsApp_Send_Message_Template_Parameter *oParameter = new TsgcWhatsApp_Send_Message_Template_Parameter()
oParameter->Text = aName;
oParameter->_Type = wapptText;
oComponent->Parameters->Add(oParameter);
whatsapp->SendMessageTemplate("107809351952205", oTemplate);
__finally
oTemplate->Free();
End;
}

```


WhatsApp Receive Messages and Status Notifications

Inscreve-se em [Webhooks](#) para receber notificações sobre as mensagens que seu negócio recebe e sobre atualizações de perfil de clientes.

Sempre que ocorre um evento de gatilho, a WhatsApp Business Platform percebe o evento e envia uma notificação para uma URL de Webhook que você especificou anteriormente. Você pode obter dois tipos de notificações:

- **Mensagens recebidas:** Este alerta avisa quando você recebeu uma mensagem.
- **Notificações de status e preços de mensagens:** Este alerta informa quando o status de uma mensagem mudou, por exemplo, a mensagem foi lida ou entregue.

Received Messages

Toda vez que uma nova mensagem é recebida, o evento **OnMessageReceived** é chamado, onde você pode acessar o conteúdo da Message e marcá-la como lida.

Encontre abaixo um exemplo em que, quando uma nova mensagem de texto é recebida, ela é ecoada para o usuário que a enviou.

```
void OnWhatsAppMessageReceived(TObject *Sender, const TsgcWhatsApp_Receive_Message *aMessage, ref bool aMarkAsRead)
{
    if (aMessage->Contacts->Count > 0)
    {
        string vTo = aMessage->Contacts->Contact[0]->WaID;
        if (aMessage->Messages->Count > 0)
        {
            if (aMessage->Messages->_Message[0]->_Type = wapmrtText)
            {
                vText = "ECHO ==> " + aMessage->Messages->_Message[0]->Text->Body;
                WhatsApp->SendMessageText(vTo, vText);
                aMarkAsRead = true;
            }
        }
    }
}
```

Mensagens Enviadas

A WhatsApp Business Platform envia notificações para informá-lo sobre o status das mensagens entre você e os usuários. Quando uma mensagem é enviada com sucesso, você recebe uma notificação quando a mensagem é enviada, entregue e lida. A ordem dessas notificações no seu app pode não refletir o tempo real do status da mensagem. Veja o timestamp para determinar o tempo, se necessário.

- **sent:** A notificação a seguir é recebida quando uma empresa envia uma mensagem como parte de uma conversa iniciada pelo usuário (se essa conversa não se originou em um free entry point):
- **delivered:** A seguinte notificação é recebida quando a mensagem de uma empresa é entregue e essa mensagem faz parte de uma conversa iniciada pelo usuário (se essa conversa não se originou em um ponto de entrada gratuito):
- **read:** A notificação a seguir é recebida quando o usuário lê a mensagem.

Cada vez que um novo status é recebido, o evento **OnMessageSent** é chamado.

```
void OnWhatsAppMessageSent(TObject *Sender, const TsgcWhatsApp_Receive_Message *aMessage, TsgcWhatsAppSendMessage
{
    string vPhone := aMessage.MetaData.DisplayPhoneNumber
    if (aStatus = wapsmstSent) {
        DoLog("Message to " + vPhone + " sent.");
    } else if (aStatus = wapsmstDelivered) {
        DoLog("Message to " + vPhone + " delivered.");
    } else if (aStatus = wapsmstRead) {
        DoLog("Message to " + vPhone + " read.");
    } else {
        DoLog("Message to " + vPhone + " unknown status.");
    }
}
```

WhatsApp Send Files

Todas as chamadas de API devem ser autenticadas com um Access Token. Os desenvolvedores podem autenticar suas chamadas de API com o access token gerado em **App Dashboard > WhatsApp > Getting Started**

As chamadas de API retornam o Message Id como uma string.

Quando você envia um arquivo utilizando a API do WhatsApp, primeiro a mensagem é enviada para os servidores do WhatsApp e, em seguida, uma nova mensagem é enviada com o id do objeto retornado após o upload do arquivo.

Image Messages

Chame o método **SendMessageImage** e passe os seguintes parâmetros:

- **aTo:** número de telefone
- **aFileName:** nome completo do arquivo (com caminho) do arquivo de imagem a enviar.
- **aFileType:**
 - image/jpeg
 - image/png
- **aCaption:** título da imagem (opcional).

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendFileImage("34605889421", "c:\\images\\image.png", "image/png");
```

Document Messages

Chame o método **SendMessageDocument** e passe os seguintes parâmetros:

- **aTo:** número de telefone
- **aFileName:** nome completo do arquivo (com caminho) do arquivo de documento a enviar.
- **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:** título do documento (opcional).

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendFileDocument("34605889421", "c:\\MyDocuments\\invoice.pdf", "application/pdf");
```

Audio Messages

Chame o método **SendMessageAudio** e passe os seguintes parâmetros:

- **aTo:** número de telefone
- **aFileName:** nome completo do arquivo (com caminho) do arquivo de áudio a enviar.
- **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");
```

Mensagens de Vídeo

Chame o método **SendMessageVideo** e passe os seguintes parâmetros:

- **aTo:** número de telefone
- **aFileName:** nome completo do arquivo (com caminho) do arquivo de vídeo a enviar.
- **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");
```

Mensagens de Sticker

Chame o método **SendMessageSticker** e passe os seguintes parâmetros:

- **aTo:** número de telefone
- **aFileName:** nome de arquivo completo (com caminho) do arquivo de sticker a enviar.
- **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 você receber uma mensagem com um link de arquivo de mídia, pode baixar o arquivo de mídia usando o método **DownloadMedia**.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFG...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->DownloadMedia("38923878928822", "c:\\whatsapp\\media\\image.png");
```

Para excluir um arquivo de mídia previamente enviado, basta chamar **DeleteMedia** e passar o id do objeto como argumento.

API Telegram

Telegram

O Telegram oferece dois tipos de APIs. Uma é a **Bot API**, que permite criar programas que usam Bots e HTTPs como protocolo. A **Telegram API e TDLib** permitem que você construa clientes Telegram personalizados e é muito mais poderosa que a Bot API.

O `sgcWebSockets` **suporta o TDLib através da biblioteca `tdjson`**, o que significa que você pode construir seu próprio cliente Telegram. O TDLib cuida de todos os detalhes de implementação de rede, criptografia e armazenamento local de dados. O TDLib suporta todos os recursos do Telegram.

TDLib (Telegram Database Library) Advantages

- **Multiplataforma:** pode ser usado em Windows, Android, iOS, MacOS, Linux...
- **Fácil de usar:** usa mensagens json para se comunicar entre a aplicação e o telegram.
- **Alto desempenho:** Na Telegram Bot API, cada instância TDLib trata mais de 24000 bots.
- **Consistent:** o TDLib garante que todas as atualizações serão entregues na ordem correta.
- **Reliable:** O TDLib permanece estável em conexões de internet lentas e não confiáveis.
- **Secure:** Todos os dados locais são criptografados utilizando uma chave de criptografia fornecida pelo usuário.
- **Totalmente Assíncrono:** As requisições ao TDLib não bloqueiam umas às outras. As respostas serão enviadas quando estiverem disponíveis.

Configuração

Windows

O TDLib requer outras bibliotecas de terceiros: OpenSSL e ZLib. Essas bibliotecas devem ser implantadas com a biblioteca `tdjson`.

* As versões para Windows requerem o VCRuntime, que pode ser baixado da Microsoft: <https://www.microsoft.com/en-us/download/details.aspx?id=52685>, Se após a instalação o problema persistir, tente copiar as seguintes dll na mesma pasta onde sua aplicação está localizada: `VCRUNTIME140.dll` e `VCRUNTIME140_1.dll`.

Copie as seguintes bibliotecas para o mesmo diretório onde sua aplicação está localizada:

Windows 32	Windows 64
<code>tdjson.dll</code>	<code>tdjson.dll</code>
<code>libcrypto-1_1.dll</code>	<code>libcrypto-1_1-x64.dll</code>
<code>libssl-1_1.dll</code>	<code>libssl-1_1-x64.dll</code>
<code>zlib1.dll</code>	<code>zlib1.dll</code>

OSX64

Implante a biblioteca `libtdjson.dylib` no seu dispositivo e você pode definir onde a biblioteca está utilizando `SetTDJ-sonPath`, exemplo:

se você implantar em "Contents\MacOS", deve definir o caminho na pasta `TPath.GetDirectoryName(ParamStr(0))`.

OSXARM64

Implante a biblioteca `libtdjson.dylib` no seu dispositivo e você pode definir onde a biblioteca está utilizando `SetTDJ-sonPath`, exemplo:

se você implantar em "Contents\MacOS", deve definir o caminho na pasta `TPath.GetDirectoryName(ParamStr(0))`.

Linux64

Implante a biblioteca `libtdjson.so` no seu dispositivo e defina o caminho da biblioteca chamando o método `SetTDJsonPath`.

Android

Implante a biblioteca `libtdjsonandroid.so` no seu dispositivo. Exemplo: se você implantar uma biblioteca Android64, defina `RemotePath` em `Project/Deployment` como `"library\lib\arm64-v8a"`. Se for Android32, defina `RemotePath` como `"library\lib\armeabi-v7a"`

iOS64

Copie a biblioteca `libtdjson.a` para estes diretórios:

- C:\Program Files (x86)\Embarcadero\Studio\<IDE Version>\lib\iosDevice64\debug
- C:\Program Files (x86)\Embarcadero\Studio\<IDE Version>\lib\iosDevice64\release

Criando sua aplicação Telegram

Para obter um API id e desenvolver sua própria aplicação usando a Telegram API, você precisa fazer o seguinte:

- Cadastre-se no Telegram utilizando qualquer aplicação.
- Faça login no seu Telegram core: <https://my.telegram.org>.
- Vá para **API development tools** e preencha o formulário.
- Você obterá os endereços básicos, bem como os parâmetros **api_id** e **api_hash** necessários para a autorização do usuário.
- Por enquanto, cada número só pode ter um **api_id** conectado a ele.

Esses valores devem ser definidos na propriedade **Telegram.API** do componente Telegram. Para autenticar, você pode autenticar como usuário ou como bot; há 2 propriedades que você pode definir para fazer login no Telegram:

- **PhoneNumber:** se você fizer login como usuário, deverá definir seu **número de telefone** (com código internacional), exemplo: +34699123456
- **BotToken:** se você fizer login como um bot, defina seu token nesta propriedade (conforme fornecido pelo telegram).
- **DatabaseDirectory:** permite especificar onde está o banco de dados do `tdlib`. Deixe vazio e a configuração padrão será utilizada.

Os seguintes parâmetros podem ser configurados:

- **ApplicationVersion:** versão da aplicação, exemplo: 1.0
- **DeviceModel:** modelo do dispositivo, exemplo: desktop
- **LanguageCode:** código de idioma do usuário, exemplo: en.
- **SystemVersion:** versão do sistema operacional, exemplo: windows.

Opcionalmente, você pode configurar o caminho onde a biblioteca `tdjson` está localizada utilizando o método **SetTDJsonPath**. Basta passar o caminho antes de iniciar uma nova sessão telegram.

Uma vez que você tenha configurado o Componente Telegram, pode definir a propriedade `Active` como `true` e o programa tentará conectar ao Telegram.

Código de Exemplo

```
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;
```

Autorização

Há dois eventos que podem ser chamados pela biblioteca para obter um Código de Autenticação (entregue no aplicativo do Telegram, não por SMS) ou para fornecer uma senha.

OnAuthenticationCode

Este evento é chamado quando o Telegram envia um Código de Autorização para a Aplicação Telegram e o usuário deve copiar esse código e definir no argumento Code deste evento.

```
void OnAuthenticationCode(TObject *Sender, ref string Code)
{
    Code = InputBox("Telegram Code", "Introduce code", "");
}
```

OnAuthenticationPassword

Este evento é chamado quando o Telegram exige que o usuário defina uma senha.

Status de Autorização

Uma vez iniciada a autorização, você pode verificar o status da autorização no evento **OnAuthorizationStatus**; este evento é chamado toda vez que há uma mudança no status da autorização. Alguns valores de Status são:

- authorizationStateWaitTdlbParameters
- authorizationStateWaitEncryptionKey
- authorizationStateWaitPhoneNumber
- authorizationStateWaitCode
- authorizationStateLoggingOut
- authorizationStateClosed
- authorizationStateReady

Connection Status

Uma vez que a conexão tenha começado, você pode verificar o status da conexão no evento **OnConnectionStatus**; este evento é chamado toda vez que há uma mudança no status da conexão. Alguns valores de Status são:

- connectionStateConnecting
- connectionStateUpdating
- connectionStateReady

Métodos Principais

O Componente da API TsgcTDLb_Telegram suporta vários métodos do Telegram, encontre abaixo os mais usados.

Method	Parâmetros	Descrição

SendText-Message	aChatId: Id do Chat para o qual a mensagem será enviada aText: Texto da Mensagem. InlineKeyboard: Botões opcionais (somente bots).	Envia uma Mensagem de Texto para um Chat
SendRich-TextMessage	aChatId: Id do Chat para o qual a mensagem será enviada aText: Texto da Mensagem. InlineKeyboard: Botões opcionais (somente bots).	Envia uma mensagem de Rich Text para um Chat. Sintaxe Markdown: • Negrito: **bold** • Itálico: <i>__italic__</i> • Tachado: ~~--strike--~~ • Sublinhado: <u>~~underline~~</u> • Code: <code>##code##</code>
SendDocumentMessage	aChatId: Id do Chat ao qual a mensagem será enviada aFilePath: caminho completo do arquivo do documento InlineKeyboard: Botões opcionais (apenas bots).	Envia um Documento a um Chat.
SendPhoto-Message	aChatId: Id do Chat para o qual a mensagem será enviada aFilePath: caminho completo do arquivo da foto Width: largura da foto. Height: largura da foto. InlineKeyboard: Botões opcionais (somente bots).	Envia uma Foto para um Chat.
SendVideo-Message	aChatId: Id do Chat para o qual a mensagem será enviada aFilePath: caminho completo do arquivo de vídeo aWidth: largura do vídeo. Height: largura do vídeo. aDuration: duração do vídeo em segundos. InlineKeyboard: Botões opcionais (somente bots).	Envia um Vídeo a um Chat.
SendInvoiceMessage	aChatId: Id do Chat para o qual a mensagem será enviada invoice: Texto da Mensagem. InlineKeyboard: Botões opcionais (somente bots).	Envia uma Invoice (disponível apenas quando é um Bot e em Canais Privados).
EditText-Message	aChatId: Id do Chat ao qual a mensagem será enviada aMessageId: Id da Mensagem a modificar Text: Texto da Mensagem. InlineKeyboard: Botões opcionais (somente bots). ShowKeyboard: Botões opcionais (apenas bots).	Edita o texto de uma mensagem (ou o texto de uma mensagem de jogo)
AddChat-Member	aChatId: Id do Chat ao qual a mensagem será enviada aUserId: Identificador do usuário. aForwardLimit: O número de mensagens anteriores do chat a serem encaminhadas ao novo membro; até 100. Ignorado para supergrupos e canais.	Adiciona um novo membro a um chat. Membros não podem ser adicionados a chats privados ou secretos. Os membros não serão adicionados até que o estado do chat tenha sido sincronizado com o servidor.
AddChat-Members	aChatId: Id do Chat para o qual a mensagem será enviada aUserIds: Identificadores dos usuários a serem adicionados ao chat.	Adiciona múltiplos novos membros a um chat. Atualmente esta opção só está disponível para supergrupos e canais. Esta opção não pode ser utilizada para entrar em um chat. Os membros não po-

		dem ser adicionados a um canal se ele tiver mais de 200 membros. Os membros não serão adicionados até que o estado do chat tenha sido sincronizado com o servidor.
GetChat-Member	aChatId: Identificador do Chat. aUserId: Identificador do Usuário.	Retorna informações sobre um único membro de um chat.
GetBasic-GroupFullInfo	aGroupId: Basic Group Identifier	Retorna informações completas sobre um grupo básico pelo seu identificador.
GetSuper-groupMembers	aSuperGroupId: Identificador do supergrupo ou canal. aSupergroupMembersFilter: O tipo de usuários a serem retornados. Por padrão null aOffset: Número de usuários a serem ignorados. aLimit: O número máximo de usuários a serem retornados; até 200.	Retorna informações sobre membros ou usuários banidos em um supergrupo ou canal.
Join-ChatByInviteLink	aLink: Invite link a importar;	Usa um link de convite para adicionar o usuário atual ao chat, se possível. O novo membro não será adicionado até que o estado do chat tenha sido sincronizado com o servidor.
CreateNew-SecretChat	aUserId: Identificador do usuário.	Cria um novo secret chat.
CreateNew-BasicGroup-Chat	aUserIds: Identificadores dos usuários a serem adicionados ao chat. aTitle: Título do novo grupo básico	Cria um novo grupo básico
CreateNew-Super-groupChat	aTitle: Título do novo SuperGrupo alsChannel: True, se um chat de canal deve ser criado. aDescription: Descrição do Chat.	Cria um novo supergrupo ou canal.
CreatePrivateChat	aUserId: Identificador do usuário. aForce: Se true, o chat será criado sem requisição de rede. Neste caso, todas as informações sobre o chat, exceto seu tipo, título e foto, podem estar incorretas	Retorna um chat existente correspondente a um determinado usuário
GetChats	aOffsetOrder: Ordem dos chats a partir da qual retornar os chats aOffsetChatId: Identificador do chat a partir do qual retornar os chats aLimit: O número máximo de chats a serem retornados.	Retorna uma lista ordenada de chats. Os chats são ordenados pelo par (order, chat_id) em ordem decrescente (não pode ser utilizado quando logado como Bot)
GetChat	aChatId: Identificador do chat	Retorna informações sobre um chat pelo seu identificador
GetChatHistory	aChatId: Identificador do Chat aFromMessageId: Identificador da mensagem a partir da qual o histórico deve ser buscado; use 0 para obter resultados a partir da última mensagem. aOffset: Especifique 0 para obter resultados a partir exatamente do from_message_id ou um offset negativo de até 99 para obter adicionalmente algumas mensagens mais recentes.	Retorna as mensagens em um chat. As mensagens são retornadas em ordem cronológica inversa

	aLimit: O número máximo de mensagens a serem retornadas	
GetUser	aUserId: Identificador do Usuário	Retorna informações sobre um usuário pelo seu identificador.
Add-ProxyHTTP	aServer: Nome do servidor do proxy. aPort: Número da porta do proxy. aUserName: Nome de usuário para fazer login; pode estar vazio. aPassword: Senha para fazer login; pode ser vazia. aHTTPOnly: Passe true, se o proxy suportar apenas requisições HTTP e não suportar conexões TCP transparentes via método HTTP CONNECT.	Adiciona um servidor proxy HTTP para requisições de rede. Pode ser chamado antes da autorização.
Add-ProxyMT-Proto	aServer: Nome do servidor do proxy. aPort: Número da porta do proxy. aSecret: O segredo do proxy em codificação hexadecimal.	Adiciona um servidor proxy MT-Proto para requisições de rede. Pode ser chamado antes da autorização.
AddProxy-Socks5	aServer: Nome do servidor do proxy. aPort: Número da porta do proxy. aUserName: Nome de usuário para fazer login; pode estar vazio. aPassword: Senha para login; pode estar vazia.	Adiciona um proxy Socks5 para requisições de rede. Pode ser chamado antes da autorização.
Enable-Proxy	aid: ID do proxy	Habilita um proxy. Apenas um proxy pode estar habilitado por vez. Pode ser chamado antes da autorização.
Disable-Proxy		Desabilita o proxy atualmente habilitado. Pode ser chamado antes da autorização.
Remove-Proxy	aid: ID do proxy	Remove um servidor proxy. Pode ser chamado antes da autorização.
GetProxies		Retorna a lista de proxies que estão atualmente configurados. Pode ser chamado antes da autorização.
getChatSponsored-Message	aChatId: ID do chat	Retorna a mensagem patrocinada a ser exibida em um chat; apenas para chats de canal. Retorna um erro 404 se não houver mensagem patrocinada no chat.
ViewMessage	aSponsorChatId: ID do Chat patrocinador aMessageId: ID da mensagem	Informa ao TDLib que as mensagens estão sendo visualizadas pelo usuário. Muitas atividades úteis dependem de se as mensagens estão sendo visualizadas no momento ou não
Logout		Faz logout do Telegram.
TDLibSend	aRequest: Requisição JSON.	Envia qualquer Requisição no protocolo JSON.

Example How to send a Text Message

```
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");
```

Exemplo Como enviar um método não implementado

Você pode enviar qualquer mensagem JSON utilizando o método TDLibSend, exemplo: entrar em um chat do 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\"}");
```

Eventos

OnBeforeReadEvent

Este evento é chamado quando uma mensagem JSON é recebida pelo componente da Telegram API e ainda não foi processada. Defina a propriedade Handled como True se você processar este evento manualmente ou não quiser que o evento seja processado pelo componente. Você também pode utilizar este evento para registrar todas as mensagens.

OnMessageText

Este evento é chamado quando um Novo Texto de Mensagem foi recebido, leia o parâmetro MessageText para acessar as propriedades do texto da mensagem.

- **ChatId:** Identificador do Chat.
- **MessageId:** Identificador da Mensagem.
- **SenderId:** Identificador do Remetente.
- **Text:** Texto da mensagem.

OnMessageDocument

Este evento é chamado quando uma Nova Mensagem de Documento é recebida. Acesse MessageDocument para ter acesso às propriedades do Document.

- **ChatId:** Identificador do Chat.
- **MessageId:** Identificador da Mensagem.
- **SenderId:** Identificador do Remetente (leia SenderChat e SenderUser a partir do tdlb 1.7.+).
- **FileName:** Nome do Documento.
- **DocumentId:** Identificador do Documento.
- **LocalPath:** caminho completo para o arquivo local, se existir.
- **MimeType:** Mime-type do documento.
- **Size:** Tamanho do Documento.
- **RemoteDocumentId:** Identificador do Documento Remoto.

OnMessagePhoto

Este evento é chamado quando uma Nova Mensagem com Foto é recebida. Acesse MessagePhoto para obter acesso às propriedades da Foto.

- **ChatId:** Identificador do Chat.
- **MessageId:** Identificador da Mensagem.
- **SenderId:** Identificador do Remetente (leia SenderChat e SenderUser a partir do tdlb 1.7.+).
- **Photoid:** Identificador da Foto.
- **LocalPath:** caminho completo para o arquivo local, se existir.
- **Size:** Tamanho da Foto.
- **RemotePhotoid:** Identificador Remoto da Foto.

OnVideoPhoto

Este evento é chamado quando uma Nova Video Message é recebida. Acesse MessageVideo para obter acesso às propriedades do Video.

- **ChatId:** Identificador do Chat.
- **MessageId:** Identificador da Mensagem.
- **SenderId:** Identificador do Remetente (leia SenderChat e SenderUser a partir do tdlb 1.7.+).
- **VideoId:** Identificador da foto.
- **LocalPath:** caminho completo para o arquivo local, se existir.
- **Width:** largura do vídeo.
- **Height:** altura do vídeo.
- **Duration:** duração em segundos do vídeo.
- **Size:** Tamanho do vídeo.
- **RemoteVideoId:** Identificador da Foto Remota.

OnMessageSponsored

Este evento é chamado quando uma Nova Mensagem Patrocinada foi recebida (após chamar o método getChatSponsoredMessage)

- **SponsorChatId:** Identificador do Chat do Patrocinador.
- **MessageId:** Identificador da Mensagem.
- **Text:** Texto da mensagem.

OnNewChat

Este evento é chamado quando um novo chat é recebido.

- **ChatId:** Identificador do Chat.
- **ChatType:** Tipo de Chat (chatTypeSupergroup, chatTypePrivate...)
- **Title:** Nome do chat.
- **SuperGroupId:** Group Id se for um SuperGroup.
- **IsChannel:** retorna se é um canal ou não.

OnNewCallbackQuery

Este evento é chamado quando uma nova callback query é recebida; somente para bots.

- **Id:** Identificador único da query.
- **SenderId:** Identificador do usuário que enviou a consulta.
- **ChatId:** Identificador do chat no qual a consulta foi enviada.
- **MessageId:** Identificador da mensagem da qual a query se originou.
- **ChatInstance:** Identificador que corresponde exclusivamente ao chat ao qual a mensagem foi enviada.
- **PayloadData:** o payload de um botão de callback geral.
 - **Data:** Dados que foram anexados ao botão de callback.

OnEvent

Este evento é chamado quando um novo Event é recebido pelo Componente API. Pode ser utilizado para processar alguns eventos não implementados pelo Componente API.

- **Event:** Nome do evento (eventos como: updateOption, updateUser...)
- **Text:** mensagem JSON completa

OnException

Este evento é chamado se houver alguma exceção ao processar os dados da Telegram API.

Properties

MyId: retorna o Identificador de Usuário do usuário atual.

Exemplo de Código Completo

Verifique o seguinte exemplo de código que mostra como conectar à API do Telegram, pedir ao usuário para introduzir um Code (se exigido pela API do Telegram), enviar uma mensagem quando a conexão estiver pronta e registrar as Mensagens de Texto recebidas.

```
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 | Send Telegram Message With Inline Buttons

A Telegram API permite que você envie mensagens com botões inline para selecionar opções como resposta (esta opção só está disponível para bots).

Antes de enviar uma mensagem, crie uma instância da classe **TsgcTelegramReplyMarkupInlineKeyboard** e chame o método **AddButtonTypeCallback** ou **AddButtonTypeUrl** para cada botão que você quiser criar.

Exemplo

Cria uma nova mensagem perguntando ao usuário se gosta ou não da mensagem e um link para responder a uma enquete. Processe a resposta utilizando o evento **OnNewCallbackQuery**.

```
TsgcTelegramReplyMarkupInlineKeyboard *oReplyMarkup = new TsgcTelegramReplyMarkupInlineKeyboard();
try
{
    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);
}
__finally
{
    oReplyMarkup->Free();
}

void OnNewCallbackQuery(TObject *Sender, TsgcTelegramCallbackQuery *CallbackQuery)
{
    if (CallbackQuery->PayloadData->Data == "I like it") then
    {
        ShowMessage("yes")
    }
    else
    {
        ShowMessage("no");
    }
}
```

Telegram | Send Bot Message With Buttons

A Telegram API permite que você envie mensagens com botões para solicitar dados do usuário (esta opção está disponível apenas para bots).

Antes de enviar uma mensagem, crie uma instância da classe **TsgcTelegramReplyMarkupShowKeyboard** e chame o método **AddButtonTypeRequestLocation**, **AddButtonTypeRequestPhoneNumber** ou **AddButtonTypeText** para cada botão que você quiser criar.

Exemplo

Cria uma nova mensagem solicitando ao usuário que forneça o PhoneNumber

```
oReplyMarkup = new TsgcTelegramReplyMarkupShowKeyboard();
oReplyMarkup->AddButtonTypeRequestPhoneNumber("Give me your phone");
sgcTelegram->SendTextMessage("123456", "Please provide the information below", null, oReplyMarkup);
oReplyMarkup->Free();
```


Telegram | Send Telegram Message Bold

Você pode destacar mensagens de texto utilizando negrito, itálico e mais estilos. Utilize o método **SendRichTextMessage**, para enviar uma mensagem de Texto com recursos de estilo; este método analisa a mensagem de texto e adiciona as entidades necessárias automaticamente para a API Telegram.

Markdown Syntax

- Bold [*]

```
**This is Bold**
```

- Italic [_]

```
__This is Italic__
```

- Strike [-]

```
--This is Strike--
```

- Sublinhado [~]

```
~~This is Underline~~
```

- Code [#]

```
##This is Monospace##
```

Telegram | Chat not found as Bot

Quando você **faz login como bot**, o método GetChats não pode ser usado, então você não obtém todos os chats disponíveis. Se for a **primeira vez que você faz login como Bot** e tenta **enviar uma mensagem** para um **Chat conhecido**, você receberá este **erro**:

```
{"@type":"error","code":5,"message":"Chat not found"}
```

A solução é chamar o método **GetChat** antes de enviar uma mensagem telegram e passar o **ChatId** como parâmetro. Depois de obter os dados do Chat, você pode enviar mensagens telegram normalmente.

Como nota, você **só** precisa **chamar GetChat** na **PRIMEIRA VEZ** antes de enviar uma mensagem, se você nunca recebeu nenhuma mensagem de bot deste chat. Se você fechar a aplicação e iniciar novamente, não há necessidade de chamar GetChat primeiro, porque o Chat já está salvo no banco de dados do telegram.

Telegram | Sponsored Messages

Cada vez que o usuário abre um canal, `channels.getSponsoredMessages` deve ser chamado para receber as sponsored messages disponíveis para esse canal. O resultado deve ser armazenado em cache por 5 minutos.

Exibição de mensagens patrocinadas

As mensagens patrocinadas devem ser exibidas abaixo de todos os outros posts no canal, depois que o usuário rola mais para baixo, além da última mensagem. O canal ou bot promovido especificado no campo `from_id` deve ser exibido como o autor da mensagem. A mensagem também deve conter um dos seguintes botões na parte inferior:

- **View Bot:** se um bot está sendo promovido. Tocar no botão deve abrir o chat com o bot. Se `start_param` for especificado, o app deve utilizar o mecanismo de deep linking para abrir o bot.
- **View Channel:** se um canal está sendo promovido. Tocar no botão deve abrir o canal.
- **View Post:** se um canal está sendo promovido e `channel_post` é especificado. Tocar no botão deve abrir o post específico do canal.

Uma vez que o texto inteiro seja exibido na tela (excluindo o botão), o método **ViewMessage** deve ser chamado com o `random_id` desta mensagem patrocinada.

Get Sponsored Messages

Envia uma requisição ao canal perguntando se há mensagens patrocinadas disponíveis, basta chamar o método **GetChatSponsoredMessage**.

```
TsgcTDLlib_Telegram *oTelegram = new TsgcTDLlib_Telegram();
oTelegram->Telegram->API->ApiHash = "ABCDEFGHJKLMN";
oTelegram->Telegram->API->ApiId = "1234";
oTelegram->PhoneNumber = "008745744155";
oTelegram->Active = true;
oTelegram->getChatSponsoredMessage("100");
```

Se o chat tiver sponsored messages, o evento **OnMessageSponsored** é chamado com o conteúdo da Sponsored message. Se não houver mensagens, um erro 404 é retornado.

```
private void(TObject *Sender, TsgcTelegramMessageSponsored *MessageSponsored)
{
    DoLog(MessageSponsored->Text);
}
```

Chame o método **ViewMethod** depois que as Sponsored Messages forem exibidas ao usuário.

```
oTelegram->ViewMessage("100", "54653256245");
```

Telegram | Send Telegram Invoice Message

Se o seu bot suportar o modo inline, os usuários também podem enviar invoices para outros chats através do bot, inclusive para chats individuais com outros usuários.

Mensagens de fatura apresentam uma foto e descrição do produto, juntamente com um botão Pay em destaque. Tocar neste botão abre uma interface de pagamento especial no app do Telegram

Os bots podem enviar invoices como uma mensagem utilizando o método **SendInvoiceMessage**.

```
private void SendInvoice()
{
    TsgcTelegramSendInvoice *oInvoice = new TsgcTelegramSendInvoice();
    Try
    {
        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);
    }
    finally
    {
        oInvoice->Free();
    }
}
```

Telegram | Get SuperGroup Members

A API do Telegram permite que você obtenha informações sobre os membros de um SuperGroup. Utilize o método **GetSuperGroupMembers** para obter informações sobre membros ou usuários banidos em um supergrupo ou canal. Só pode ser utilizado se `SupergroupFullInfo.can_get_members` for true; adicionalmente, privilégios de administrador podem ser necessários para alguns filtros.

Por padrão, o método retorna Todos os membros do grupo, mas você pode filtrar os membros retornados utilizando o parâmetro `Filter`. Existem os seguintes parâmetros:

tsgmFilterNone

Valor padrão, significa que os membros não são filtrados.

tsgmFilterAdministrators

Retorna o criador e os administradores.

tsgmFilterBanned

Retorna os usuários banidos do supergrupo ou canal; só pode ser utilizado por administradores. Você pode utilizar o argumento `aSuperGroupMembersQuery` para pesquisar utilizando uma query.

tsgmFilterBots

Retorna os membros bot do supergrupo ou canal.

tsgmFilterContacts

Retorna os contatos do usuário que são membros do supergrupo ou canal. Você pode utilizar o argumento `aSuperGroupMembersQuery` para pesquisar utilizando uma query.

tsgmFilterMention

Retorna os usuários que podem ser mencionados no supergrupo.

tsgmFilterRecent

Retorna os usuários recentemente ativos em ordem cronológica inversa.

tsgmFilterRestricted

Retorna os membros restritos de um supergrupo; só pode ser utilizado por administradores. Você pode utilizar o argumento `aSuperGroupMembersQuery` para pesquisar utilizando uma query.

tsgmFilterSearch

Usado para pesquisar membros de supergrupo ou canal por meio de uma consulta (string). Você pode utilizar o argumento `aSuperGroupMembersQuery` para pesquisar utilizando uma query.

Você pode ler o resultado usando o callback `OnEvent` e filtrando por `event = "chatMembers"`.

```
Telegram->GetSupergroupMembers(1452979380);
```

```
private void OnTelegramEvent(TObject *Sender, const string Event, const string Text)
{
    if (Event == "chatMembers")
    {
        ReadJSON(Text);
    }
}
```

Telegram | Add Telegram Proxy

O Cliente Telegram pode ser configurado para utilizar um proxy. Atualmente, o Telegram suporta 3 tipos de proxies:

1. HTTP
2. MTProto
3. Socks5

Add Proxy

Para configurar um HTTP Proxy, primeiro você deve adicionar o proxy à configuração do telegram; para fazer isto, basta chamar **AddProxyHTTP** e, se bem-sucedido, uma mensagem será retornada com o novo proxy adicionado. Uma vez que o proxy tenha sido adicionado à lista, basta chamar **EnableProxy** e passar o **ID do proxy** recebido na mensagem de confirmação.

```
Telegram->AddProxyHTTP("8.8.8.8", 8080, "", "", true);  
// ... read the confirmation message and save the ID of the proxy.  
Telegram->EnableProxy(2);
```

Remover Proxy

Chame o método **RemoveProxy** e passe o ID do proxy que você deseja remover.

Telegram | Register Telegram User

O processo para registrar um novo usuário no Telegram é muito simples; você precisa do seu API Id e API Hash, e do número de telefone da nova conta.

Configure o cliente telegram:

- API Id
- API Hash
- Número de Telefone da nova conta do telegram.

Inicie o cliente e um novo código será enviado ao telefone, o cliente solicitará o código do telegram e, se estiver correto, o evento `OnRegisterUser` será chamado. Neste evento, defina o First Name e o Last Name do usuário e o registro será concluído.

```
TsgcTDLib_Telegram *oTelegram = new TsgcTDLib_Telegram();
oTelegram->Telegram->API->ApiHash = "ABCDEFGHijklmn";
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

O Source RCON Protocol é um protocolo de comunicação baseado em TCP/IP usado pelo Source Dedicated Server, que permite que comandos de console sejam emitidos ao servidor por meio de um "console remoto", ou RCON. O uso mais comum do RCON é permitir que proprietários de servidores controlem seus game servers sem acesso direto à máquina em que o servidor está sendo executado.

Configuração

As **RCON_Options** permitem que você configure as seguintes propriedades:

- **Host:** endereço remoto do servidor.
- **Port:** porta de escuta do servidor.
- **Password:** é a string secreta usada para autenticar contra o servidor

Connect

Utilize a propriedade **Active** para Conectar / Desconectar do servidor.

Quando Active é definido como True, o cliente tenta conectar ao servidor, se conseguir conectar, ele tentará autenticar utilizando a senha fornecida.

O servidor enviará uma resposta a uma requisição de autenticação. O evento **OnAuthenticate** será chamado e você pode ler se a autenticação foi bem-sucedida ou não utilizando o parâmetro Authenticate.

Enviar Comandos

Utilize o método **ExecCommand** para enviar comandos ao servidor. As respostas estarão disponíveis no 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

O CryptoHopper é um bot de negociação de cripto automatizado que permite que você automatize a negociação e o gerenciamento de portfólio de Bitcoin, Ethereum, Litecoin e mais.

Configuração

Requer uma **Developer Account** e, depois de aprovado, você pode começar a criar um novo App. A API utiliza OAuth2 para autenticar, então você pode obter o **client_id** e o **client_secret** do seu 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->Text = "read,notifications,manage,trade";
```

Métodos

CryptoHopper utiliza HTTPs como protocolo para enviar Requests à API. Alguns métodos exigem autenticação (colocar ordens, recuperar dados do usuário...) e outros são públicos (obter dados da exchange, por exemplo).

As funções retornam a resposta do CryptoHopper e, se houver algum erro, uma exceção será gerada.

Hoppers

Gerencia Operações Básicas de Hopper.

Method	Arguments	Descrição
GetHoppers		Obtém os Hoppers dos usuários.
CreateHopper	aBody: texto json de configuração.	Cria um novo Hopper.
GetHopper	aid: id do hopper	Retrieve Hopper
DeleteHopper	aid: id do hopper	Delete Hopper
UpdateHopper	aid: id do hopper aBody: texto json de configuração.	Atualizar Hopper

Orders

Gerencie as Orders do seu Hopper.

Method	Arguments	Descrição
GetOpenOrders	aid: id do hopper	Recuperar todas as ordens abertas do hopper.
CreateNewOrder	aid: id do hopper aOrder: instância de Tsg-cHTTPCTHOrder	Cria uma nova ordem de compra ou venda. Para venda, utilize preferencialmente o endpoint sell.
PlaceMarketOrder	aid: id do hopper aOrderSide: cthosBuy ou cthosSell. aCoin: nome da moeda, exemplo: EOS aAmount: tamanho da ordem.	Coloca uma ordem a mercado.
PlaceLimitOrder	aid: id do hopper aOrderSide: cthosBuy ou cthosSell. aCoin: nome da moeda, exemplo: EOS aAmount: tamanho da ordem. aPrice: preço limite.	Coloca uma Limit Order
DeleteOrder	aid: hopper id aOrderId: order id	Exclui a ordem do hopper selecionado.
DeleteAllOrders	aid: id do hopper	Exclui todas as ordens abertas do hopper selecionado.
GetOpenOrder	aid: hopper id aOrderId: order id	Obtém a ordem aberta no hopper por id.
CancelOrder	aid: hopper id aOrderId: order id	Cancela uma ordem aberta.

Position

Gerencie as Posições do seu Hopper.

Method	Arguments	Descrição
GetPosition	aid: id do hopper	Obtém as posições abertas do hopper.

Trade

Histórico de Negociações do seu Hopper.

Method	Arguments	Descrição
GetTradeHistory		Obtém o histórico de negociações do hopper.
GetTradeHistoryById	aid: id do hopper aTradeId: id da negociação	Obter uma negociação pelo id do hopper.

Exchange

Obter informações dos exchanges disponíveis no CryptoHopper

Method	Arguments	Descrição
GetExchange		Obter todos os exchanges disponíveis no Cryptohopper.
GetAllTickers	aExchange: nome do exchange	Obter o ticker de todos os pares
GetMarketTicker	aExchange: nome do exchange aPair: nome do par	Obtém o ticker de um par de mercado.
GetOrderBook	aExchange: nome do exchange aPair: nome do par aDepth: profundidade do li-	Obtém o orderbook para a exchange, o mercado e a profundidade de orderbook selecionados.

	vro de ordens	

Webhooks

Histórico de Negociações do seu Hopper.

Method	Arguments	Descrição
CreateWebhook	aURL: url do webhook aMessageTypes: tipos de mensagem separados por vírgula.	Atualiza ou cria um Webhook
DeleteWebhook	aURL: webhook url	Exclui um Webhook existente.

Signals

Send Signals to CryptoHopper API.

Method	Arguments	Descrição
SendSignal	aSignal: é a classe com todos os campos necessários para enviar um sinal.	Envia um Signal
SendTestSignal	aSignal: é a classe com todos os campos necessários para enviar um sinal.	Envia um Sinal de Teste
GetSignalStats	aSignalId: id do sinal. aExchange: opcional, nome do exchange.	Recupera algumas das estatísticas de sinal.

Como atualizar a configuração do Cryptohopper

Utilize o método `UpdateHopper` para atualizar a Configuração do Hopper. O método é sobrecarregado para que você possa passar a string JSON ou utilizar o objeto `TsgcHTTPCTHopper` e usar as propriedades para habilitar ou desabilitar as Propriedades do Hopper.

```
public string EnableHopper()
{
    TsgcHTTPCTHopper *oHopper = new TsgcHTTPCTHopper();
    try
    {
        oHopper->Enabled = 1;
        result = Cryptohopper->UpdateHopper("1234", oHopper);
    }
    __finally
    {
        oHopper->Free();
    }
}
```

Como Configurar o Webhook

O Webhook permite que você receba notificações quando algo acontece em um hopper. Os Webhooks exigem um Servidor HTTPs público que escutará em um endereço de URL todas as mensagens enviadas pelo cryptohopper. O servidor público precisa estar protegido com um certificado SSL (certificados autoassinados não são permitidos).

Primeiro você deve criar um webhook, portanto configure a propriedade Webhook do cliente Cryptohopper definindo o Host e a Port em que o servidor estará escutando. Em seguida, configure o certificado na propriedade `SSLOptions`.

Exemplo: O endereço IP público será 1.1.1.1 e a porta de escuta será 443. O certificado é armazenado como arquivo PEM com o nome `sgc.pem` e sem senha.

```
/* 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

O RTCMultiConnection é uma biblioteca JavaScript WebRTC para aplicações peer-to-peer (compartilhamento de tela, conferência de áudio/vídeo, compartilhamento de arquivos, streaming de mídia, etc.)

Configuração

O RTCMultiConnection requer um servidor WebSocket para Sinalização, portanto vincule a propriedade server do RTCMultiConnection a um Servidor WebSocket (como [TsgcWebSocketHTTPServer](#)). Encontre abaixo as propriedades que você deve configurar.

Server

Host: é o endereço IP público ou o nome DNS do servidor WebSocket.

Port: é a porta de escuta do servidor WebSocket.

IceServers

É a configuração dos servidores ICE (STUN/TURN) para permitir a comunicação entre peers. Exemplo:

```
[
  {
    "urls": "stun:www.yourstun.com"},
  {
    "urls": "turn:www.yourturn.com",
    "username": "user",
    "credential": "secret"
  }
]
```

VideoResolution

Aqui você pode configurar a Resolução de Vídeo das Videoconferências; quanto maior a resolução, mais largura de banda é exigida pela conexão.

HTMLDocuments

Configure para cada Application qual é o nome da página HTML que serve este conteúdo.

Exemplo: se o servidor estiver rodando no site www.webrtc.com na porta 8443 e o HTMLDocuments.VideoConferencing = /RTCMultiConnection-VideoConferencing.html, a url para acessar a VideoConferencing será

<https://www.webrtc.com:8443/RTCMultiConnection-VideoConferencing.html>

O WebRTC requer uma conexão segura (HTTPS), portanto exige o uso de certificados; leia mais em [Server SSL](#).

Applications

Name	Descrição
VideoConferencing	Videochamada multiusuário (muitos-para-muitos) usando o modelo de rede em malha (mesh).
ScreenSharing	Compartilhamento de tela multiusuário (um-para-muitos) usando topologia em estrela.

VideoBro- adcasting	Broadcasting de vídeo multiusuário (um-para-muitos) utilizando topologia em estrela.

WebPush

[RFC 8030](#)

[RFC 8291](#)

O protocolo WebPush é definido pela **RFC 8030** (Delivery using HTTP Push) e pela **RFC 8291** (Message Encryption).

O Web Push é um **protocolo padronizado** para **entregar notificações push a navegadores web**. Ele utiliza a Push API, que é uma API web padrão que permite que sites se registrem e recebam mensagens de push. A Push API permite que um site envie mensagens de push ao navegador de um usuário, mesmo quando o usuário não está navegando ativamente no site.

Para utilizar o Web Push, um site primeiro precisa **obter uma push subscription do navegador do usuário**. A subscription consiste em uma endpoint URL única e uma chave de criptografia. A endpoint URL é uma URL que o site pode utilizar para enviar mensagens push ao navegador do usuário, e a chave de criptografia é utilizada para criptografar e descriptografar as mensagens push.

Uma vez que o site tenha **obtido uma assinatura de push**, ele pode **enviar mensagens de push** para o navegador do usuário fazendo uma requisição HTTP para a URL do endpoint. A mensagem de push é enviada em um formato especial chamado Web Push Protocol Message, que consiste em um conjunto de cabeçalhos e um payload. Os cabeçalhos contêm informações como a chave de criptografia e o TTL (time-to-live) da mensagem, enquanto o payload contém o conteúdo real da mensagem.

Quando o **navegador do usuário recebe uma mensagem push**, ele **primeiro descriptografa a mensagem** usando a chave de criptografia. Em seguida, ele **exibe a notificação ao usuário**, juntamente com quaisquer ações adicionais que o usuário pode realizar, como dispensar a notificação ou abrir o site.

Para garantir a segurança e a privacidade das mensagens push, o Web Push utiliza criptografia de ponta a ponta e exige que as inscrições push sejam obtidas por uma conexão segura (por exemplo, HTTPS). Adicionalmente, o protocolo fornece mecanismos para autenticar o remetente de uma mensagem push e prevenir abusos (por exemplo, limitando o número de mensagens push que um site pode enviar a um usuário).

Componentes

Existem 2 componentes que suportam WebPush:

- **[TsgcWSAPIServer_WebPush](#)**: implementa o protocolo WebPush no lado do servidor, permitindo solicitar permissão aos usuários, registrar assinaturas, enviar notificações e mais. Este componente já encapsula um cliente WebPush para enviar notificações.
- **[TsgcWebPush_Client](#)**: implementa o protocolo WebPush no lado do cliente, permitindo que você envie notificações aos usuários via web desktop e mobile. Isto é útil se você já tem as chaves e o endpoint e só quer publicar mensagens WebPush para os clientes inscritos.

TsgcWSAPIServer_WebPush

Componente *TsgcWSAPIServer_WebPush*.

Introdução

TsgcWSServer_API_WebPush é um componente que fornece funcionalidade para tratar assinaturas WebPush. O WebPush é um protocolo para entregar notificações em tempo real a aplicações web que rodam no navegador. Este componente pode ser usado para gerenciar assinaturas e enviar notificações a clientes assinados. Encontre abaixo as propriedades, eventos e métodos fornecidos pela classe *TsgcWSServer_API_WebPush*, junto com exemplos de código que demonstram como usá-los.

Configuração

1. **Anexe um TsgcWSServer_API_WebPush** a um servidor WebSocket utilizando a propriedade **Server**.
2. Configure as **chaves pública e privada** na propriedade **WebPush.VAPID**. (Usuários registrados podem baixar um executável que gera as chaves VAPID para windows).
3. Requer a implantação da **versão OpenSSL 3.0.0**
4. Na propriedade **WebPush.Endpoints** você pode definir seus próprios endpoints para tratar as assinaturas WebPush; por padrão, acessar o endpoint `"/sgcWebPush.html"` mostrará uma página web simples que permite que você assine as notificações WebPush.
5. Inicie o servidor e acesse o endpoint configurado para testá-lo.

Referência

- Propriedades — 3 propriedades disponíveis no componente.
- Métodos — 3 métodos públicos.
- Eventos — 3 eventos que você pode tratar.

TsgcWebPush_Client

O *TsgcWebPush_Client* é uma classe que permite que você envie uma notificação assim que obtém os detalhes da assinatura.

Introdução

O *TsgcWebPush_Client* é uma classe que permite que você envie uma notificação assim que obtém os detalhes da assinatura.

Encontre abaixo um exemplo de uso do cliente WebPush para enviar uma notificação dado um endpoint, uma chave pública e um segredo de autenticação de uma assinatura WebPush.

```
void SendWebPushNotification()
{
    TsgcHTTP_API_WebPush_PushSubscription* oSubscription = new TsgcHTTP_API_WebPush_PushSubscription();
    try
    {
        oSubscription->Endpoint = "endpoint";
        oSubscription->PublicKey = "public key";
        oSubscription->AuthSecret = "authentication secret";
        TsgcHTTP_API_WebPush_Client* oWebPush = new TsgcHTTP_API_WebPush_Client(NULL);
        try
        {
            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\"}");
        }
        __finally
        {
            oWebPush->Free();
        }
    }
    __finally
    {
        oSubscription->Free();
    }
}
```

Referência

- Propriedades — 3 propriedades disponíveis no componente.
- Métodos — 1 método público.
- Eventos — 0 eventos que você pode tratar.

Extensions

O protocolo WebSocket foi projetado para ser estendido. Os WebSocket Clients podem solicitar extensões e os WebSocket Servers podem aceitar algumas ou todas as extensões solicitadas pelos clientes.

Extensões suportadas:

1. [Deflate-Frame](#): comprime frames WebSocket.
2. [PerMessage-Deflate](#): comprimir mensagens WebSocket.

Extensions | PerMessage-Deflate

PerMessage é uma extensão do protocolo WebSocket; se a extensão for suportada pelo Servidor e pelo Cliente, ambos podem comprimir as mensagens transmitidas:

- Usa Deflate como o método de compressão.
- A compressão se aplica apenas aos dados de Aplicação (frames de controle e cabeçalhos não são afetados).
- O servidor e o cliente podem selecionar quais mensagens serão compactadas.

Max Window Bits

Esta extensão permite personalizar o tamanho da janela deslizante no lado do servidor e do cliente usada pelo algoritmo LZ77 (entre 8 e 15). Quanto maior este valor, mais provável é encontrar e eliminar duplicatas, mas consome mais memória e ciclos de CPU. 15 é o valor padrão.

No Context Take Over

Por padrão, as mensagens anteriores são usadas para compressão e descompressão. Se as mensagens forem semelhantes, isso melhora a taxa de compressão. Se habilitado, então cada mensagem é comprimida usando apenas os dados da própria mensagem. Por padrão, está desabilitado.

MemLevel

Este valor não é negociado entre servidor e cliente. Quando definido como 1, utiliza a menor quantidade de memória, mas reduz a velocidade do algoritmo de compressão e diminui a taxa de compressão; quando definido como 9, utiliza mais memória e oferece o melhor desempenho. Por padrão, é definido como 1.

** A versão do Indy fornecida com o Rad Studio XE2 gera uma exceção por causa de uma incompatibilidade de versão do zlib com as funções de inicialização. Para corrigir isto, basta atualizar sua versão do Indy para a mais recente.*

Extensions | Deflate-Frame

Esta é uma extensão do protocolo WebSocket que permite a compressão de frames enviados usando o protocolo WebSocket, suportada por navegadores WebKit como Chrome ou Safari. Esta extensão é suportada nos componentes Server e Client.

Esta extensão foi descontinuada.

** A versão do Indy fornecida com o Rad Studio XE2 gera uma exceção por causa de uma incompatibilidade de versão do zlib com as funções de inicialização. Para corrigir isto, basta atualizar sua versão do Indy para a mais recente.*

MCP

MCP (Model Context Protocol) é um protocolo padronizado projetado para definir e gerenciar a **troca contextual de informações** entre modelos de linguagem e sistemas externos.

Ele permite uma comunicação estruturada, segura e dinâmica que possibilita que os modelos de IA **compreendam, estendam e interajam** com ferramentas ou fontes de dados externas além de seu ambiente nativo.

O MCP fornece uma **interface uniforme** para passar metadados contextuais, permitindo que os modelos interpretem instruções, recuperem dados e executem tarefas de maneira previsível e interoperável, semelhante em espírito à forma como o *WebAuthn* padroniza os fluxos de autenticação.

Propósito

O MCP foi desenvolvido para resolver os desafios de **fragmentação de contexto** e **inconsistência de integração** entre modelos, clientes e tools.

Com o MCP, qualquer cliente compatível pode comunicar-se com qualquer modelo ou serviço compatível com MCP através de uma estrutura compartilhada e bem definida.

Casos de uso típicos incluem:

- Anexar **metadados de contexto** (sessão do usuário, estado da aplicação, permissões, etc.) a consultas de modelo.
- Troca de **definições de função ou ferramenta** dinamicamente.
- Gerenciamento de **interações com estado** entre sessões do modelo.
- Estabelecer **interoperabilidade segura** entre LLMs, SDKs e serviços de backend.

Componentes

- **TsgcWSAPIServer_MCP**: O componente fornece uma solução simples, porém poderosa, para implementar o servidor MCP, com recursos como requisições de Tools, Prompts e Resources, Autenticação, controle de fluxo usando eventos e muito mais.
- **TsgcWSAPIClient_MCP**: O componente fornece uma solução simples, porém poderosa, para implementar o cliente MCP, com recursos como requisições de Tools, Prompts e Resources e mais.

TsgcWSAPIServer_MCP

Componente *TsgcWSAPIServer_MCP*.

Introdução

O **TsgcWSAPIServer_MCP** expõe o Model Context Protocol (MCP) por meio de um endpoint de servidor HTTP do **sgcWebSockets**. O componente faz a ponte entre requisições HTTP de entrada e o engine **TsgcAI_MCP_Server**, para que clientes compatíveis com MCP possam negociar sessões, enumerar prompts/resources/tools e invocar tools por meio de chamadas no estilo JSON-RPC.

Configuração

- **Posicione os componentes:** O componente **TsgcWSAPIServer_MCP** deve estar anexado a um servidor HTTP, **TsgcWebSocketHTTP_Server** ou **TsgcWebSocketServer_HTTPAPI** utilizando a propriedade **Server**.
- **Escolha um endpoint:** Ajuste **EndpointOptions.Endpoint** se o caminho padrão **/mcp** precisar ser alterado. Requisições cujo documento é igual a este caminho são tratadas como chamadas MCP; outras recorrem ao tratamento de recursos da API pai.
- **Configure as definições do MCP:** Utilize **MCPOptions.ServerInfo** para nomear/versionar o servidor, **MCPOptions.SessionTimeout** para controlar o tempo de vida da sessão e **MCPOptions.AuthenticationOptions** para exigir uma chave de API ou um cabeçalho HTTP personalizado antes que qualquer chamada JSON-RPC seja processada.
- **Registre prompts, ferramentas, recursos e modelos de recurso:** Popule as listas somente leitura **Tools**, **Prompts**, **Resources** e **ResourceTemplates** expostas pelo componente.
- **Tratar eventos:** Conecte os eventos publicados descritos abaixo para inserir a lógica de negócio para o ciclo de vida da sessão, resolução de prompt/recurso ou execução de ferramenta. As exceções podem ser interceptadas para personalizar os códigos de resposta HTTP.

Endpoints HTTP integrados (sem configuração necessária)

Quando o componente está anexado a um servidor HTTP, os seguintes endpoints e comportamentos são servidos automaticamente junto com **/mcp**, sem nenhuma nova propriedade para habilitá-los. Isso habilita compatibilidade pronta para uso com conectores MCP baseados na web, como o **claude.ai**, que requerem OAuth 2.1 com Dynamic Client Registration sobre CORS.

- **Suporte a CORS com reflexão de origem.** Um preflight **OPTIONS** para qualquer caminho tratado retorna 204 No Content com os seguintes cabeçalhos:
 - **Access-Control-Allow-Origin:** eco do cabeçalho **origin** da requisição, ou ***** se ausente.
 - **Access-Control-Allow-Methods:** POST, GET, DELETE, OPTIONS
 - **Access-Control-Allow-Headers:** content-type, mcp-session-id, mcp-protocol-version, authorization, accept
 - **Access-Control-Expose-Headers:** mcp-session-id
 - **Access-Control-Max-Age:** 86400
 - **Strict-Transport-Security:** max-age=31536000
 Todas as respostas não-OPTIONS também incluem **Access-Control-Allow-Origin**, **Access-Control-Expose-Headers** e **Strict-Transport-Security**.
- **401 Unauthorized com www-Authenticate.** Quando **MCPOptions.AuthenticationOptions.ApiKey** (ou **CustomHeader**) está habilitado e a requisição não tem credenciais ou tem credenciais erradas, o servidor responde com 401 e um cabeçalho no formato:


```
Bearer realm="MCP", resource_metadata="https://<host>/.well-known/oauth-protected-resource"
```
- **Endpoints de descoberta e de fluxo OAuth 2.1.** Os endpoints a seguir são servidos no mesmo servidor HTTP que **/mcp**, suportando PKCE (S256), grants de authorization-code e refresh-token, e Dynamic Client Registration:

Path	Spec	Comportamento
<code>GET /.well-known/oauth-authorization-server</code>	RFC 8414	Metadados JSON do servidor de autorização. Anuncia PKCE (S256), os grants <code>authorization_code</code> + <code>refresh_token</code> , os métodos de autenticação public-client + <code>client_secret_basic</code> , registro dinâmico em <code>/oauth/register</code> e o scope <code>mcp</code> .
<code>GET /.well-known/oauth-protected-resource</code>	RFC 9728	JSON descrevendo o endpoint MCP como um recurso protegido e apontando para este servidor como o servidor de autorização.
<code>POST /oauth/register</code>	RFC 7591 (DCR)	Aceita um corpo de registro de cliente JSON (<code>client_name</code> , <code>redirect_uris</code>). Emite um <code>client_id</code> gerado (sem <code>client_secret</code> — cliente público). Retorna <code>201</code> com a resposta de informações do cliente registrado.
<code>GET /oauth/authorize</code>	RFC 6749 + RFC 7636	Renderiza uma página HTML de consentimento. Parâmetros de query obrigatórios: <code>client_id</code> , <code>redirect_uri</code> , <code>response_type=code</code> , <code>code_challenge</code> , <code>code_challenge_method=S256</code> , <code>state</code> , <code>scope</code> .
<code>POST /oauth/authorize</code>	RFC 6749	Processa o formulário de consentimento. O usuário deve inserir o mesmo valor de <code>MCPOptions.AuthenticationOptions.ApiKey.Value</code> para aprovar. Ao aprovar, redireciona para <code>redirect_uri</code> com <code>?code=...&state=...</code> . Ao negar ou em caso de chave errada, redireciona com <code>?error=access_denied&state=...</code> ou re-renderiza o formulário.
<code>POST /oauth/token</code>	RFC 6749 + RFC 7636	Corpo codificado em formulário. Suporta <code>grant_type=authorization_code</code> (valida PKCE: <code>BASE64URL(SHA256(code_verifier)) == code_challenge</code>) e <code>grant_type=refresh_token</code> . Emite um token de acesso bearer de 1 hora mais um token de atualização. Os auth codes expiram após 60 segundos e são de uso único.

- **Validação de Bearer token estendida.** Quando `MCPOptions.AuthenticationOptions.ApiKey.Enabled` é `True`, o cabeçalho `Authorization: Bearer <X>` em requisições `/mcp` é aceito se `<X>` corresponder **ou** ao `ApiKey.Value` estático configurado, **ou** a um access token emitido por OAuth atualmente válido (emitido por meio do fluxo acima).
- **Armazenamento em memória.** Os clientes, códigos e tokens OAuth são mantidos apenas em memória. Eles são perdidos quando o servidor é reiniciado.
- **Sobrescrever via `onMCPHttpRequest`.** Handlers conectados a `onMCPHttpRequest` podem curto-circuitar qualquer um dos comportamentos acima definindo `Handled := True`.

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Métodos — 9 métodos públicos.
- Eventos — 12 eventos que você pode tratar.

MCP Server | Sessions

Após uma inicialização bem-sucedida entre o servidor e o cliente, uma nova sessão é criada. A sessão é sempre enviada na requisição e na resposta entre o cliente e o servidor para identificar qual sessão está sendo utilizada em um fluxo de requisição/resposta. Ao utilizar o transporte HTTP, o tempo de vida de uma sessão é uma única requisição e resposta HTTP, ou seja, uma vez que a resposta tenha sido enviada, a sessão é fechada.

O objeto de sessão é passado em cada evento e, para saber quando uma sessão foi criada ou excluída, utilize os seguintes eventos:

- **OnMCPSessionNew:** toda vez que um novo id de sessão é criado, este evento é chamado.
- **OnMCPSessionEnd:** quando uma sessão é encerrada, este evento é chamado.

OnMCPSessionNew

O evento é chamado quando uma nova sessão é criada. Você pode acessar o parâmetro Session para obter o ID da sessão, quando a sessão expira e mais.

OnMCPSessionEnd

O evento é chamado quando uma sessão é finalizada.

```
void __fastcall TForm1::OnMCPSessionEnd(TObject *Sender, const TsgcAI_MCP_Session *aSession)
{
    printf("#session_end: %s\n", aSession->Id.c_str());
}
void __fastcall TForm1::OnMCPSessionNew(TObject *Sender, const TsgcAI_MCP_Session *aSession)
{
    printf("#session_new: " + aSession->Id);
}
```

MCP Server | Tools

O **TsgcWSAPIServer_MCP** é o componente de alto nível que expõe recursos de servidor Model Context Protocol (MCP) sobre o **sgcWebSockets**. Quando **clientes MCP solicitam ferramentas** de acordo com a especificação MCP Server Tools, este componente orquestra o gateway HTTP, o ciclo de vida da sessão e a serialização JSON-RPC, de modo que seu código Delphi possa focar na implementação da lógica de negócios.

Requisições HTTP recebidas que atingem o endpoint MCP definido em **EndpointOptions.Endpoint** são interceptadas por **TsgcWSServer_API_MCP**. O componente cria (ou retoma) uma sessão MCP, encaminha a requisição para o motor **TsgcAI_MCP_Server** subjacente e, por fim, escreve a resposta JSON-RPC de volta para o cliente HTTP preservando o cabeçalho **mcp-session-id** exigido pela especificação.

Tools List

O servidor mantém um catálogo de ferramentas em memória em **TsgcAI_MCP_ToolsList**. As ferramentas têm unicidade garantida por nome, expõem descrições e fornecem uma descrição de entrada semelhante a JSON-Schema que é emitida na resposta ao método **tools.list** da especificação. Quando um cliente invoca **tools.list**, o **TsgcWSAPIServer_MCP** carrega a requisição, serializa o catálogo atual e o envia de volta com um código de status HTTP 200.

Código de exemplo que publica uma ferramenta de calculadora simples:

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    TsgcAI_MCP_Tool* oTool;
    // Add a tool named "math.add"
    oTool = MCPServer->Tools->AddTool("math.add", "Adds two numbers");
    // Define input schema properties
    oTool->InputSchema->Properties->AddProperty("a", true, aimcpjtNumber, "Left operand");
    oTool->InputSchema->Properties->AddProperty("b", true, aimcpjtNumber, "Right operand");
}
```

Uma resposta de **tools.list** em conformidade enviada ao cliente espelhará o payload do JSON schema do MCP. Por exemplo:

```
{
  "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" }
          }
        }
      }
    ]
  }
}
```

Requisição de Ferramenta

Quando um cliente emite uma requisição JSON-RPC **tools.call**, o **TsgcWSAPIServer_MCP** hidrata o objeto de requisição fortemente tipado (incluindo o nome da ferramenta e os argumentos fornecidos) antes de gerar o evento

OnMCPRequestTool . Seu handler popula a carga útil da resposta, que é então serializada de volta ao cliente, junto com um código de status HTTP de sucesso.

Um handler típico se parece com isto:

```
void __fastcall TMainForm::MCPServerMCPRequestTool(
    TObject *Sender,
    const TsgcAI_MCP_Session *ASession,
    const TsgcAI_MCP_Request_ToolsCall *ARequest,
    const TsgcAI_MCP_Response_ToolsCall *AResponse)
{
    double LA, LB;
    if (ARequest->Params->Name == "math.add")
    {
        LA = ARequest->Params->Arguments->Node["a"]->AsNumber;
        LB = ARequest->Params->Arguments->Node["b"]->AsNumber;
        AResponse->Result->Content->AddText(
            Format("Sum = %.2f", ARRAYOFCONST((LA + LB)))
        );
    }
    else
    {
        AResponse->Result->IsError = true;
    }
}
```

A resposta JSON-RPC gerada seguirá o schema tools.call da especificação, envolvendo uma ou mais partes de conteúdo e, opcionalmente, um payload estruturado. Como o componente limpa a resposta e define o código HTTP para você, nenhuma infraestrutura adicional é necessária.

Public API surface

Várias classes auxiliares expõem o catálogo de ferramentas e os helpers de resposta de eventos. As listas a seguir resumem os métodos que normalmente são chamados a partir de código de produção.

TsgcAI_MCP_ToolsList

- **AddTool(const Name, Description):** cria ou substitui uma entrada de tool e retorna o **TsgcAI_MCP_Tool** subjacente para que você possa configurar schemas e padrões.
- **Clear:** remove todas as ferramentas registradas. Chame isto antes de reconstruir o catálogo quando a configuração é carregada do disco.
- **Count / Items[Index]:** expõem o conteúdo da lista para enumeração, diagnósticos ou serialização personalizada.

TsgcAI_MCP_Tool

- **InputSchema.Properties.AddProperty(Name, Required, JsonType, Description):** documenta o payload de entrada JSON que será anunciado por meio de **tools.list**.
- **InputSchema.Properties.Clear:** reconstrói o schema dinamicamente quando sua aplicação atualiza as capacidades de ferramentas em tempo de execução.
- **Description:** propriedade gravável que armazena o resumo legível por humanos compartilhado com os clientes.

TsgcAI_MCP_Response_ToolsCall

- **Result.Content.AddText(Value):** anexa blocos de texto à resposta da ferramenta. Múltiplas chamadas produzem respostas de várias partes.
- **Result.Content.AddImage(Data, MimeType):** anexa blocos de imagem à resposta da ferramenta. Múltiplas chamadas produzem respostas multi-part

- **Result.Content.AddAudio(Data, MimeType):** anexa blocos de áudio à resposta da ferramenta. Múltiplas chamadas produzem respostas multipart.
- **Result.Content.AddEmbeddedResource(Uri, Title, Text, MimeType):** anexa blocos de recurso embutido à resposta da ferramenta. Múltiplas chamadas produzem respostas multipart.
- **Result.Content.AddResourceLink(Uri, Name, MimeType):** anexa blocos de resource link à resposta da tool. Múltiplas chamadas produzem respostas de múltiplas partes.
- **Result.Content.Clear:** redefine qualquer saída gerada anteriormente antes de você anexar novo conteúdo.
- **Result.IsError:** defina este flag como **True** quando a chamada da ferramenta não puder ser atendida, para que o framework serialize um objeto de erro MCP.

Exemplo avançado

O próximo trecho constrói uma tool de busca geográfica que valida as entradas, retorna múltiplos blocos de texto e ativa **IsError** quando os parâmetros são inválidos.

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    TsgcAI_MCP_Tool *tool = MCPServer->Tools->AddTool("geo.locate", "Returns coordinates for a city and country")
    tool->InputSchema->Properties->Clear();
    tool->InputSchema->Properties->AddProperty("city", true, aimcpjtString, "City to search");
    tool->InputSchema->Properties->AddProperty("country", false, aimcpjtString, "Optional ISO country code");
}
void __fastcall TMainForm::MCPServerMCPRequestTool(
    TObject *Sender,
    const TsgcAI_MCP_Session *ASession,
    const TsgcAI_MCP_Request_ToolsCall *ARequest,
    const TsgcAI_MCP_Response_ToolsCall *AResponse)
{
    if (!SameText(ARequest->Params->Name, "geo.locate"))
        return;
    UnicodeString city = ARequest->Params->Arguments->Node["city"]->AsString();
    UnicodeString country = ARequest->Params->Arguments->Node["country"]->AsString();
    AResponse->Result->Content->Clear();
    if (city.IsEmpty())
    {
        AResponse->Result->Content->AddText("City cannot be empty.");
        AResponse->Result->IsError = true;
        return;
    }
    AResponse->Result->Content->AddText("City: " + city);
    if (!country.IsEmpty())
        AResponse->Result->Content->AddText("Country: " + country);
    AResponse->Result->Content->AddText("Coordinates: " + LookupCityCoordinates(city, country));
}
```

MCP Server | Prompts

No MCP, *prompts* são templates ou fluxos de trabalho reutilizáveis que o servidor expõe. Eles são controlados pelo usuário (os clientes escolhem qual prompt invocar). Os dois principais métodos são:

- **prompts/list**: Requisita a lista de todas as definições de prompt disponíveis expostas pelo servidor. É tratado internamente pelo componente `TsgcWSAPIServer_MCP`.
- **prompts/get**: Recupera o conteúdo de um prompt específico e metadados opcionais, normalmente uma sequência de mensagens descrevendo como o cliente deve interagir. O evento **OnMCPRequestPrompt** é chamado quando uma nova requisição é recebida pelo servidor.

Adicionalmente, se o servidor definir `prompts.listChanged = true` em suas capabilities declaradas, ele pode enviar notificações como `notifications/prompts/list_changed` quando o catálogo de prompts é atualizado.

Quando um cliente solicita `prompts/get`, ele envia argumentos (se houver) e espera uma sequência de mensagens (papéis `user` ou `assistant`) com blocos de conteúdo (geralmente texto, mas podem incorporar recursos ou imagens).

Se algo estiver errado (prompt ausente, args inválidos), o servidor retornará códigos de erro JSON-RPC como `-32602` (Invalid params) ou `-32603` (Internal error).

Prompts List

O servidor mantém um catálogo de prompts em memória em **TsgcAI_MCP_PromptsList**. Os prompts têm unicidade garantida por nome, expõem descrições e fornecem uma descrição de entrada semelhante a JSON-Schema que é emitida na resposta ao método `prompts.list` da especificação. Quando um cliente invoca `prompts.list`, o `TsgcWSAPIServer_MCP` carrega a requisição, serializa o catálogo atual e o envia de volta com um código de status HTTP 200.

Código de exemplo que publica um prompt de revisão de código:

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    TsgcAI_MCP_Prompt* oPrompt;
    // Clear all existing prompts
    MCPServer->Prompts->Clear();
    // Create a new prompt named "CodeReview"
    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
    );
}
```

Prompt Request

Quando um cliente emite uma requisição JSON-RPC `prompts.get`, o **TsgcWSAPIServer_MCP** hidrata o objeto de requisição fortemente tipado (incluindo o nome do prompt e os argumentos fornecidos) antes de gerar o evento **OnMCPRequestPrompt**. Seu handler preenche o payload da resposta, que é então serializado de volta para o cliente, junto com um código de status HTTP de sucesso.

Um handler típico se parece com isto:

```

void __fastcall TFRMMCPServer::MCPServerMCPRequestPrompt(
    TObject *Sender,
    const TsgcAI_MCP_Session *aSession,
    const TsgcAI_MCP_Request_PromptsGet *aRequest,
    const TsgcAI_MCP_Response_PromptsGet *aResponse)
{
    if (aRequest->Params->Name == "CodeReview")
    {
        aResponse->Result->Description = "Code review prompt";
        aResponse->Result->Messages->AddText(
            "user",
            "Please review this Delphi code: ShowMessage('Hello World');"
        );
    }
}

```

Public API surface

Os prompts são modelados através de um punhado de classes helper. Os métodos a seguir são usados com mais frequência ao curar templates de prompt e ao compor uma resposta **prompts/get**.

TsgcAI_MCP_PromptsList

- **AddPrompt(const Name, Description):** registra uma nova entrada de prompt e retorna um objeto **TsgcAI_MCP_Prompt** para que os argumentos e as mensagens possam ser configurados.
- **Clear:** limpa o catálogo, útil durante a inicialização quando os prompts são carregados de arquivos de configuração.
- **Count / Items[Index]:** fornecem acesso ao conteúdo da lista para diagnósticos, visualização na interface ou serialização manual.

TsgcAI_MCP_Prompt

- **Arguments.AddArgument(Name, Description, Required):** declara as entradas estruturadas que o prompt aceita.
- **Arguments.Clear:** reconstrói a lista de argumentos quando seu prompt muda em tempo de execução.
- **Messages.AddText(Role, Text):** pré-popula turnos conversacionais que são enviados de volta ao cliente como parte da definição do prompt.

TsgcAI_MCP_Response_PromptsGet

- **Result.Description:** resumo curto e legível para humanos que acompanha o payload do prompt.
- **Result.Messages.AddText(Role, Text):** injeta mensagens adicionais dinamicamente quando o prompt é buscado.
- **Result.Messages.AddImage(Data, MimeType):** acrescenta blocos de imagem à resposta do prompt. Múltiplas chamadas produzem respostas multi-parte.
- **Result.Messages.AddAudio(Data, MimeType):** anexa blocos de áudio à resposta do prompt. Múltiplas chamadas produzem respostas multi-parte.
- **Result.Messages.AddEmbeddedResource(Uri, Title, Text, MimeType):** anexa blocos de embedded resource à resposta do prompt. Múltiplas chamadas produzem respostas multipartes.
- **Result.Messages.Clear:** reinicia a resposta antes de reutilizar o objeto.

Exemplo avançado

O próximo exemplo publica um prompt que guia um LLM por um fluxo de trabalho de triagem. Ele destaca os argumentos opcionais e como retornar instruções de múltiplos passos no handler.

```

void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    TsgcAI_MCP_Prompt *prompt;
    MCPServer->Prompts->Clear();
    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.");
}

void __fastcall TMainForm::MCPServerMCPRequestPrompt(
    TObject *Sender,
    const TsgcAI_MCP_Session *aSession,
    const TsgcAI_MCP_Request_PromptsGet *aRequest,
    const TsgcAI_MCP_Response_PromptsGet *aResponse)
{
    if (!SameText(aRequest->Params->Name, "IssueTriage"))
        return;
    aResponse->Result->Description = "Guided checklist for support analysts.";
    aResponse->Result->Messages->Clear();
    aResponse->Result->Messages->AddText("user", "Review the ticket summary and ask clarifying questions.");
    aResponse->Result->Messages->AddText("assistant", "Acknowledge the request and confirm the escalation path.");
    if (aRequest->Params->Arguments->Node["customerPriority"]->AsString() != "")
        aResponse->Result->Messages->AddText("assistant", "Adjust SLA checks based on the provided customer prior
}

```

MCP Server | Resources

No MCP, os **resources** representam objetos de dados endereçáveis que o servidor expõe, como arquivos, registros de banco de dados, documentos gerados ou saídas dinâmicas de API.

Eles são *controlados pelo cliente*, o que significa que o cliente pode decidir qual recurso solicitar e como interpretar o conteúdo.

Os dois principais métodos envolvidos são:

- **resources/list**: Solicita a lista de todas as definições de recurso disponíveis que o servidor expõe. É tratado internamente pelo componente `TsgcWSAPIServer_MCP`.
- **resources/read**: Recupera o conteúdo e os metadados opcionais de um recurso específico. O evento **OnMCPRequestResource** é chamado quando uma nova requisição é recebida pelo servidor. Um recurso normalmente inclui:
 - **uri**: identificador do recurso.
 - **mimeType**: tipo de conteúdo (text/plain, application/json, image/png, etc.).
 - **data ou content**: a carga útil real (texto ou binário).

Adicionalmente, se o servidor define `resources.listChanged = true` em suas capacidades declaradas, ele pode enviar notificações como `notifications/resources/list_changed` quando o catálogo de recursos é atualizado.

Lista de Recursos

O servidor mantém um catálogo em memória de recursos em **TsgcAI_MCP_ResourcesList**. Os recursos têm garantia de serem únicos por uri, expõem descrições e fornecem uma URI de entrada semelhante a JSON-Schema que é emitida na resposta ao método `resources.list` da especificação. Quando um cliente invoca `resources.list`, o **TsgcWSAPIServer_MCP** carrega a requisição, serializa o catálogo atual e o envia de volta com um código de status HTTP 200.

Código de exemplo que publica um recurso de arquivo:

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    // Clear any previously registered 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
    );
}
```

Requisição de Recursos

Quando um cliente emite uma requisição JSON-RPC `resource.read`, o **TsgcWSAPIServer_MCP** hidrata o objeto de requisição fortemente tipado (incluindo o recurso uri, name e os argumentos fornecidos) antes de gerar o evento **OnMCPRequestResource**. Seu handler preenche o payload de resposta, que é então serializado de volta ao cliente, juntamente com um código de status HTTP de sucesso.

Um handler típico se parece com isto:

```
void __fastcall TMainForm::OnMCPRequestResource(
    TObject *Sender,
```



```

const TsgcAI_MCP_Session *aSession,
const TsgcAI_MCP_Request_ResourcesRead *aRequest,
const TsgcAI_MCP_Response_ResourcesRead *aResponse)
{
    if (aRequest->Params->Uri == "file:///project/src/main.rs")
    {
        aResponse->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
        );
    }
}

```

Public API surface

Os Resources combinam uma entrada de catálogo (**TsgcAI_MCP_Resource**) e os auxiliares de streaming utilizados dentro de **OnMCPRequestResource**. Estes são os métodos que você utilizará com mais frequência.

TsgcAI_MCP_ResourcesList

- **AddResource(Uri, Name, Title, Description, MimeType)**: registra um recurso estático ou gerado dinamicamente.
- **Clear**: remove todas as definições de recurso, útil antes de reconstruir a lista a partir de arquivos de configuração.
- **Count / Items[Index]**: expõem o catálogo para enumeração ou visualizações de diagnóstico.

TsgcAI_MCP_Resource

- **MimeType / Title / Description**: propriedades graváveis que descrevem o payload do recurso retornado ao cliente.
- **Name / Uri**: utilize estas propriedades para atualizar identificadores sem registrar novamente a entrada de recurso.

TsgcAI_MCP_Response_ResourcesRead

- **Result.Contents.AddContentText(Uri, Name, Title, MimeType, Text)**: envia um payload de texto de volta ao chamador.
- **Result.Contents.AddContentBlob(Uri, Name, Title, MimeType, Text)**: envia um payload de texto de volta ao chamador.
- **Result.Contents.Clear**: reinicia a carga útil de saída antes de reutilizar o objeto de resposta.
- **Result.IsError**: sinaliza a resposta como um erro quando o recurso não pode ser produzido.

Exemplo avançado

O exemplo abaixo publica um recurso JSON que retorna métricas agregadas. Ele demonstra como validar a URI recebida e elaborar respostas diferentes dependendo da requisição.

```

void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    MCPServer->Resources->Clear();
    MCPServer->Resources->AddResource(
        "metrics://daily",
        "DailyMetrics",
        "Daily KPI snapshot",
        "Summaries generated each night",

```

```

        "application/json");
    }
    void __fastcall TMainForm::MCPServerMCPRequestResource(
        TObject *Sender,
        const TsgcAI_MCP_Session *aSession,
        const TsgcAI_MCP_Request_ResourcesRead *aRequest,
        const TsgcAI_MCP_Response_ResourcesRead *aResponse)
    {
        UnicodeString targetUri = aRequest->Params->Uri.LowerCase();
        aResponse->Result->Contents->Clear();
        if (targetUri == "metrics://daily")
        {
            aResponse->Result->Contents->AddContentText(
                "metrics://daily",
                "DailyMetrics",
                "Daily KPI snapshot",
                "application/json",
                BuildMetricsPayload(Date()));
            return;
        }
        aResponse->Result->IsError = true;
        aResponse->Result->Contents->AddContentText(
            aRequest->Params->Uri,
            "UnknownResource",
            "Not found",
            "text/plain",
            "Resource is not published by this server.");
    }

```

MCP Server | Roots

Roots marcam as localizações do sistema de arquivos que um cliente permite que o servidor explore. Cada entrada é um URI file:// com um nome de exibição opcional, permitindo que as automações MCP respeitem os limites do projeto enquanto o usuário mantém o controle sobre o que é compartilhado. Os servidores consultam o catálogo emitindo roots/list e, quando o cliente anuncia listChanged, recebem notifications/roots/list_changed para que possam atualizar as visualizações em cache.

Como os roots pertencem ao cliente, o servidor sempre atua como o requisitante. A capacidade roots deve estar presente no payload de initialise do cliente antes de você tentar uma chamada de descoberta; caso contrário, RequestRootsList gera CapabilityNotSupported.

Solicitando o catálogo raiz

TsgcWSAPIServer_MCP expõe um helper que serializa roots/list e roteia a resposta de volta ao seu código:

- **Helper de requisição:** chame RequestRootsList(Session) assim que a sessão MCP anunciar a capacidade roots. O método atribui um identificador automaticamente (a menos que você forneça um) e persiste a requisição pendente para que a resposta possa ser correlacionada.
- **Evento de resposta:** trate OnMCPResponseRootsList(Sender, Session, Request, Response). O evento é disparado quando o cliente responde com os diretórios disponíveis. Inspeione Response.Roots ou registre Response.Write para examinar o payload JSON.
- **Deteção de alterações:** se o cliente enviar posteriormente notifications/roots/list_changed, envie outro RequestRootsList para que o cache do lado do servidor permaneça sincronizado.

Código de exemplo

Os trechos abaixo requisitam roots assim que uma sessão é estabelecida e registram as entradas retornadas.

```
void __fastcall TFRMMCPServer::FormCreate(TObject *Sender)
{
    MCPServer->OnMCPSessionNew = MCPServerSessionNew;
    MCPServer->OnMCPResponseRootsList = MCPServerRootsList;
}

void __fastcall TFRMMCPServer::MCPServerSessionNew(
    TObject *Sender,
    const TsgcAI_MCP_Session *ASession)
{
    if (ASession->ClientCapabilities->Roots->Enabled)
    {
        System::UnicodeString requestId = MCPServer->RequestRootsList(ASession);
        MemoLog->Lines->Add("roots/list sent id " + requestId);
    }
}

void __fastcall TFRMMCPServer::MCPServerRootsList(
    TObject *Sender,
    const TsgcAI_MCP_Session *ASession,
    const TsgcAI_MCP_Request_RootsList *ARequest,
    const TsgcAI_MCP_Response_RootsList *AResponse)
{
    MemoLog->Lines->Add(
        Format("roots/list returned %d entries", ARRAYOFCONST((AResponse->Roots->Count)))
    );
    MemoLog->Lines->Add("roots/list payload " + AResponse->Write());
}
```

MCP Server | Sampling

O sampling permite que o servidor peça ao cliente para executar uma geração de LLM em seu nome. As requisições descrevem um histórico de conversa, prompt de sistema opcional, preferências de modelo e orçamentos de tokens. O cliente mantém o controle de qual provedor é utilizado, realiza quaisquer revisões com humano no loop e, por fim, retorna a completion aprovada.

Os clientes devem declarar a capacidade sampling durante o initialize; caso contrário, uma tentativa de sampling é rejeitada com um erro de capacidade. A especificação incentiva a aprovação do usuário e permite blocos de conteúdo de texto, imagem ou áudio dentro do histórico de conversa.

Emitindo uma requisição de sampling

TsgcWSAPIServer_MCP simplifica o fluxo sampling/createMessage com um helper e um evento de resposta:

- **Helper de requisição:** construa um `TsgcAI_MCP_Request_SamplingCreateMessage`, preencha `Params.Messages`, `Params.ModelPreferences`, `Params.SystemPrompt` e, opcionalmente, `Params.MaxTokens`. Passe-o a `RequestSamplingCreateMessage(Session, Request)`; o método carimba um id (se necessário) e registra a chamada pendente.
- **Evento de resposta:** trate `OnMCPResponseSamplingCreateMessage(Sender, Session, Request, Response)`. Inspeção `Response.Role`, `Response.Content`, `Response.Model` e `Response.StopReason` para continuar seu fluxo de trabalho.
- **Tratamento de erros:** requisições recusadas ou canceladas chegam como erros JSON-RPC, então envolva o helper em try/except e monitore `OnMCPException` para registro.

Código de exemplo

Os trechos abaixo solicitam um resumo curto ao cliente e imprimem a resposta do assistente.

```
void __fastcall TFRMMCPServer::FormCreate(TObject *Sender)
{
    MCPServer->OnMCPResponseSamplingCreateMessage = MCPServerSamplingResponse;
}

void __fastcall TFRMMCPServer::RequestTicketSummary(
    const TsgcAI_MCP_Session *ASession,
    const System::UnicodeString ATicketText)
{
    if (!ASession->ClientCapabilities->Sampling->Enabled)
        return;
    std::unique_ptr<TsgcAI_MCP_Request_SamplingCreateMessage> request(
        new TsgcAI_MCP_Request_SamplingCreateMessage());
    request->Params->Messages->AddTextMessage(
        "user",
        "Summarise the following ticket: " + ATicketText
    );
    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;
    MemoLog->Lines->Add(
        "sampling/createMessage id " +
        MCPServer->RequestSamplingCreateMessage(ASession, request.get())
    );
}

void __fastcall TFRMMCPServer::MCPServerSamplingResponse(
    TObject *Sender,
    const TsgcAI_MCP_Session *ASession,
    const TsgcAI_MCP_Request_SamplingCreateMessage *ARequest,
    const TsgcAI_MCP_Response_SamplingCreateMessage *AResponse)
{
    if (dynamic_cast<TsgcAI_MCP_Response_Result_Content_Text*>(AResponse->Content) != nullptr)
    {
        auto *text = static_cast<TsgcAI_MCP_Response_Result_Content_Text*>(AResponse->Content);
        MemoLog->Lines->Add("assistant: " + text->Text);
    }
    MemoLog->Lines->Add(
        "model: " + AResponse->Model + " reason: " + AResponse->StopReason
    );
}
```

MCP Server | Elicitation

A elicitation permite que o servidor solicite ao cliente que colete entrada estruturada do usuário no meio de um fluxo de trabalho. A requisição contém uma mensagem amigável ao usuário, além de um JSON schema restrito que descreve os campos que devem ser capturados. O cliente renderiza um formulário apropriado, valida a resposta localmente e retorna se o usuário aceitou, recusou ou cancelou o prompt.

O cliente deve anunciar a capacidade elicitation durante o initialize. Os esquemas são intencionalmente limitados a objetos planos com propriedades primitivas (string, number, boolean ou enum) para tornar a renderização direta e manter dados sensíveis fora do escopo.

Solicitando entrada do usuário

O `TsgcWSAPIServer_MCP` fornece um auxiliar para elicitation/create e gera um evento de resposta quando o usuário termina de interagir:

- **Request helper:** crie `TsgcAI_MCP_Request_ElicitationCreate`, defina `Params._Message` e descreva o payload esperado através de `Params.RequestedSchema`. Chame `RequestElicitationCreate(Session, Request)` para enviá-lo ao cliente.
- **Evento de resposta:** trate `OnMCPResponseElicitationCreate(Sender, Session, Request, Response)`. Inspecione `Response.Action` (accept, decline ou cancel) e analise `Response.Content` quando o usuário aceitar.
- **Validação:** as requisições rejeitadas chegam como erros JSON-RPC. Sempre verifique `Action` antes de consumir qualquer conteúdo e esteja preparado para perguntar novamente se o usuário cancelou.

Código de exemplo

Os trechos abaixo solicitam informações de contato, aguardam a resposta e registram a ação resultante mais o payload.

```
void __fastcall TFRMMCPServer::FormCreate(TObject *Sender)
{
    MCPServer->OnMCPResponseElicitationCreate = MCPServerElicitationResponse;
}

void __fastcall TFRMMCPServer::RequestContactDetails(
    const TsgcAI_MCP_Session *ASession)
{
    if (!ASession->ClientCapabilities->Elicitation->Enabled)
        return;
    std::unique_ptr <TsgcAI_MCP_Request_ElicitationCreate> 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.";
    auto *name = request->Params->RequestedSchema->Properties->AddProperty("name");
    name->_Type = "string";
    name->Title = "Full name";
    name->Required = true;
    auto *email = request->Params->RequestedSchema->Properties->AddProperty("email");
    email->_Type = "string";
    email->Title = "Email address";
    email->Format = "email";
    email->Required = true;
    auto *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;
    MemoLog->Lines->Add(
        "elicitation/create id " +
        MCPServer->RequestElicitationCreate(ASession, request.get())
    );
}

void __fastcall TFRMMCPServer::MCPServerElicitationResponse(
    TObject *Sender,
    const TsgcAI_MCP_Session *ASession,
    const TsgcAI_MCP_Request_ElicitationCreate *ARequest,
    const TsgcAI_MCP_Response_ElicitationCreate *AResponse)
{
    MemoLog->Lines->Add("elicitation action: " + AResponse->Action);
    if (AResponse->Action.LowerCase() == "accept" && AResponse->Content != nullptr)
```

```
    MemoLog->Lines->Add("payload: " + AResponse->Content->Text);  
}
```

TsgcWSAPIClient_MCP

Componente *TsgcWSAPIClient_MCP*.

Introdução

O **TsgcWSAPIClient_MCP** implementa um consumidor de Model Context Protocol (MCP) sobre os componentes de rede *sgcWebSockets*. Ele cuida da negociação de sessão, do marshalling de requisição/resposta JSON-RPC e de eventos de conveniência, de modo que aplicações Delphi & Cbuilder possam descobrir prompts, resources e tools expostos por qualquer servidor compatível com MCP.

Configuração

- **Solte o componente:** coloque o **TsgcWSAPIClient_MCP** em um formulário/data-module.
- **Aponte para um endpoint MCP:** defina **MCOptions.HttpOptions.URL** com a URL HTTP do servidor que expõe o protocolo (por exemplo `https://localhost:5001/mcp`). Configure TLS e registro por meio dos objetos aninhados **TLSoptions** e **LogOptions** se necessário.
- **Descreva o cliente:** popule **MCOptions.ClientInfo** com o *Name* do produto, o *Title* legível por humanos e a *Version* semântica. Esses valores são encaminhados durante o handshake **initialize**.
- **Proteja a sessão:** configure **MCOptions.AuthenticationOptions** quando o servidor espera uma API key ou um cabeçalho HTTP personalizado. As credenciais são anexadas a cada requisição HTTP, incluindo upgrades de streaming.
- **Ajuste o keep-alive:** utilize **MCOptions.HeartBeat** para habilitar o ciclo automático de ping e defina o intervalo preferido (segundos). O componente chama internamente **Ping** quando o temporizador interno de keep-alive dispara.
- **Manipule os eventos:** assine os eventos descritos mais adiante para registrar respostas em log, aceitar/rejeitar sessões ou pós-processar payloads retornados pelo servidor.

Referência

- Propriedades — 2 propriedades disponíveis no componente.
- Métodos — 12 métodos públicos.
- Events — 12 eventos que você pode tratar.

MCP Client | Tools

O componente `TsgcWSAPIClient_MCP` expõe as chamadas JSON-RPC definidas pelo Model Context Protocol (MCP) para enumerar as tools que o servidor oferece e para invocar uma tool. Cada requisição deve ser enviada depois que `Initialize` for bem-sucedido, porque cada método valida que a sessão já está no estado *initialized*.

Requisição ListTools

Chame **ListTools** para recuperar o registro de ferramentas que o servidor MCP expõe atualmente. O componente constrói automaticamente a requisição JSON-RPC `tools/list` e gera o evento **OnMCPListTools** com a resposta parseada.

- **Request:** `ListTools` (sem parâmetros).
- **Event:** `OnMCPListTools(Sender, Request, Response)`.
- **Response:** `TsgcAI_MCP_Response_ToolsList` com uma fila `Tools`. Cada item é um `TsgcAI_MCP_Tool` que expõe o `Name`, `Title`, `Description` da ferramenta e seu `InputSchema` (tipo de schema JSON mais propriedades nomeadas).

Utilize os helpers `Count` e `Item[Index]` da `queue`, ou `GetTool`, para inspecionar o catálogo.

ToolsCall request

Chame **RequestTool**(`Name`, `Arguments`) para invocar uma das ferramentas publicadas. O componente copia os argumentos JSON para o payload da requisição e gera **OnMCPResponseTool** depois que a resposta é parseada.

- **Requisição:** `RequestTool` com o **Name** da tool e os **Arguments** JSON opcionais.
- **Event:** `OnMCPResponseTool(Sender, Request, Response)`.
- **Response:** `TsgcAI_MCP_Response_ToolsCall`. Inspeione `Result.IsError`, `Result.StructuredContent` e itere `Result.Content` para obter os fragmentos de saída estruturada. As respostas de ferramentas podem conter texto, imagens, blobs de áudio, `embedded resources` ou `resource links`.

Código de exemplo

Os trechos a seguir mostram como solicitar o inventário de ferramentas, executar uma ferramenta e ler os resultados.

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    MCP->OnMCPListTools = MCPListTools;
    MCP->OnMCPResponseTool = MCPResponseTool;
}
void __fastcall TForm1::LoadTools()
{
    MCP->ListTools();
}
void __fastcall TForm1::MCPListTools(TObject *Sender,
const TsgcAI_MCP_Request_ToolsList &ARequest,
const TsgcAI_MCP_Response_ToolsList &AResponse)
{
    for (int LIndex = 0; LIndex < AResponse.Tools->Count; LIndex++)
    {
        TsgcAI_MCP_Tool *LTool =
            static_cast<TsgcAI_MCP_Tool*>(AResponse.Tools->Item[LIndex]);
        Memo1->Lines->Add(Format("%s (%s)", ARRAYOFCONST((LTool->Name, LTool->Description))));
    }
}
void __fastcall TForm1::CallTemperatureTool()
{
    TsgcJSON *LArgs = new TsgcJSON(nullptr);
    try
    {
        LArgs->AddPair("city", "Barcelona");
        MCP->RequestTool("GetTemperature", LArgs);
    }
    __finally
```



```

    {
        delete LArgs;
    }
}
void __fastcall TForm1::MCPResponseTool(TObject *Sender,
const TsgcAI_MCP_Request_ToolsCall &ARequest,
const TsgcAI_MCP_Response_ToolsCall &AResponse)
{
    if (AResponse.Result->IsError)
        Memo1->Lines->Add("Tool returned an error payload");
    for (int LIndex = 0; LIndex < AResponse.Result->Content->Count; LIndex++)
    {
        TsgcAI_MCP_Response_Result_Content *LContent =
            static_cast<TsgcAI_MCP_Response_Result_Content*>(
                AResponse.Result->Content->Item[LIndex]);
        Memo1->Lines->Add(LContent->Write());
    }
}

```

MCP Client | Prompts

O componente MCP Client oferece dois métodos JSON-RPC que giram em torno de prompts fornecidos pelo servidor: listar os modelos de prompt disponíveis e solicitar a execução de um prompt. Certifique-se de que o cliente foi inicializado antes de emitir essas chamadas.

PromptsList request

ListPrompts solicita o catálogo de prompts atual. Quando a resposta é recebida, o componente dispara **OnMCPListPrompts**.

- **Request:** ListPrompts (sem parâmetros).
- **Evento:** OnMCPListPrompts(Sender, Request, Response).
- **Response:** TsgcAI_MCP_Response_PromptsList expando uma fila Prompts. Cada TsgcAI_MCP_Prompt contém seu Name, Title, Description e a coleção Arguments (nome do argumento, descrição e flag de obrigatoriedade).

PromptsGet request

RequestPrompt(Name, Arguments) executa um template de prompt com os argumentos JSON fornecidos. O componente gera **OnMCPResponsePrompt** quando a resposta do servidor está pronta.

- **Request:** RequestPrompt com o **Name** do prompt e **Arguments** JSON opcionais.
- **Event:** OnMCPResponsePrompt(Sender, Request, Response).
- **Response:** TsgcAI_MCP_Response_PromptsGet. Inspeção Result.Description para um resumo e itere Result.Messages para ler cada TsgcAI_MCP_Response_PromptsGet_Result_Message. As mensagens são polimórficas—utilize os descendentes específicos (..._Text, ..._Image, ..._Audio, ..._EmbeddedResource) para acessar o conteúdo subjacente e as anotações.

Código de exemplo

O exemplo abaixo carrega a lista de prompts, invoca um prompt e imprime as mensagens retornadas.

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    MCP->OnMCPListPrompts = MCPListPrompts;
    MCP->OnMCPResponsePrompt = MCPResponsePrompt;
}
void __fastcall TForm1::LoadPrompts()
{
    MCP->ListPrompts();
}
void __fastcall TForm1::MCPListPrompts(TObject *Sender,
const TsgcAI_MCP_Request_PromptsList &ARequest,
const TsgcAI_MCP_Response_PromptsList &AResponse)
{
    for (int LIndex = 0; LIndex < AResponse.Prompts->Count; LIndex++)
    {
        TsgcAI_MCP_Prompt *LPrompt = (TsgcAI_MCP_Prompt*)AResponse.Prompts->Item[LIndex];
        Memo1->Lines->Add(Format("%s: %s", ARRAYOFCONST((LPrompt->Name, LPrompt->Description))));
    }
}
void __fastcall TForm1::ExecutePrompt()
{
    TsgcJSON *LArgs = new TsgcJSON(nullptr);
    try
    {
        LArgs->AddPair("code", "ShowMessage('Hello World')");
        MCP->RequestPrompt("CodeReview", LArgs);
    }
    __finally
    {
        delete LArgs;
    }
}
```

```

void __fastcall TForm1::MCPResponsePrompt(TObject *Sender,
const TsgcAI_MCP_Request_PromptsGet &ARequest,
const TsgcAI_MCP_Response_PromptsGet &AResponse)
{
    Memo1->Lines->Add("Prompt description: " + AResponse.Result->Description);
    for (int LIndex = 0; LIndex < AResponse.Result->Messages->Count; LIndex++)
    {
        TsgcAI_MCP_Response_PromptsGet_Result_Message *LMessage =
            (TsgcAI_MCP_Response_PromptsGet_Result_Message*)AResponse.Result->Messages->Item[LIndex];
        if (dynamic_cast<TsgcAI_MCP_Response_PromptsGet_Result_Message_Text*>(LMessage))
        {
            auto *MsgText = (TsgcAI_MCP_Response_PromptsGet_Result_Message_Text*)LMessage;
            Memo1->Lines->Add(LMessage->Role + ": " + MsgText->Content.Text);
        }
        else
        {
            Memo1->Lines->Add(LMessage->Role + ": " + LMessage->Write());
        }
    }
}

```

MCP Client | Resources

As APIs de recursos do MCP permitem que o cliente descubra artefatos do lado do servidor e recupere seu conteúdo através do componente **TsgcWSAPIClient_MCP** uma vez que a sessão tenha sido inicializada.

Requisição ResourcesList

Use **ListResources** para obter o catálogo de recursos. O componente dispara **OnMCPListResources** após fazer parse da resposta JSON-RPC.

- **Request:** ListResources (sem parâmetros).
- **Event:** OnMCPListResources(Sender, Request, Response).
- **Response:** TsgcAI_MCP_Response_ResourcesList expondo uma fila Resources. Cada entrada TsgcAI_MCP_Resource inclui o Uri, Name, Title, Description e MimeType do recurso.

Requisição ResourcesRead

Chame **RequestResource(Uri)** para baixar um recurso específico. O componente gera **OnMCPResponseResource** com o payload decodificado.

- **Request:** RequestResource com a **Uri** de destino.
- **Evento:** OnMCPResponseResource(Sender, Request, Response).
- **Response:** TsgcAI_MCP_Response_ResourcesRead. Leia Result.Contents para acessar cada bloco TsgcAI_MCP_Response_Resources_ReadResult_Content, que contém o Uri, Name, Title, MimeType, além dos dados Text ou Blob.

Código de exemplo

Este trecho lista os recursos e recupera um único arquivo.

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    MCP->OnMCPListResources = MCPListResources;
    MCP->OnMCPResponseResource = MCPResponseResource;
}
void __fastcall TForm1::LoadResources()
{
    MCP->ListResources();
}
void __fastcall TForm1::MCPListResources(TObject *Sender,
const TsgcAI_MCP_Request_ResourcesList &ARequest,
const TsgcAI_MCP_Response_ResourcesList &AResponse)
{
    for (int LIndex = 0; LIndex < AResponse.Resources->Count; LIndex++)
    {
        TsgcAI_MCP_Resource *LResource =
            static_cast<TsgcAI_MCP_Resource*>(AResponse.Resources->Item[LIndex]);
        Memo1->Lines->Add(Format("%s -> %s", ARRAYOFCONST((LResource->Name, LResource->Uri))));
    }
}
void __fastcall TForm1::FetchResource(const String AUri)
{
    MCP->RequestResource(AUri);
}
void __fastcall TForm1::MCPResponseResource(TObject *Sender,
const TsgcAI_MCP_Request_ResourcesRead &ARequest,
const TsgcAI_MCP_Response_ResourcesRead &AResponse)
{
    for (int LIndex = 0; LIndex < AResponse.Result->Contents->Count; LIndex++)
    {
        TsgcAI_MCP_Response_Resources_ReadResult_Content *LContent =
            static_cast<TsgcAI_MCP_Response_Resources_ReadResult_Content*>(
                AResponse.Result->Contents->Item[LIndex]);
        if (LContent->Text != "")
            Memo1->Lines->Add(LContent->Uri + " : " + LContent->Text);
        else
    }
```

```
        Memo1->Lines->Add(LContent->Uri + " (binary, mime=" + LContent->MimeType + ")");  
    }
```

MCP Client | Roots

A especificação MCP introduz a capacidade **roots** para que um cliente possa descrever as localizações do sistema de arquivos que está disposto a expor a um servidor. Cada root é uma URI `file://` acompanhada de um nome de exibição opcional. Os servidores consultam o catálogo com `roots/list` e, quando suportado, recebem uma mensagem `notifications/roots/list_changed` sempre que a seleção muda. Esse handshake permite que as ferramentas respeitem os limites de projeto, workspaces e layouts de múltiplos repositórios, mantendo o usuário no controle do que o servidor pode acessar.

Entendendo roots

Roots atuam como guardrails que delimitam o escopo de recursos do lado do servidor. O cliente anuncia o suporte durante o initialize ao habilitar a capacidade roots e, opcionalmente, a flag `listChanged`, de modo que atualizações dinâmicas possam ser enviadas. Um payload típico retornado de `roots/list` contém uma ou mais entradas, cada uma com `Uri` e `Name`, representando workspaces, repositórios ou outras pastas selecionadas.

Tratando requisições roots/list

TsgcWSAPIClient_MCP analisa as mensagens JSON-RPC de entrada e gera **OnMCPListRoots** sempre que um servidor solicita o catálogo atual. Use o evento para sincronizar o modelo de workspace que sua aplicação mantém com a resposta do protocolo.

- **Request:** `TsgcAI_MCP_Request_RootsList`. A mensagem não inclui parâmetros, mas fornece o identificador JSON-RPC que você deve espelhar na resposta.
- **Event:** `OnMCPListRoots(Sender, Request, Response)`. Este é disparado depois que o componente valida a sessão.
- **Resposta:** `TsgcAI_MCP_Response_RootsList`. Chame `Roots.AddRoot(Uri, Name)` (ou construa instâncias de `TsgcAI_MCP_Root` manualmente) para descrever cada local acessível. O componente serializa o objeto preenchido assim que o seu handler retorna, então certifique-se de que a lista reflita o estado atual. Use `Response.Write` para diagnósticos ou registro.

Quando sua aplicação permite que os usuários adicionem ou removam workspaces em tempo de execução, lembre-se de gerar a notificação `notifications/roots/list_changed` correspondente para que os servidores atualizem seu catálogo em cache.

Código de exemplo

Os trechos a seguir publicam duas roots quando o servidor as solicita e registram o payload JSON que será enviado de volta.

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    MCP->OnMCPListRoots = MCPListRoots;
}
void __fastcall TForm1::MCPListRoots(TObject *Sender,
const TsgcAI_MCP_Request_RootsList *ARequest,
const TsgcAI_MCP_Response_RootsList *AResponse)
{
    AResponse->Roots->Clear();
    AResponse->Roots->AddRoot("file:///c:/projects/app", "Application Workspace");
    AResponse->Roots->AddRoot("file:///c:/projects/libs", "Shared Libraries");
    MemoLog->Lines->Add("Roots response: " + AResponse->Write());
}
```

MCP Client | Sampling

O sampling permite que servidores MCP deleguem gerações de modelos de linguagem ao cliente sem compartilhar credenciais de provedor. Um servidor emite requisições `sampling/createMessage` que encapsulam o estado da conversa, as preferências de modelo e restrições opcionais, como `maxTokens`. O cliente pode envolver um revisor humano, impor guard rails e, por fim, retornar uma resposta que inclui o modelo selecionado, o conteúdo gerado e o motivo de parada.

Entendendo a amostragem

Quando o recurso `sampling` é anunciado durante o `initialize`, os servidores são livres para solicitar gerações no meio do fluxo de trabalho. As requisições agrupam arrays de mensagens (blocos de texto, imagem ou áudio), um `systemPrompt`, limites de token opcionais e uma estrutura `ModelPreferences` que equilibra custo, velocidade e inteligência por meio de prioridades numéricas e strings de dica. O cliente é responsável por mapear essas dicas aos modelos disponíveis, solicitar a entrada do usuário quando necessário e filtrar conclusões inseguras antes de responder.

Tratando requisições `sampling/createMessage`

TsgcWSAPIClient_MCP expõe gerações iniciadas pelo servidor por meio de **OnMCPSamplingCreateMessage**. Seu handler deve examinar o contexto fornecido, rotear a chamada para o modelo desejado (ou revisor humano) e preencher o payload da resposta.

- **Request:** `TsgcAI_MCP_Request_SamplingCreateMessage`. Inspecione `Params.Messages` para reproduzir o diálogo, `Params.ModelPreferences` para priorização de hints e `Params.SystemPrompt` para guardrails.
- **Event:** `OnMCPSamplingCreateMessage(Sender, Request, Response)`. Gerado depois que o componente valida a sessão MCP e desserializa a requisição.
- **Response:** `TsgcAI_MCP_Response_SamplingCreateMessage`. Forneça o `Role` do assistente, atribua um objeto de conteúdo (texto, imagem, áudio ou link de recurso) e, opcionalmente, registre o `Model` e o `StopReason` concretos. Chame `Response.Write` para visualizar o payload JSON antes de seu envio.

Código de exemplo

Os trechos abaixo registram a contagem de mensagens recebidas, geram uma resposta fixa e anotam o modelo e o motivo de parada. Substitua o conteúdo fixo por uma invocação real de modelo ou um loop de aprovação humana em produção.

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    MCP->OnMCPSamplingCreateMessage = MCPSamplingCreateMessage;
}
void __fastcall TForm1::MCPSamplingCreateMessage(TObject *Sender,
const TsgcAI_MCP_Request_SamplingCreateMessage *ARequest,
const TsgcAI_MCP_Response_SamplingCreateMessage *AResponse)
{
    MemoLog->Lines->Add(Format("Sampling request with %d messages", ARRAYOFCOUNT(ARequest->Params->Messages->Content)));
    AResponse->Role = "assistant";
    auto *text = new TsgcAI_MCP_Response_Result_Content_Text();
    text->Text = "The capital of France is Paris.";
    AResponse->Content = text;
    AResponse->Model = "claude-3-sonnet-20240307";
    AResponse->StopReason = "endTurn";
    MemoLog->Lines->Add("Sampling response: " + AResponse->Write());
}
```

MCP Client | Elicitation

A Elicitation permite que um servidor solicite entrada estruturada do usuário por meio do cliente MCP. A requisição carrega uma mensagem legível por humanos e um JSON Schema restrito que define os campos esperados. Os clientes mantêm o controle da interação, validam os dados fornecidos e retornam uma ação que indica se o usuário aceitou, recusou ou cancelou o prompt.

Entendendo elicitation

A capacidade elicitation é negociada durante o initialize. Uma vez habilitados, os servidores podem enviar chamadas elicitation/create que embutem um schema de objeto plano limitado a tipos primitivos (string, number/integer, boolean e enum com nomes de exibição opcionais). Cada propriedade pode especificar metadados de exibição, restrições de comprimento ou intervalo e formatos suportados (email, uri, date, date-time). Os usuários devem sempre ser informados sobre quem está solicitando a informação, ter a oportunidade de editar o payload e poder recusar ou cancelar a requisição por completo.

Tratando requisições elicitation/create

TsgcWSAPIClient_MCP gera **OnMCPElicitationCreate** sempre que o servidor exige entrada adicional. O evento dá acesso à mensagem de prompt, ao schema solicitado e a um objeto de resposta vazio que você pode preencher com base nas decisões do usuário.

- **Request:** TsgcAI_MCP_Request_ElicitationCreate. Inspeione Params.Message para exibir um prompt e Params.RequestedSchema.Properties para construir controles de UI ou validações.
- **Evento:** OnMCPElicitationCreate(Sender, Request, Response). Acionado depois que o cliente deserializa o payload JSON-RPC.
- **Response:** TsgcAI_MCP_Response_ElicitationCreate. Defina Action como accept, decline ou cancel. Ao aceitar, escreva os valores coletados em Content (um helper TsgcJSON) para que correspondam ao esquema solicitado. Use Response.Write para inspecionar o documento JSON final.

Código de exemplo

O exemplo abaixo solicita ao usuário um handle do GitHub. Se o operador fornecer um valor, a resposta é aceita e o campo é retornado; caso contrário, a requisição é recusada.

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    MCP->OnMCPElicitationCreate = MCPElicitationCreate;
}
void __fastcall TForm1::MCPElicitationCreate(TObject *Sender,
const TsgcAI_MCP_Request_ElicitationCreate *ARequest,
const TsgcAI_MCP_Response_ElicitationCreate *AResponse)
{
    MemoLog->Lines->Add("Elicitation prompt: " + ARequest->Params->Message);
    String github = InputBox("GitHub", "Provide your GitHub username", "");
    if (!github.IsEmpty())
    {
        AResponse->Action = "accept";
        AResponse->Content->Clear();
        AResponse->Content->AddPair("name", github);
    }
    else
    {
        AResponse->Action = "decline";
    }
    MemoLog->Lines->Add("Elicitation response: " + AResponse->Write());
}
```


OpenAI

A OpenAI é um laboratório de pesquisa privado que visa desenvolver e direcionar a inteligência artificial (AI) de maneiras que beneficiem a humanidade como um todo. A OpenAI desenvolveu os seguintes projetos:

- **GPT-3:** Este poderoso modelo de linguagem serve como base para outros produtos da OpenAI. Ele analisa texto gerado por humanos para aprender a gerar texto semelhante por conta própria.
- **DALL-E e DALL-E 2:** Essas plataformas de IA generativa podem analisar descrições baseadas em texto de imagens que os usuários desejam que elas produzam e então gerar essas imagens exatamente como descritas.
- **CLIP:** O CLIP é uma rede neural que sintetiza imagens e textos relacionados a elas para prever as melhores legendas possíveis que descrevam com mais precisão essas imagens. Devido à sua capacidade de aprender com mais de um tipo de dado (tanto imagens quanto texto), ele pode ser categorizado como IA multimodal.
- **ChatGPT:** O ChatGPT é atualmente o chatbot de IA mais avançado, projetado para gerar texto semelhante ao humano e produzir respostas às perguntas dos usuários. Tendo sido treinado em grandes conjuntos de dados, ele pode gerar respostas da forma como um humano faria.
- **Codex:** O Codex foi treinado em bilhões de linhas de código em diversas linguagens de programação para ajudar os desenvolvedores de software a simplificar os processos de programação. Ele é baseado na tecnologia GPT-3, mas em vez de gerar texto, gera código.
- **Whisper:** O Whisper é uma ferramenta de reconhecimento automático de fala (ASR). Foi treinada em uma infinidade de dados de áudio para reconhecer, transcrever e traduzir fala em cerca de 100 idiomas diferentes, incluindo linguagem técnica e diferentes sotaques.

OpenAI API

A OpenAI API pode ser aplicada a praticamente qualquer tarefa que envolva compreender ou gerar linguagem natural, código ou imagens. A OpenAI oferece um espectro de modelos com diferentes níveis de potência adequados para diferentes tarefas, bem como a capacidade de fazer fine-tune de seus próprios modelos personalizados. Esses modelos podem ser usados para tudo, desde geração de conteúdo até busca semântica e classificação.

Most common uses

- **Completion**
 - [Exemplos de OpenAI Completion](#)
- **Chat**
 - [Exemplos de Chat da OpenAI](#)
- **Edit**
 - [Exemplos de OpenAI Edit](#)
- **Audio**
 - [Exemplos de Transcrição e Tradução OpenAI](#)
- **Moderation**
 - [Exemplos de OpenAI Moderation](#)
- **RealTime**
 - [Exemplos de OpenAI RealTime](#)
- **Responses**
 - [Exemplos de OpenAI Responses](#)
- **Speech**
 - [OpenAI Speech Examples](#)
- **Fine-Tuning**
 - [Exemplos de Fine-Tuning do OpenAI](#)

- Batch
 - [Exemplos de OpenAI Batch](#)
- Uploads
 - [Exemplos de Uploads do OpenAI](#)

Configuração

OpenAI

A API da OpenAI utiliza chaves de API para autenticação. Acesse sua página de [API Keys](#) para obter a chave de API que você utilizará em suas requisições.

Lembre-se de que sua chave de API é um segredo! Não a compartilhe com outras pessoas nem a exponha em qualquer código do lado do cliente (navegadores, aplicativos). As requisições de produção devem ser roteadas por meio do seu próprio servidor de backend, onde sua chave de API pode ser carregada com segurança a partir de uma variável de ambiente ou de um serviço de gerenciamento de chaves.

Esta **API Key** deve ser configurada na propriedade **OpenAIOptions.ApiKey** do componente. Opcionalmente, para usuários que pertencem a múltiplas organizações, você pode definir sua Organization na propriedade **OpenAIOptions.Organization** se sua conta pertencer a uma organização.

Uma vez que a API Key esteja configurada, veja abaixo uma lista de funções disponíveis para interagir com a API da OpenAI.

Azure

O cliente suporta os Microsoft Azure OpenAI Services, então você também pode usar sua conta Azure para interagir com a API do Azure OpenAI. Para configurar o cliente para trabalhar com o Azure, siga os passos abaixo:

1. Configure a propriedade **OpenAIOptions.Provider** = oapvAzure
2. Defina os valores de ResourceName e DeploymentId (esses valores podem ser localizados na sua Conta Azure)
 1. **OpenAIOptions.AzureOptions.ResourceName** = <your resource name>.
 2. **OpenAIOptions.AzureOptions.DeploymentId** = <your deployment id>.
3. Defina a API Key da sua Conta Azure
 1. **OpenAIOptions.ApiKey** = <azure api key>.

Tenha em mente que nem todos os métodos da OpenAI são suportados pelo Azure; atualmente, apenas os seguintes métodos são suportados:

1. Completion
2. Chat Completion

Properties

OpenAIOptions

- **ApiKey:** A API key para autenticação com a API da OpenAI.
- **Organization:** ID de organização opcional para usuários que pertencem a múltiplas organizações.
- **Provider:** Selecione o provedor: oapvOpenAI (padrão) ou oapvAzure para Microsoft Azure OpenAI Services.
- **AzureOptions:** Configuração para os Azure OpenAI Services.
 - **ResourceName:** O nome do recurso do Azure.
 - **DeploymentId:** O ID de deployment do Azure.
 - **APIVersion:** A versão da API do Azure.
- **HttpOptions:** configurações de conexão HTTP.
 - **ReadTimeout:** Timeout em milissegundos para a leitura das respostas HTTP. O padrão é 0 (sem timeout).
- **LogOptions:** Configuração de logging.
 - **Enabled:** Quando True, requisições e respostas HTTP são registradas em um arquivo.
 - **FileName:** O caminho do arquivo onde a saída do log é escrita.
- **RetryOptions:** Configuração de retry automático para requisições de API que falharam.
 - **Enabled:** Quando True, as requisições que falharam são repetidas automaticamente.
 - **Retries:** Número máximo de tentativas de retry.
 - **Wait:** Tempo de espera em milissegundos entre as tentativas de retry.

Modelos

Lista e descreve os vários modelos disponíveis na API.

- **GetModels:** Lista os modelos atualmente disponíveis e fornece informações básicas sobre cada um, como o proprietário e a disponibilidade.
- **GetModel:** Recupera uma instância de modelo, fornecendo informações básicas sobre o modelo, como o proprietário e o permissionamento.
 - **Model:** O ID do modelo a ser utilizado para esta requisição

Completions

Dado um prompt, o modelo retornará uma ou mais completions previstas e também pode retornar as probabilidades de tokens alternativos em cada posição.

- **CreateCompletion:** Cria uma completion para o prompt e os parâmetros fornecidos
 - **Model:** ID do modelo a utilizar. Você pode utilizar a API List models para ver todos os seus modelos disponíveis, ou consulte nossa Model overview para descrições deles.
 - **Prompt:** O prompt para gerar completions.

Chat

Dada uma conversa de chat, o modelo retornará uma resposta de chat completion.

- **Model:** ID do modelo a ser utilizado. Chame GetModels para obter uma lista de todos os modelos suportados pela Chat API.
- **Message:** A mensagem para a qual gerar as conclusões de chat.
- **Role:** por padrão user; outras opções são: system, assistant.

Edits

Dado um prompt e uma instrução, o modelo retornará uma versão editada do prompt.

- **CreateEdit:** Cria uma nova edição para a entrada, instrução e parâmetros fornecidos.
 - **Model:** ID do modelo a usar. Você pode usar o modelo text-davinci-edit-001 ou code-davinci-edit-001 com este endpoint.
 - **Instruction:** A instrução que diz ao modelo como editar o prompt.
 - **Input:** (opcional) O texto de entrada a usar como ponto de partida para a edição.

Images

Dado um prompt e/ou uma imagem de entrada, o modelo gerará uma nova imagem.

- **CreateImage:** Cria uma imagem dado um prompt.
 - **Prompt:** Uma descrição em texto da(s) imagem(ns) desejada(s). O comprimento máximo é de 1000 caracteres.
- **CreateImageEdit:** Cria uma imagem editada ou estendida a partir de uma imagem original e de um prompt.
 - **Image:** A imagem a ser editada. Deve ser um arquivo PNG válido, com menos de 4MB e quadrado. Se a mask não for fornecida, a imagem deve ter transparência, que será usada como a mask.
 - **Prompt:** Uma descrição textual da(s) imagem(ns) desejada(s). O comprimento máximo é de 1000 caracteres.
- **CreateImageVariations:** Cria uma variação de uma imagem informada.
 - **Image:** A imagem a ser utilizada como base para a(s) variação(ões). Deve ser um arquivo PNG válido, com menos de 4MB e quadrado.

Embeddings

Obtém uma representação vetorial de uma determinada entrada que pode ser facilmente consumida por modelos e algoritmos de machine learning.

- **CreateEmbeddings:** Cria um vetor de embedding representando o texto de entrada.
 - **Model:** ID do modelo a utilizar.
 - **Input:** Texto de entrada para o qual obter embeddings.

Audio

Transforma Áudio em Texto.

- **CreateTranscriptionFromFile:** Transcreve áudio no idioma de entrada a partir de um nome de arquivo
 - **Model:** ID do modelo a utilizar. Apenas whisper-1 está disponível no momento.
 - **Filename:** O arquivo de áudio a transcrever, em um destes formatos: mp3, mp4, mpeg, mpga, m4a, wav ou webm.
- **CreateTranscription:** Grava áudio por X segundos e o transcreve.
 - **Model:** ID do modelo a ser utilizado. Apenas whisper-1 está disponível atualmente.
 - **Time:** tempo em milissegundos, por padrão 10 segundos.
- **CreateTranslationFromFile:** Traduz áudio para o inglês.
 - **Model:** ID do modelo a ser utilizado. Apenas whisper-1 está disponível no momento.
 - **Filename:** O arquivo de áudio a ser traduzido, em um destes formatos: mp3, mp4, mpeg, mpga, m4a, wav ou webm.
- **CreateTranslation:** Grava áudio por X segundos e o traduz.
 - **Model:** ID do modelo a usar. Apenas whisper-1 está disponível atualmente.
 - **Time:** tempo em milissegundos, por padrão 10 segundos.

Files

Files são usados para fazer upload de documentos que podem ser usados com recursos como Fine-tuning.

- **ListFiles:** Retorna uma lista de arquivos que pertencem à organização do usuário.
- **UploadFile:** Faz upload de um arquivo que contém documento(s) a serem utilizados em vários endpoints/recursos. Atualmente, o tamanho de todos os arquivos enviados por uma organização pode chegar a 1 GB.
 - **Filename:** Nome do arquivo JSON Lines a ser enviado. Se a finalidade estiver definida como "fine-tune", cada linha é um registro JSON com os campos "prompt" e "completion" que representam seus exemplos de treinamento.
 - **Purpose:** A finalidade pretendida dos documentos enviados. Utilize "fine-tune" para Fine-tuning.
- **DeleteFile:** Exclui um arquivo.
 - **FileId:** O ID do arquivo a ser utilizado nesta requisição
- **RetrieveFile:** Retorna informações sobre um arquivo específico.
 - **FileId:** O ID do arquivo a ser utilizado nesta requisição
- **RetrieveFileContent:** Retorna o conteúdo do arquivo especificado
 - **FileId:** O ID do arquivo a ser usado para esta requisição.

Fine-Tunes

Gerencie jobs de fine-tuning para adaptar um modelo aos seus dados de treinamento específicos.

- **CreateFineTune:** Cria um job que ajusta um modelo especificado a partir de um conjunto de dados fornecido. A resposta inclui detalhes do job enfileirado, incluindo o status do job e o nome dos modelos ajustados quando concluído.
 - **TrainingFile:** O ID de um arquivo enviado que contém os dados de treinamento.
- **ListFineTunes:** Lista os jobs de fine-tuning da sua organização

- **RetrieveFineTune:** Obtém informações sobre o fine-tune job.
 - **FineTuneId:** O ID do fine-tune job
- **CancelFineTune:** Cancela imediatamente um job de fine-tune.
 - **FineTuneId:** O ID do job de fine-tune
- **ListFineTuneEvents:** Obtém atualizações de status detalhadas para um job de fine-tune.
 - **FineTuneId:** O ID do job de fine-tune
- **DeleteFineTuneModel:** Exclui um modelo fine-tuned. Você deve ter o papel de Owner em sua organização.
 - **Model:** O modelo a excluir.

Moderations

Dado um texto de entrada, indica se o modelo o classifica como uma violação da política de conteúdo da OpenAI.

- **CreateModeration:** Classifica se o texto viola a Política de Conteúdo da OpenAI
 - **Input:** O texto de entrada a classificar

RealTime

A OpenAI Realtime API permite interações multimodais de baixa latência, incluindo experiências conversacionais speech-to-speech e transcrição em tempo real.

- **Transcription:** Você pode utilizar a Realtime API para casos de uso somente de transcrição, seja com entrada de um microfone ou de um arquivo. Por exemplo, você pode utilizá-la para gerar legendas ou transcrições em tempo real. No modo somente de transcrição, o modelo não gera respostas.

Assistants

Construa assistentes de IA que podem chamar modelos e usar tools para realizar tarefas.

- **CreateAssistant:** Cria um assistente com um modelo e instruções.
- **ListAssistants:** Retorna uma lista de assistants.
- **RetrieveAssistant:** Recupera um assistant pelo ID.
- **ModifyAssistant:** Modifica um assistant existente.
- **DeleteAssistant:** Exclui um assistant.

Threads

Os Threads representam uma sessão de conversa. As mensagens são adicionadas aos threads, que então são processados por runs.

- **CreateThread:** Cria um novo thread.
- **RetrieveThread:** Recupera um thread por ID.
- **ModifyThread:** Modifica um thread.
- **DeleteThread:** Exclui um thread.

Thread Messages

As mensagens são adicionadas aos threads e contêm o conteúdo de uma conversa.

- **CreateMessage:** Cria uma mensagem dentro de um thread.
 - **ThreadId:** O ID do thread.
- **ListMessages:** Retorna uma lista de mensagens de uma determinada thread.

- **RetrieveMessage:** Recupera uma mensagem pelos IDs de thread e de mensagem.
- **ModifyMessage:** Modifica uma mensagem.
- **DeleteMessage:** Exclui uma mensagem de um thread.

Runs

Os Runs representam uma execução em um thread com um assistant. O assistant usa sua configuração e as mensagens do thread para realizar tarefas chamando modelos e ferramentas.

- **CreateRun:** Cria um run para uma thread com um assistant especificado.
- **ListRuns:** Retorna uma lista de runs pertencentes a um thread.
- **RetrieveRun:** Recupera um run pelos IDs de thread e de run.
- **ModifyRun:** Modifica uma run.
- **SubmitToolOutputsToRun:** Envia as saídas das ferramentas para um run que requer ação.
- **CancelRun:** Cancela um run em andamento.

Run Steps

Os run steps representam as etapas individuais realizadas durante a execução de um run.

- **ListRunSteps:** Retorna uma lista de run steps pertencentes a um run.
- **RetrieveRunSteps:** Recupera um run step por thread, run e step ID.

Vector Stores

Os vector stores são usados para armazenar e pesquisar arquivos usando embeddings para retrieval-augmented generation (RAG).

- **CreateVectorStore:** Cria um vector store.
- **ListVectorStores:** Retorna uma lista de vector stores.
- **RetrieveVectorStore:** Recupera um vector store por ID.
- **ModifyVectorStore:** Modifica um vector store.
- **DeleteVectorStore:** Exclui um vector store.

Vector Store Files

Gerencia arquivos dentro de vector stores.

- **CreateVectorStoreFile:** Anexa um arquivo a um vector store.
- **ListVectorStoreFiles:** Retorna uma lista de arquivos em um vector store.
- **RetrieveVectorStoreFile:** Recupera um arquivo de vector store.
- **DeleteVectorStoreFile:** Remove um arquivo de um vector store.

Vector Store File Batches

Operações de batch para adicionar arquivos a vector stores.

- **CreateVectorStoreFileBatch:** Cria um batch para adicionar múltiplos arquivos a um vector store.
- **RetrieveVectorStoreFileBatch:** Recupera um lote de arquivos pelo ID.
- **CancelVectorStoreFileBatch:** Cancela um lote de arquivos em andamento.
- **ListVectorStoreFilesBatch:** Lista os arquivos em um batch.

Speech

Gera áudio falado a partir de texto utilizando modelos text-to-speech.

- **CreateSpeech:** Gera áudio a partir do texto de entrada. Retorna os dados de áudio como string ou escreve em um response stream.
 - **Model:** O modelo TTS a utilizar (por exemplo tts-1, tts-1-hd).
 - **Input:** O texto para o qual gerar áudio.
 - **Voice:** A voz a utilizar (alloy, echo, fable, onyx, nova, shimmer).

Fine-Tuning Jobs

Gerencia jobs de fine-tuning para criar modelos personalizados. Esta é a API de fine-tuning mais recente que substitui o endpoint legado de Fine-Tunes.

- **CreateFineTuningJob:** Cria um job de fine-tuning a partir de um arquivo de treinamento.
- **ListFineTuningJobs:** Lista os fine-tuning jobs da sua organização.
- **RetrieveFineTuningJob:** Recupera informações sobre um job de fine-tuning.
- **CancelFineTuningJob:** Cancela um job de fine-tuning.
- **ListFineTuningJobEvents:** Lista as atualizações de status de um job de fine-tuning.
- **ListFineTuningJobCheckpoints:** Lista os checkpoints de um job de fine-tuning.

Responses

Cria e gerencia respostas de modelo. A Responses API suporta conversas com múltiplos turnos, uso de ferramentas e saídas estruturadas.

- **CreateResponse:** Cria uma resposta de modelo.
- **RetrieveResponse:** Recupera uma resposta por ID.
- **DeleteResponse:** Exclui uma resposta.
- **CancelResponse:** Cancela uma resposta em andamento.
- **ListInputItems:** Lista os input items de uma resposta.

Batches

Cria e gerencia requisições de batch API para processamento assíncrono a um custo menor.

- **CreateBatch:** Cria um batch job.
- **RetrieveBatch:** Recupera um lote por ID.
- **ListBatches:** Lista todos os batches.
- **CancelBatch:** Cancela um batch em andamento.

Uploads

Faz upload de arquivos grandes em partes. Útil para arquivos que excedem o limite padrão de tamanho de upload.

- **CreateUpload:** Cria uma sessão de upload.
- **AddUploadPart:** Adiciona uma parte a um upload.
- **CompleteUpload:** Conclui um upload multipart e cria um arquivo.
- **CancelUpload:** Cancela um upload em andamento.

TsgcHTTP_API_OpenAI

Cliente REST da API da OpenAI cobrindo Chat Completions, Images, Audio (TTS/STT), Embeddings, Files, Fine-tuning, Moderation, Assistants, Threads, Runs, Messages e Vector Stores.

Introdução

Cliente REST da API da OpenAI cobrindo Chat Completions, Images, Audio (TTS/STT), Embeddings, Files, Fine-tuning, Moderation, Assistants, Threads, Runs, Messages e Vector Stores.

Introdução

Cliente REST da API da OpenAI cobrindo Chat Completions, Images, Audio (TTS/STT), Embeddings, Files, Fine-tuning, Moderation, Assistants, Threads, Runs, Messages e Vector Stores.

Configure a propriedade **OpenAIOptions** com sua chave de API e configurações opcionais de HTTP/retry/log, depois chame os métodos tipados como `CreateMessage/CreateChatCompletion` com um objeto de requisição e receba uma classe de resposta tipada.

Para contexto sobre o provedor, consulte o guia [OpenAI](#).

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Métodos — 78 métodos públicos.
- Events — 2 eventos que você pode tratar.

OpenAI | Completion

Dado um prompt, o modelo retornará uma ou mais completions previstas, e também pode retornar as probabilidades de tokens alternativos em cada posição.

Exemplo Simples

Utilize o modelo text-davinci-003 para obter um completion previsto.

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";
ShowMessage(OpenAI->_CreateCompletion("text-davinci-003", "Say this is a test"));
```

Exemplo avançado

Use o modelo text-davinci-003 para obter um completion previsto com saída mais aleatória e gerar 2 completions para cada prompt.

```
TsgcHTTP_OpenAI_JSON *OpenAI = new TsgcHTTP_OpenAI_JSON(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";

TsgcOpenAIClass_Request_Completion *oRequest = new TsgcOpenAIClass_Request_Completion();
try
{
    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 (Length(oResponse->Choices) > 0)
    {
        ShowMessage(oResponse->Choices[0]->Text);
    }
}
finally
{
    delete oRequest;
    delete oResponse;
}
```

OpenAI | Chat

Dada uma conversa de chat, o modelo retornará uma resposta de chat completion.

Exemplo Simples

Interaja com o ChatGPT enviando uma mensagem Hello.

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";
ShowMessage(OpenAI->_CreateChatCompletion("gpt-3.5-turbo", "Hello!"));
```

Exemplo avançado

Utilize o modelo gpt-3-5 para conversar com saída mais aleatória e gerar 2 completions para cada prompt.

```
TsgcHTTP_OpenAI_JSON *OpenAI = new TsgcHTTP_OpenAI_JSON(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";

TsgcOpenAIClass_Request_ChatCompletion *oRequest = new TsgcOpenAIClass_Request_ChatCompletion();
try
{
    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 (Length(oResponse->Choices) > 0)
    {
        ShowMessage(oResponse->Choices[0]->_Message->Content);
    }
}
__finally
{
    delete oRequest;
    delete oResponse;
}
```

OpenAI | Edit

Dado um prompt e uma instrução, o modelo retornará uma versão editada do prompt.

Exemplo Simples

Pede à OpenAI para corrigir os erros de ortografia de um prompt.

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";
ShowMessage(OpenAI->_CreateEdit("text-davinci-edit-001", "Fix the spelling mistakes", "What day of the wek is it?"));
```

Exemplo avançado

Diga à OpenAI para corrigir os erros de ortografia de um prompt, com saída mais aleatória, e gere 2 conclusões para cada prompt.

```
TsgcHTTP_OpenAI_JSON *OpenAI = new TsgcHTTP_OpenAI_JSON(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";

TsgcOpenAIClass_Request_Edit *oRequest = new TsgcOpenAIClass_Request_Edit();
try
{
    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 (Length(oResponse->Choices) > 0)
    {
        ShowMessage(oResponse->Choices[0]->Text);
    }
}
__finally
{
    delete oRequest;
    delete oResponse;
}
```

OpenAI | Áudio

Create Transcription

Transcreve o áudio para o idioma de entrada.

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);  
OpenAI->OpenAIOptions->ApiKey = "API_KEY";  
  
ShowMessage(OpenAI->_CreateTranscriptionFromFile("whisper-1", "c:\\media\\audio.mp3"));
```

Create Translation

Traduz áudio para o inglês.

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);  
OpenAI->OpenAIOptions->ApiKey = "API_KEY";  
  
ShowMessage(OpenAI->_CreateTranslationFromFile("whisper-1", "c:\\media\\audio.mp3"));
```

OpenAI | Moderation

Dado um texto de entrada, retorna se o modelo o classifica como violando a política de conteúdo da OpenAI.

Exemplo Simples

Modere o texto a seguir

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";

ShowMessage(OpenAI->_CreateModeration("I want to kill them."));
```

Exemplo avançado

Modera o texto a seguir escolhendo o modelo.

```
TsgcHTTP_OpenAI_JSON *OpenAI = new TsgcHTTP_OpenAI_JSON(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";

TsgcOpenAIClass_Request_Moderation *oRequest = new TsgcOpenAIClass_Request_Moderation();
try
{
    oRequest->Model = "text-moderation-latest";
    oRequest->Input = "I want to kill them.";
    TsgcOpenAIClass_Response_Moderation *oResponse = OpenAI->CreateModeration(oRequest);

    if (Length(oResponse->results) > 0)
    {
        ShowMessage(oResponse->results[0]->flagged);
    }
}
__finally
{
    delete oRequest;
    delete oResponse;
}
```

OpenAI | RealTime

Você pode usar a Realtime API para casos de uso somente de transcrição, seja com entrada de um microfone ou de um arquivo. Por exemplo, você pode usá-la para gerar legendas ou transcrições em tempo real. Com o modo somente de transcrição, o modelo não gerará respostas.

Para utilizar a Realtime API para transcrição, você precisa criar uma sessão de transcrição, conectando via WebSockets. Utilize o componente **TsgcWSAPI_OpenAI** e **TsgcWebSocketClient** para iniciar uma nova sessão de transcrição.

Encontre abaixo um exemplo de transcrição em tempo real usando a OpenAI API.

```
TsgcWebSocketClient *WClient = new TsgcWebSocketClient(NULL);
TsgcAudioRecorderWave *oAudio = new TsgcAudioRecorderWave(NULL);
TsgcWSAPI_OpenAI *OpenAI = new TsgcWSAPI_OpenAI(NULL);
OpenAI->Client = WClient;
OpenAI->AudioRecorder = oAudio;
OpenAI->OpenAIOptions->APIKey = "your-api-key-here";
OpenAI->OpenAIOptions->Method = sgcoaimTranscription;
OpenAI->OpenAIOptions->Provider = sgcoaipOpenAI;
OpenAI->InputAudio->Language = "en";
OpenAI->InputAudio->Model = "whisper-1";
void __fastcall TForm1::OnOpenAIAudioTranscriptionCompleted(TObject *Sender, TsgcWSOpenAIConversation_Item_Comple
    Log("#transcription_completed: " + aItem->Transcript);
}
```

Send Audio Manually

O componente permite que você envie arquivos de áudio manualmente. Você pode utilizar o método `AppendInputAudioBuffer` e passar o áudio como um `TStream`. O formato de áudio deve ser PCM mono de 24 kHz (apenas uma taxa de 24000 é suportada).

OpenAI | Responses

A Responses API é a API central para interagir com os modelos da OpenAI. Ela suporta entradas de texto e imagem, uso de ferramentas, streaming e saídas estruturadas.

Exemplo Simples

Cria uma resposta utilizando a Responses API.

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";
ShowMessage(OpenAI->_CreateResponse("gpt-4o", "What is the capital of France?"));
```

Métodos

- **CreateResponse:** Cria uma resposta de modelo dado um modelo e uma entrada.
 - **Model:** ID do modelo a usar (por exemplo gpt-4o, gpt-4o-mini).
 - **Input:** Texto ou entrada estruturada para o modelo.
- **RetrieveResponse:** Recupera uma resposta previamente criada pelo seu ID.
 - **ResponseId:** O ID da resposta a recuperar.
- **DeleteResponse:** Exclui uma resposta pelo seu ID.
 - **ResponseId:** O ID da resposta a excluir.
- **CancelResponse:** Cancela uma resposta em andamento.
 - **ResponseId:** O ID da resposta a cancelar.
- **ListInputItems:** Retorna uma lista de input items para uma determinada response.
 - **ResponseId:** O ID da response.

OpenAI | Speech

Gera áudio falado a partir de texto usando a Text-to-Speech (TTS) API. Suporta múltiplas vozes e formatos de saída.

Exemplo Simples

Gera fala a partir de texto e salva em um stream de arquivo.

```
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
{
    delete oStream;
}
```

Métodos

- **CreateSpeech:** Gera áudio a partir do texto de entrada.
 - **Model:** ID do modelo a ser utilizado (tts-1 ou tts-1-hd).
 - **Input:** O texto para o qual gerar áudio. Máximo de 4096 caracteres.
 - **Voice:** A voz a ser utilizada (alloy, echo, fable, onyx, nova, shimmer).
 - **ResponseStream:** O stream onde os dados de áudio serão gravados.
 - **ResponseFormat:** (opcional) O formato de áudio: mp3 (padrão), opus, aac ou flac.
 - **Speed:** (opcional) A velocidade do áudio gerado (0.25 a 4.0, padrão 1.0).

OpenAI | Fine-Tuning

Gerencie jobs de fine-tuning para adaptar um modelo aos seus dados de treinamento específicos. O fine-tuning permite que você aproveite mais os modelos fornecendo resultados de maior qualidade, a capacidade de treinar com mais exemplos e economia de custos.

Exemplo Simples

Cria um job de fine-tuning e lista os jobs existentes.

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";

// Create a fine-tuning job
ShowMessage(OpenAI->_CreateFineTuningJob("gpt-4o-mini-2024-07-18", "file-abc123"));

// List fine-tuning jobs
ShowMessage(OpenAI->_ListFineTuningJobs());
```

Métodos

- **CreateFineTuningJob:** Cria um job de fine-tuning que inicia o processo de criação de um novo modelo a partir de um conjunto de dados fornecido.
 - **Model:** O nome do modelo a ser ajustado.
 - **TrainingFile:** O ID de um arquivo enviado que contém os dados de treinamento.
- **ListFineTuningJobs:** Lista os jobs de fine-tuning da sua organização.
- **RetrieveFineTuningJob:** Obtém informações sobre um job de fine-tuning.
 - **JobId:** O ID do job de fine-tuning.
- **CancelFineTuningJob:** Cancela imediatamente um job de fine-tuning.
 - **JobId:** O ID do job de fine-tuning a ser cancelado.
- **ListFineTuningJobEvents:** Obtém atualizações de status para um fine-tuning job.
 - **JobId:** O ID do fine-tuning job.
- **ListFineTuningJobCheckpoints:** Lista os checkpoints de um job de fine-tuning.
 - **JobId:** O ID do job de fine-tuning.

OpenAI | Batch

Cria grandes lotes de requisições de API para executar de forma assíncrona. A Batch API retorna as completions em até 24 horas com 50% de desconto.

Exemplo Simples

Cria um job de batch e recupera seu status.

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";

// Create a batch
ShowMessage(OpenAI->CreateBatch("file-abc123", "/v1/chat/completions"));

// List batches
ShowMessage(OpenAI->_ListBatches());
```

Métodos

- **CreateBatch:** Cria e executa um batch a partir de um arquivo de requisições enviado.
 - **InputFileId:** O ID de um arquivo enviado que contém as requisições para o batch.
 - **Endpoint:** O endpoint a ser utilizado para todas as requisições no batch (por exemplo /v1/chat/completions).
 - **CompletionWindow:** (opcional) O intervalo de tempo dentro do qual o batch deve ser processado. O padrão é 24h.
- **RetrieveBatch:** Recupera um lote (batch) pelo seu ID.
 - **BatchId:** O ID do lote a recuperar.
- **ListBatches:** Lista os batches da sua organização.
- **CancelBatch:** Cancela um lote em andamento.
 - **BatchId:** O ID do lote a cancelar.

OpenAI | Uploads

Faça upload de arquivos grandes em partes. A Uploads API permite que você faça upload de arquivos maiores que 512 MB dividindo-os em partes e fazendo upload de cada parte separadamente.

Exemplo Simples

Cria um upload, adiciona uma parte e o completa.

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";

// Create an upload
AnsiString vUploadResponse = OpenAI->_CreateUpload("training_data.jsonl",
    "fine-tune", 2147483648, "application/jsonl");

// Add a part
AnsiString vPartResponse = OpenAI->_AddUploadPart("upload_abc123", "part1.jsonl");

// Complete the upload
ShowMessage(OpenAI->_CompleteUpload("upload_abc123", "[\"part_abc123\"]"));
```

Métodos

- **CreateUpload:** Cria um objeto Upload intermediário para iniciar um upload multi-part.
 - **Filename:** O nome do arquivo a ser enviado.
 - **Purpose:** O propósito pretendido do arquivo enviado (por exemplo, fine-tune).
 - **Bytes:** O número de bytes no arquivo.
 - **MimeType:** O tipo MIME do arquivo (por exemplo, application/jsonl).
- **AddUploadPart:** Adiciona uma parte a um objeto Upload.
 - **UploadId:** O ID do Upload.
 - **Filename:** O bloco de bytes para esta parte.
- **CompleteUpload:** Conclui o Upload. O arquivo enviado só é utilizável após a conclusão.
 - **UploadId:** O ID do Upload.
 - **PartIds:** A lista ordenada de IDs de Part como uma string de array JSON.
- **CancelUpload:** Cancela o Upload. Nenhuma parte pode ser adicionada após o cancelamento.
 - **UploadId:** O ID do Upload a ser cancelado.

OpenAI Applications

Visão geral

Utilizando a API da OpenAI, você pode criar uma ampla variedade de aplicações; aqui estão alguns exemplos:

- **Chatbots e assistentes virtuais:** Aplicações que podem conversar com humanos de maneira natural, semelhante à humana. Estas podem ser usadas para suporte ao cliente, tratamento de consultas e fornecimento de informações em um site ou aplicativo móvel.
- **Geração de Conteúdo:** Aplicações que podem gerar texto semelhante ao humano, como artigos, posts de blog ou relatórios. Por exemplo, o GPT-3 pode ser utilizado para automatizar a criação de conteúdo para redes sociais, gerar código ou criar conteúdo otimizado para SEO.
- **Serviços de Tradução:** Aplicações que podem traduzir texto de um idioma para outro.
- **Tutoria e Educação:** a IA pode ser utilizada para criar experiências de aprendizado personalizadas, ajudar com tarefas escolares ou explicar conceitos complexos em linguagem simples.
- **Games:** A OpenAI pode ser utilizada para criar jogos imersivos e interativos, especialmente aqueles que envolvem personagens conversacionais ou narrativas complexas.
- **Análise de Sentimentos:** Analisar e categorizar os sentimentos expressos em dados textuais pode ser útil para pesquisa de mercado, monitoramento de marca e compreensão do feedback dos clientes.
- **Recomendações Personalizadas:** Com base nos comportamentos e preferências passados dos usuários, a IA pode gerar recomendações personalizadas de produtos, serviços ou conteúdo.
- **Text Completion:** Completar frases do usuário ou ajudar na assistência de escrita em clientes de e-mail ou software de processamento de texto.
- **Speech Recognition:** Transcreve linguagem falada em texto escrito, útil em serviços de transcrição, assistentes de voz e muito mais.
- **Consultoria Médica e Jurídica:** Embora não sejam capazes de substituir o aconselhamento profissional, os modelos de IA podem fornecer orientações preliminares ou sugestões com base nas entradas fornecidas.

Componentes

Veja abaixo uma lista dos componentes disponíveis e uma breve descrição sobre eles.

- **TsgcAIOpenAIAssistant:** Um Assistant tem instruções e pode aproveitar modelos, ferramentas e arquivos para responder às consultas do usuário.
- **TsgcAIOpenAIChatBot:** um ChatBot que escuta a fala e a converte em texto usando a OpenAI Whisper API. Este texto é enviado à ChatCompletion API, que fornece uma resposta da OpenAI, e essa resposta é convertida de texto em fala.
- **TsgcAIOpenAITranslator:** uma aplicação de Tradução que permite traduzir a fala de qualquer idioma para inglês e ouvir a tradução usando qualquer um dos componentes TextToSpeech disponíveis.
- **TsgcAIOpenAIEmbeddings:** este componente é utilizado para construir aplicações com IA com dados personalizados, exemplo: um chatbot com os dados do nosso produto. Os seguintes bancos de dados são suportados:
 - **TsgcAIDatabaseVectorPinecone:** suporta o Pinecone Database Vector.

Os seguintes componentes são usados para capturar áudio do microfone, reproduzir arquivos de áudio e converter texto em fala.

- **TsgcAudioRecorderMCI:** (apenas para Windows) este componente permite que você acesse o microfone e converta a fala em um arquivo wave.
- **TsgcAudioPlayerMCI:** (apenas para Windows) este componente permite que você reproduza um arquivo mp3.
- **TsgcTextToSpeechSystem:** (apenas para Windows) converte texto em fala sem a necessidade de utilizar um arquivo mp3 externo.
- **TsgcTextToSpeechGoogle:** converte texto em fala usando qualquer uma das vozes do Google Cloud disponíveis.
- **TsgcTextToSpeechAmazon:** converte texto em fala utilizando qualquer uma das vozes do Amazon AWS disponíveis.

TsgcAIOpenAIAssistant

A Assistants API permite que você construa assistentes de IA dentro de suas próprias aplicações.

Introdução

A Assistants API permite que você construa assistentes de IA dentro de suas próprias aplicações. Um Assistant possui instruções e pode aproveitar modelos, ferramentas e arquivos para responder às consultas dos usuários. A Assistants API atualmente suporta três tipos de ferramentas: Code Interpreter, File Search e chamada de função.

Visão geral

Uma integração típica da Assistants API tem o seguinte fluxo:

- Crie um Assistant definindo suas instruções personalizadas e escolhendo um modelo. Se for útil, adicione arquivos e habilite ferramentas como Code Interpreter, File Search e Function calling.
- Cria uma Thread quando um usuário inicia uma conversa.
- Adicione Messages ao Thread à medida que o usuário faz perguntas.
- Executa o Assistant na Thread para gerar uma resposta chamando o modelo e as ferramentas.

Passo 1: Crie um Assistant

Um Assistant representa uma entidade que pode ser configurada para responder às mensagens de um usuário utilizando vários parâmetros, como modelo, instruções e ferramentas.

```
// Create a new assistant
TsgcAIOpenAIAssistant *oAssistant = new TsgcAIOpenAIAssistant(NULL);
// Set your API key
oAssistant->OpenAIOptions->ApiKey = txtAPIKey->Text;
// Assistant options
oAssistant->AssistantOptions->Name = "Math Tutor";
oAssistant->AssistantOptions->Instructions->Text = "You are a personal math tutor. Write and run code to answer questions";
oAssistant->AssistantOptions->Model = "gpt-4o";
// Create the assistant
oAssistant->CreateAssistant();
```

Passo 2: Criar um Thread

Um Thread representa uma conversa entre um usuário e um ou vários Assistants. Você pode criar um Thread quando um usuário (ou sua aplicação de IA) inicia uma conversa com o seu Assistant.

```
oThread = oAssistant->CreateThread();
```

Passo 3: Adicionar uma Mensagem à Thread e Executar (Run)

O conteúdo das mensagens que seus usuários ou aplicações criam é adicionado como objetos Message ao Thread. As mensagens podem conter tanto texto quanto arquivos. Não há limite para o número de Messages que você pode adicionar aos Threads — truncamos de forma inteligente qualquer contexto que não caiba na janela de contexto do modelo.

Uma vez que todas as Messages do usuário tenham sido adicionadas ao Thread, você pode executar o Thread (Run) com qualquer Assistant. Criar um Run utiliza o modelo e as ferramentas associadas ao Assistant para gerar uma resposta. Essas respostas são adicionadas ao Thread como Messages do assistant.

```
void SendMessage(TsgcAIOpenAIAssistant *oAssistant, TsgcAIClass_Thread *oThread, const UnicodeString aMessage)
{
    int i;
    TsgcOpenAIClass_Message *oMessage;
    TsgcOpenAIClass_Response_List_Messages *oMessages;
    TsgcOpenAIClass_Run *oRun;
    DoLog(L"[user]: " + aMessage);
    oMessage = oAssistant->CreateMessageText(oThread->Id, aMessage);
    if (oMessage != NULL)
    {
        oRun = oAssistant->CreateRunAndWait(oThread->Id);
        if (oRun != NULL)
        {
            oMessages = oAssistant->GetMessages(oThread->Id, oRun->Id);
            if (oMessages != NULL && oMessages->Messages.Length > 0)
            {
                for (i = 0; i < oMessages->Messages.Length; i++)
                {
                    DoLog(L"[assistant]: " + DoFormatResponse(oMessages->Messages[i]->ContentText + L"\r\n"));
                }
            }
        }
    }
}
```

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Métodos — 9 métodos públicos.
- Eventos — 8 eventos que você pode tratar.

TsgcAIOpenAIAssistant | File Search

O File Search aprimora as capacidades do Assistant integrando conhecimento de fontes externas, como informações proprietárias de produtos ou documentos fornecidos pelos usuários. Veja como funciona:

- **Processamento de Documentos:** A OpenAI processa e segmenta automaticamente (ou divide em "chunks") seus documentos em partes gerenciáveis de informação.
- **Criação de Embedding:** Esses blocos são transformados em embeddings — representações matemáticas do conteúdo — que capturam seu significado e contexto.
- **Armazenamento e Busca:** O sistema armazena esses embeddings, habilitando tanto a busca vetorial (para encontrar conteúdo semanticamente similar) quanto a busca por palavra-chave (para localizar correspondências exatas).

Essa combinação garante que o Assistant possa recuperar e aproveitar conteúdo relevante para fornecer respostas precisas e fundamentadas às consultas do usuário.

Passo 1: Criar um novo Assistant com File Search habilitado

Crie um novo assistant com `file_search` habilitado no parâmetro `tools` do Assistant. Uma vez que a ferramenta `file_search` é habilitada, o modelo decide quando recuperar conteúdo com base nas mensagens do usuário.

```
TsgcAIOpenAIAssistant* Assistant = new TsgcAIOpenAIAssistant(nullptr);
Assistant->OpenAIOptions->ApiKey = "sk-askdjfalskdjfl23kjkjasdefasdfj";
Assistant->AssistantOptions->Name = "sgcWebSockets HelpDesk";
Assistant->AssistantOptions->Instructions->Text =
    "You are a sgcWebSockets HelpDesk Agent. "
    "Answer questions briefly, in a sentence or less. When asked a question, use the manual to answer the question";
Assistant->AssistantOptions->Model = "gpt-4o-mini";
Assistant->AssistantOptions->Tools->FileSearch->Enabled = true;
Assistant->AssistantOptions->Tools->CodeInterpreter->Enabled = false;
```

Passo 2: Faça o upload dos arquivos e adicione-os a um Vector Store

Para acessar seus arquivos, a ferramenta `file_search` usa o objeto Vector Store. Faça o upload dos seus arquivos e crie um Vector Store para contê-los.

```
void UploadFile()
{
    TOpenDialog* oDialog = new TOpenDialog(nullptr);
    try
    {
        if (oDialog->Execute())
        {
            Screen->Cursor = crHourGlass; // Change cursor to hourglass
            try
            {
                Assistant->UploadVectorStoreFile("sgcVectorStore", oDialog->FileName.c_str());
            }
            finally
            {
                Screen->Cursor = crDefault; // Reset cursor to default
            }
        }
    }
}
```

```

    _finally
    {
        delete oDialog; // Clean up dialog
    }
}

```

Passo 3: Crie um run e verifique a saída

Agora, crie um Run e observe que o modelo usa a tool File Search para fornecer uma resposta à pergunta do usuário.

```

void SendMessage()
{
    int i;
    TsgcOpenAIClass_Message* oMessage = nullptr;
    TsgcOpenAIClass_Response_List_Messages* oMessages = nullptr;
    TsgcOpenAIClass_Run* oRun = nullptr;
    DoLog("[user]: " + memoMessage->Lines->Text);
    Screen->Cursor = crHourGlass; // Change cursor to hourglass
    try
    {
        oMessage = Assistant->CreateMessageText("thread_id", "Create a WebSocket Client that connects to eSeGeCe");
        if (oMessage != nullptr)
        {
            oRun = Assistant->CreateRunAndWait("thread_id");
            if (oRun != nullptr)
            {
                oMessages = Assistant->GetMessages("thread_id", oRun->Id);
                if (oMessages != nullptr && oMessages->Messages.Length > 0)
                {
                    memoMessage->Lines->Clear();
                    for (i = 0; i < oMessages->Messages.Length; i++)
                    {
                        DoLog("[assistant]: " + DoFormatResponse(oMessages->Messages[i].ContentText + "\r\n"));
                    }
                }
            }
        }
    }
    _finally
    {
        Screen->Cursor = crDefault; // Reset cursor to default
    }
}

```


TsgcAIOpenAIAssistant | Streaming

Em vez de esperar pela resposta completa do assistant, você pode fazer streaming da resposta utilizando Server-Sent Events. Basta passar o parâmetro **Stream = True** ao utilizar a função **CreateRun** e a resposta utilizará streaming.

Os eventos a seguir são utilizados para tratar as respostas de streaming:

- **OnStreamRun:** o evento é chamado quando há uma atualização no objeto run.
- **OnStreamMessage:** o evento é chamado quando há uma atualização no objeto de mensagem: criado, em progresso, concluído...
- **OnStreamMessageDelta:** Ocorre quando partes de uma Message estão sendo transmitidas via streaming.
- **OnStreamDone:** Ocorre quando um stream termina.
- **OnStreamError:** Ocorre quando ocorre um erro. Isso pode acontecer devido a um erro interno do servidor ou a um timeout.

Passo 1: Crie um Assistant

Um Assistant representa uma entidade que pode ser configurada para responder às mensagens de um usuário utilizando vários parâmetros, como modelo, instruções e ferramentas.

```
// Create a new assistant
TsgcAIOpenAIAssistant *oAssistant = new TsgcAIOpenAIAssistant(NULL);
// Set your API key
oAssistant->OpenAIOptions->ApiKey = txtAPIKey->Text;
// Assistant options
oAssistant->AssistantOptions->Name = "Math Tutor";
oAssistant->AssistantOptions->Instructions->Text = "You are a personal math tutor. Write and run code to answer math questions";
oAssistant->AssistantOptions->Model = "gpt-4o";
// Create the assistant
oAssistant->CreateAssistant();
```

Passo 2: Criar um Thread

Um Thread representa uma conversa entre um usuário e um ou vários Assistants. Você pode criar um Thread quando um usuário (ou sua aplicação de IA) inicia uma conversa com o seu Assistant.

```
oThread = oAssistant->CreateThread();
```

Etapa 3: Adicionar uma Mensagem à Thread e Executar utilizando Streaming

O conteúdo das mensagens que seus usuários ou aplicações criam é adicionado como objetos Message ao Thread. As mensagens podem conter tanto texto quanto arquivos. Não há limite para o número de Messages que você pode adicionar aos Threads — truncamos de forma inteligente qualquer contexto que não caiba na janela de contexto do modelo.

Uma vez que todas as Mensagens do usuário tenham sido adicionadas ao Thread, você pode executar (Run) o Thread com qualquer Assistant. Criar um Run usa o modelo e as tools associadas ao Assistant para gerar uma resposta. As respostas são enviadas usando um stream Server-Sent.

```

void SendMessage(TsgcAIOpenAIAssistant *oAssistant, TsgcAIClass_Thread *oThread, const UnicodeString aMessage)
{
    int i;
    TsgcOpenAIClass_Message *oMessage;
    TsgcOpenAIClass_Response_List_Messages *oMessages;
    TsgcOpenAIClass_Run *oRun;
    DoLog(L"[user]: " + aMessage);
    oMessage = oAssistant->CreateMessageText(oThread->Id, aMessage);
    if (oMessage != NULL)
    {
        oRun = oAssistant->CreateRun(oThread->Id, true);
    }
}

```

Passo 4: Tratar a Resposta

Use o evento OnStreamMessageDelta para ler a mensagem de stream enviada pelo servidor.

```

void __fastcall OnStreamMessageDelta(TObject *Sender, const TsgcOpenAIClass_MessageDelta &aMessageDelta, const St
{
    for (int i = aMessageDelta.Content.Low(); i <= aMessageDelta.Content.High(); i++)
    {
        String vType = aMessageDelta.Content[i]->_Type;
        if (vType == "text")
        {
            String vResponse = static_cast<TsgcOpenAIClass_MessageDeltaContent_Text*>(aMessageDelta.Content[i])->
        }
    }
}

```

TsgcAIOpenAIAssistant Function Calling

Semelhante à Chat Completions API, a Assistants API suporta chamada de função. A chamada de função permite que você descreva funções para a Assistants API e faça com que ela retorne de forma inteligente as funções que precisam ser chamadas junto com seus argumentos.

Neste exemplo, criaremos um assistente de clima e definiremos duas funções, **get_current_temperature** e **get_rain_probability**, como ferramentas que o Assistant pode chamar. Em nosso exemplo, que utiliza chamada de função em paralelo, perguntaremos ao Assistant como está o clima em San Francisco hoje e as chances de chuva. Também mostramos como exibir a resposta do Assistant com streaming.

Passo 1 Definir Funções

Ao criar seu assistant, você primeiro definirá as funções no parâmetro tools do assistant.

```
TsgcAIOpenAIAssistant *Assistant = new TsgcAIOpenAIAssistant(nullptr);
Assistant->OpenAIOptions->ApiKey = "sk-askdjfalskdjfl23kjkjasdefasdfj";
Assistant->AssistantOptions->Name = "Delphi Weather Bot";
Assistant->AssistantOptions->Instructions->Text = "You are a weather bot. Use the provided functions to answer qu";
Assistant->AssistantOptions->Model = "gpt-4o";
Assistant->AssistantOptions->Tools->Functions->Enabled = false;
Assistant->AssistantOptions->Tools->Functions->Functions->Text =
    "[{\"type\":\"function\",\"function\":{\"name\":\"get_current_temperature\",\"description\":\"Get the current
Assistant->AssistantOptions->Tools->FileSearch->Enabled = false;
Assistant->AssistantOptions->Tools->CodeInterpreter->Enabled = false;
```

Passo 2: Criar uma Thread e adicionar Mensagens

Crie um Thread quando um usuário inicia uma conversa e adicione Messages ao Thread conforme o usuário faz perguntas.

```
void SendMessage()
{
    int i;
    TsgcOpenAIClass_Message* oMessage = nullptr;
    TsgcOpenAIClass_Response_List_Messages* oMessages = nullptr;
    TsgcOpenAIClass_Run* oRun = nullptr;
    DoLog("[user]: " + memoMessage->Lines->Text);
    Screen->Cursor = crHourGlass; // Change cursor to hourglass
    try
    {
        oMessage = Assistant->CreateMessageText("thread_id", "What is the weather in San Francisco today and the
        if (oMessage != nullptr)
        {
            oRun = Assistant->CreateRunAndWait("thread_id");
            if (oRun != nullptr)
            {
                oMessages = Assistant->GetMessages("thread_id", oRun->Id);
                if (oMessages != nullptr && oMessages->Messages.Length > 0)
                {
                    memoMessage->Lines->Clear();
                    for (i = 0; i < oMessages->Messages.Length; i++)
                    {
                        DoLog("[assistant]: " + DoFormatResponse(oMessages->Messages[i].ContentText + "\r\n"));
                    }
                }
            }
        }
    }
    __finally
    {
        Screen->Cursor = crDefault; // Reset cursor to default
    }
}
```

Passo 3: Trate o Evento OnFunctionCall

Quando o componente detecta que um valor de parâmetro de função é necessário, o evento **OnFunctionCall** é chamado. Use o parâmetro **Request._Function** para conhecer os detalhes da requisição e use o **Response.Output** para enviar a resposta.

```
void __fastcall TFRMOpenAIAssistant::AssistantFunctionCall(TObject *Sender,
    const TsgcOpenAIClass_ToolCall &aRequest,
    const TsgcHTTPOpenAI_ToolCall_Response &aResponse)
{
    if (aRequest.Function.Name == "get_current_temperature")
        aResponse.Output = 30;
    else if (aRequest.Function.Name == "get_rain_probability")
        aResponse.Output = 10;
}
```

OpenAI Audio

Para utilizar as APIs da OpenAI com comandos de voz, os seguintes passos são necessários:

1. O áudio do microfone deve ser capturado, portanto é necessário um sistema speech-to-text para obter o texto que será enviado à OpenAI.
 1. A captura do áudio do microfone é feita usando o componente [TsgcAudioRecorderMCI](#).
 2. Uma vez capturado o áudio, ele é enviado à OpenAI Whisper API para converter o arquivo de áudio em texto.
2. Assim que obtemos o resultado de speech-to-text, enviamos o texto à OpenAI usando a API ChatCompletion.
3. A resposta da OpenAI deve então ser convertida em fala usando um dos seguintes componentes:
 1. [TsgcTextToSpeechSystem](#): (atualmente apenas para Windows) usa o text-to-speech do Windows do sistema operacional.
 2. [TsgcTextToSpeechGoogle](#): envia a resposta da OpenAI para os Servidores do Google Cloud e um arquivo mp3 é retornado, que é reproduzido pelo [TsgcAudioPlayerMCI](#).
 3. [TsgcTextToSpeechAmazon](#): envia a resposta da OpenAI para os Servidores Amazon AWS e um arquivo mp3 é retornado, que é reproduzido pelo [TsgcAudioPlayerMCI](#).

Componentes

Os seguintes componentes são usados para capturar áudio do microfone, reproduzir arquivos de áudio e converter texto em fala.

- [TsgcAudioRecorderMCI](#): (somente para windows) este componente permite que você acesse o microfone e converta a fala em um arquivo wave.
- [TsgcAudioPlayerMCI](#): (apenas para windows) este componente permite que você reproduza um arquivo mp3.
- [TsgcTextToSpeechSystem](#): (somente para windows) converte texto em fala sem a necessidade de usar um arquivo mp3 externo.
- [TsgcTextToSpeechGoogle](#): converte texto em fala utilizando qualquer uma das vozes do Google Cloud disponíveis.
- [TsgcTextToSpeechAmazon](#): converte texto em fala utilizando qualquer uma das vozes do Amazon AWS disponíveis.

TsgcAudioRecorderMCI

Este componente é usado para capturar o áudio do microfone e armazená-lo em um arquivo wave.

Introdução

Este componente é utilizado para capturar o áudio do microfone e armazená-lo em um arquivo wave. Atualmente, apenas o Windows é suportado.

Referência

- Propriedades — 3 propriedades disponíveis no componente.
- Métodos — 2 métodos públicos.
- Eventos — 0 eventos que você pode tratar.

TsgcAudioPlayerMCI

Este componente é utilizado para reproduzir os arquivos mp3 recebidos dos provedores de text-to-speech.

Introdução

Este componente é usado para reproduzir os arquivos mp3 recebidos dos provedores de text-to-speech. Atualmente apenas o Windows é suportado.

Referência

- Propriedades — 1 propriedade disponível no componente.
- Métodos — 1 método público.
- Eventos — 0 eventos que você pode tratar.

TsgcTextToSpeechSystem

Este é o Text-To-Speech padrão fornecido pelo Sistema Operacional; atualmente, apenas o Windows é suportado.

Introdução

Este é o Text-To-Speech padrão fornecido pelo Sistema Operacional; atualmente, apenas o Windows é suportado.

Referência

- Propriedades — 1 propriedade disponível no componente.
- Methods — 1 método público.
- Eventos — 0 eventos que você pode tratar.

TsgcTextToSpeechGoogle

O Text-To-Speech é uma API fornecida pelo Google Cloud que permite converter texto em arquivos mp3.

Introdução

O Text-To-Speech é uma API fornecida pelo Google Cloud que permite que você converta texto em arquivos mp3. Requer uma conta Google Cloud e a configuração da conta Text-To-Speech.

Uma vez configurada a Conta Text-To-Speech, um arquivo de configurações JSON deve ser baixado e definido na propriedade `GoogleOptions.Settings`.

Referência

- Propriedades — 3 propriedades disponíveis no componente.
- Métodos — 1 método público.
- Eventos — 0 eventos que você pode tratar.

TsgcTextToSpeechAmazon

Text-To-Speech é uma API fornecida pela Amazon AWS que permite que você converta texto em arquivos mp3.

Introdução

O Text-To-Speech é uma API fornecida pela Amazon AWS que permite converter texto em arquivos mp3. Requer uma conta Amazon AWS e a configuração da API Polly.

Referência

- Propriedades — 3 propriedades disponíveis no componente.
- Métodos — 1 método público.
- Eventos — 0 eventos que você pode tratar.

TsgcAIChat

TsgcAIChat é um único componente que funciona com qualquer provedor de IA.

Introdução

TsgcAIChat é um único componente que funciona com qualquer provedor de IA. Alterne os provedores alterando uma propriedade. O componente inclui gerenciamento integrado de histórico de conversas, facilitando a criação de aplicações de chat sem rastrear mensagens manualmente.

Provedores suportados

- OpenAI
- Anthropic
- Gemini
- DeepSeek
- Ollama
- Grok
- Mistral

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Métodos — 4 métodos públicos.
- Eventos — 3 eventos que você pode tratar.

TsgcAIOpenAIChatBot

O áudio do microfone deve ser capturado, portanto é necessário um sistema speech-to-text para obter o texto que será enviado para a OpenAI.

Introdução

Para construir um ChatBot com comandos de voz, são necessários os seguintes passos:

1. O áudio do microfone deve ser capturado, então um sistema de speech-to-text é necessário para obter o texto que será enviado à OpenAI.
 1. A captura do áudio do microfone é feita usando o componente [TsgcAudioRecorderMCI](#).
 2. Depois de capturar o áudio, ele é enviado à API OpenAI Whisper para converter o arquivo de áudio em texto.
2. Assim que obtemos o resultado de speech-to-text, enviamos o texto à OpenAI usando a API ChatCompletion.
3. A resposta da OpenAI deve então ser convertida em fala usando um dos seguintes componentes:
 1. [TsgcTextToSpeechSystem](#): (atualmente apenas para Windows) usa o text-to-speech do Windows do sistema operacional.
 2. [TsgcTextToSpeechGoogle](#): envia a resposta da OpenAI para os Servidores do Google Cloud e um arquivo mp3 é retornado, que é reproduzido pelo [TsgcAudioPlayerMCI](#).
 3. [TsgcTextToSpeechAmazon](#): envia a resposta da OpenAI para os Servidores Amazon AWS e um arquivo mp3 é retornado, que é reproduzido pelo [TsgcAudioPlayerMCI](#).

Referência

- Propriedades — 6 propriedades disponíveis no componente.
- Métodos — 4 métodos públicos.
- Events — 4 eventos que você pode tratar.

TsgcAIOpenAITranslator

O áudio do microfone deve ser capturado, portanto é necessário um sistema speech-to-text para obter o texto que será enviado para a OpenAI.

Introdução

Para construir um Tradutor com comandos de voz, os seguintes passos são necessários:

1. O áudio do microfone deve ser capturado, então um sistema de speech-to-text é necessário para obter o texto que será enviado à OpenAI.
 1. A captura do áudio do microfone é feita usando o componente [TsgcAudioRecorderMCI](#).
 2. Depois de capturar o áudio, ele é enviado à API OpenAI Whisper para converter o arquivo de áudio em texto.
2. Assim que obtemos o resultado de speech-to-text, enviamos o texto à OpenAI usando a API ChatCompletion.
3. A resposta da OpenAI deve então ser convertida em fala usando um dos seguintes componentes:
 1. [TsgcTextToSpeechSystem](#): (atualmente apenas para Windows) usa o text-to-speech do Windows do sistema operacional.
 2. [TsgcTextToSpeechGoogle](#): envia a resposta da OpenAI para os Servidores do Google Cloud e um arquivo mp3 é retornado, que é reproduzido pelo [TsgcAudioPlayerMCI](#).
 3. [TsgcTextToSpeechAmazon](#): envia a resposta da OpenAI para os Servidores Amazon AWS e um arquivo mp3 é retornado, que é reproduzido pelo [TsgcAudioPlayerMCI](#).

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Métodos — 2 métodos públicos.
- Eventos — 3 eventos que você pode tratar.

TsgcAIOpenAIEmbeddings

Embeddings são uma forma de representar palavras, frases ou até mesmo outros tipos de dados, como imagens ou áudio, em uma forma numérica.

Introdução

Embeddings são uma forma de representar palavras, frases ou até outros tipos de dados, como imagens ou áudio, em uma forma numérica. É como transformar palavras ou dados em números para que os computadores possam entendê-los e trabalhar com eles melhor.

Imagine que você tem um conjunto de palavras, como "cachorro", "gato" e "pássaro". Essas palavras têm significado, certo? Pois bem, embeddings atribuem a cada palavra um conjunto único de números (vetores) que capturam seu significado e suas relações com outras palavras.

Por exemplo, a palavra "dog" pode ser representada como [0.5, 0.2, -0.7], "cat" como [0.8, -0.3, 0.1] e "bird" como [0.3, 0.9, 0.4]. Os números nos vetores carregam informações sobre as características de cada palavra, como se estão relacionadas a animais ou quão semelhantes são entre si.

O incrível é que os embeddings podem ser aprendidos a partir de grandes quantidades de dados, então eles conseguem identificar semelhanças e diferenças entre palavras automaticamente. Essas representações numéricas ajudam os algoritmos de IA a entender a linguagem e a dar sentido a padrões complexos, o que é crucial em diversas aplicações como tradução de idiomas, análise de sentimentos e sistemas de recomendação. Elas também tornam mais fácil e rápido para os modelos de IA processar informações e fornecer resultados mais precisos!

Referência

- Propriedades — 4 propriedades disponíveis no componente.
- Métodos — 2 métodos públicos.
- Eventos — 5 eventos que você pode tratar.

TsgcAIDatabaseVectorFile

O componente armazena os vetores do banco de dados e os prompts em dois arquivos de texto.

Introdução

O componente armazena os vetores e prompts do banco de dados em dois arquivos de texto. Este componente deve ser usado apenas para fins de teste, não para produção, porque não é otimizado quando o número de vetores é alto.

Configuração

- **VectorFileOptions:**
 - **InputFilename:** o nome do arquivo utilizado para armazenar os dados de entrada.
 - **VectorFilename:** o nome do arquivo utilizado para armazenar os vetores.

Referência

- Propriedades — 2 propriedades disponíveis no componente.
- Métodos — 4 métodos públicos.
- Eventos — 0 eventos que você pode tratar.

TsgcAIDatabaseVectorPinecone

Componente *TsgcAIDatabaseVectorPinecone*.

Introdução

O componente é baseado no [cliente da Pinecone API](#) REST, que permite que você crie / atualize / exclua índices e vetores.

Configuração

- **PineconeOptions:**
 - **ApiKey:** configure aqui a API Key fornecida pelo pinecone, que pode ser obtida na sua conta pinecone.
 - **Environment:** por padrão é a conta gratuita "us-west4-gcp-free".
 - **LogOptions:** configure aqui se você quiser armazenar as requisições HTTP em um arquivo de texto.
- **PineconeIndexOptions:**
 - **IndexName:** o nome do índice utilizado para armazenar ou consultar os dados.
 - **ProjectId:** o id do projeto.

Referência

- Propriedades — 3 propriedades disponíveis no componente.
- Métodos — 4 métodos públicos.
- Eventos — 0 eventos que você pode tratar.

Embeddings | Criar vetores

Para utilizar os embeddings, primeiro devemos converter nossos dados em vetores.

Exemplo

Se você tiver um arquivo PDF, primeiro converta o arquivo PDF em texto e então utilize o método **CreateEmbeddingsFromFile** para obter os dados do vetor. Devido à limitação de tamanho dos Embeddings da OpenAI, se o arquivo for muito grande, os dados serão divididos automaticamente em partes, de modo que, a partir de um arquivo, você pode obter um ou múltiplos vetores.

Veja abaixo um exemplo de código.

```
void ConvertFileToVector()
{
    TOpenDialog* oDialog = new TOpenDialog(NULL);
    try
    {
        oDialog->Filter = "TXT Files|.txt";
        if (oDialog->Execute())
        {
            TsgcAIOpenAIEmbeddings* oEmbeddings = new TsgcAIOpenAIEmbeddings(NULL);
            try
            {
                TsgcAIDatabaseVectorFile* oFile = new TsgcAIDatabaseVectorFile(NULL);
                try
                {
                    oEmbeddings->Database = oFile;
                    oEmbeddings->OpenAIOptions->ApiKey = "<your api key>";
                    oEmbeddings->CreateEmbeddingsFromFile(oDialog->FileName);
                }
                __finally
                {
                    delete oFile;
                }
            }
            __finally
            {
                delete oEmbeddings;
            }
        }
    }
    __finally
    {
        delete oDialog;
    }
}
```

Embeddings | ChatBot

Uma vez convertidos todos os nossos dados em vetores, podemos começar a construir nosso próprio modelo. A ideia por trás disso é muito simples: toda vez que perguntamos ao bot, primeiro convertemos a pergunta em um vetor, depois pesquisamos em nosso banco de dados qual vetor é mais similar à pergunta e, por fim, usamos os dados mais similares e os adicionamos como contexto.

```
void AskToChatGPT(const std::string& aQuestion)
{
    TsgcAIOpenAIChatBot* oChatBot = new TsgcAIOpenAIChatBot(NULL);
    try
    {
        oChatBot->OpenAIOptions->ApiKey = "<your api key>";
        TsgcAIOpenAIEmbeddings* oEmbeddings = new TsgcAIOpenAIEmbeddings(NULL);
        try
        {
            oChatBot->Embeddings = oEmbeddings;
            TsgcAIDatabaseVectorFile* oFile = new TsgcAIDatabaseVectorFile(NULL);
            try
            {
                oEmbeddings->Database = oFile;
                std::string vContext = oChatBot->GetEmbedding(aQuestion);
                std::string message = "Answer the question based on the context below.\n\nContext:\n" +
                    vContext + "\nQuestion:" + aQuestion + "\nAnswer:";
                oChatBot->ChatAsUser(message.c_str());
            }
            __finally
            {
                delete oFile;
            }
        }
        __finally
        {
            delete oEmbeddings;
        }
    }
    __finally
    {
        delete oChatBot;
    }
}
```

Pinecone

[Pinecone.io](https://pinecone.io)

Pinecone é um banco de dados vetorial que permite enviar, consultar e excluir dados vetoriais de forma fácil e poderosa.

O Pinecone tem uma API pública que permite a terceiros integrar o Pinecone em suas próprias aplicações. O componente `TsgcHTTP_API_Pinecone` é um wrapper sobre a Pinecone API.

Configuração

Antes de começar, você deve se registrar no site da Pinecone e solicitar uma API key. Esta API key é utilizada para enviar as requisições da API e deve ser definida na propriedade **PineconeOptions.ApiKey** do componente `TsgcHTTP_API_Pinecone`.

Operações de Índice

Os seguintes métodos são suportados:

Method	Parâmetros	Descrição
IndexesList		Esta operação retorna uma lista dos seus índices Pinecone.
IndexCreate	TsgcHTTPIPineconeIndexCreate	Esta operação cria um índice Pinecone. Você pode utilizá-la para especificar a medida de similaridade, a dimensão dos vetores a serem armazenados no índice, o número de réplicas a utilizar e muito mais.
IndexDescribe	Nome do Índice	Obter a descrição de um índice.
IndexDelete	Nome do Índice	Esta operação exclui um índice existente.
IndexConfigure	Index Name, Replicas, PodType	Esta operação especifica o pod type e o número de réplicas de um índice.

Operações de coleção

Os seguintes métodos são suportados:

Method	Parâmetros	Descrição
CollectionsList		Esta operação retorna uma lista das suas coleções Pinecone.
CollectionCreate	Collection Name, Source	Esta operação cria uma collection do Pinecone.
CollectionDescribe	Collection Name	Obtém a descrição de uma coleção.

CollectionDelete	Collection Name	Esta operação exclui uma coleção existente.

Operações Vetoriais

Os seguintes métodos são suportados:

Method	Parâmetros	Descrição
VectorsDescribeIndexStats	Index Name, Project Id, Filter	A operação DescribeIndexStats retorna estatísticas sobre o conteúdo do índice, incluindo a contagem de vetores por namespace e o número de dimensões.
VectorsQuery	Index Name, Project Id, Params	A operação Query pesquisa um namespace, utilizando um vetor de consulta. Ela recupera os ids dos itens mais semelhantes em um namespace, juntamente com suas pontuações de similaridade.
VectorsDelete	Index Name, Project Id, Params	A operação Delete exclui vetores, por id, de um único namespace. Você pode excluir itens pelo id, de um único namespace.
VectorsFetch	Index Name, Project Id, Ids	A operação Fetch procura e retorna vetores, por ID, a partir de um único namespace. Os vetores retornados incluem os dados do vetor e/ou os metadados.
VectorsUpdate	Index Name, Project Id, Params	A operação Update atualiza um vetor em um namespace. Se um value for incluído, ele sobrescreverá o valor anterior. Se um set_metadata for incluído, os valores dos campos especificados nele serão adicionados ou sobrescreverão o valor anterior.
VectorsUpsert	Index Name, Project Id, Params	A operação Upsert grava vetores em um namespace. Se um novo valor for upsertado para um id de vetor existente, ele sobrescreverá o valor anterior.

Exemplo UPSERT

Encontre abaixo um exemplo de UPSERT de um único vetor com Id = "id1".

```
void UpsertPinecone(const String aIndexName, const String aProjectId, const std::vector<double> aVector)
{
    TsgcHTTP_API_Pinecone* oPinecone = new TsgcHTTP_API_Pinecone(NULL);
    try
    {
        oPinecone->PineconeOptions.API = "your-api-key";
        TsgcHTTTPineconeVectorUpserts* oParams = new TsgcHTTTPineconeVectorUpserts();
        try
        {
            TsgcArrayOfVectorUpsert oVectors;
            oVectors.push_back(new TsgcHTTTPineconeVectorUpsert());
            oVectors[0]->Id = "id1";
            oVectors[0]->Values = aVector;
            oParams->Vectors = oVectors;
            oPinecone->VectorsUpsert(aIndexName, aProjectId, oParams);
        }
        __finally
        {
            oParams->Free();
        }
    }
    __finally
}
```

```
{
    oPinecone->Free();
}
```

Exemplo de QUERY

Encontre abaixo um exemplo de QUERY de um único vetor.

```
void QueryPinecone(const string aIndexName, const string aProjectId, const std::vector<double>& aVector)
{
    TsgcHTTTPineconeVectorQuery* oParams = new TsgcHTTTPineconeVectorQuery();
    try
    {
        oParams->Vector = aVector;
        Pinecone.VectorsQuery(aIndexName, aProjectId, oParams);
    }
    finally
    {
        oParams->Free();
    }
}
```

Anthropic Claude

A Anthropic é uma empresa de segurança em IA que constrói sistemas de IA confiáveis, interpretáveis e direcionáveis. Sua família de modelos principal é a Claude, que se destaca em diálogo ponderado, criação de conteúdo, raciocínio complexo, codificação e muito mais.

A biblioteca `sgcWebSockets` fornece um componente Delphi **TsgcHTTP_API_Anthropic** para interagir com a API Anthropic Claude.

Anthropic API

A API Anthropic fornece acesso aos modelos Claude para construir aplicações com IA. A API suporta geração de texto, vision (compreensão de imagem), uso de ferramentas (chamada de função), extended thinking, processamento de documentos/PDF, prompt caching, citations, web search, streaming, contagem de tokens e message batches.

Features

- Messages
 - [Exemplos de Anthropic Messages](#)
- Vision
 - [Exemplos de Anthropic Vision](#)
- Tool Use
 - [Exemplos de Tool Use Anthropic](#)
- Models
 - [Exemplos de Anthropic Models](#)
- Batches
 - [Exemplos de Anthropic Batches](#)
- Extended Thinking
 - [Exemplos de Extended Thinking Anthropic](#)
- Documents
 - [Exemplos de Anthropic Documents](#)
- Prompt Caching
 - [Exemplos de Prompt Caching do Anthropic](#)
- Citations
 - [Exemplos de Anthropic Citations](#)
- Web Search
 - [Exemplos de Web Search do Anthropic](#)
- Structured Outputs
 - [Exemplos de Anthropic Structured Outputs](#)
- Files
 - [Exemplos de Anthropic Files API](#)
- MCP Connector
 - [Exemplos de Anthropic MCP Connector](#)

Configuração

A API da Anthropic usa chaves de API para autenticação. Visite sua página de [API Keys](#) no Console da Anthropic para recuperar a chave de API que você usará em suas requisições.

Lembre-se de que sua API key é um segredo! Não a compartilhe com outras pessoas nem a exponha em nenhum código do lado do cliente.

Esta **API Key** deve ser configurada na propriedade **AnthropicOptions.ApiKey** do componente. A propriedade **AnthropicOptions.AnthropicVersion** especifica a versão da API (padrão: 2023-06-01).

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "YOUR_API_KEY";
```

Properties

AnthropicOptions

- **ApiKey:** A API key para autenticação com a API da Anthropic.
- **AnthropicVersion:** A string de versão da API (padrão: 2023-06-01).
- **BetaHeaders:** Cabeçalhos opcionais de recursos beta para habilitar recursos de API pré-lançamento (por exemplo, files-api-2025-04-14, mcp-client-2025-11-20).
- **HttpOptions:** configurações de conexão HTTP.
 - **ReadTimeout:** Timeout em milissegundos para a leitura das respostas HTTP. O padrão é 0 (sem timeout).
- **LogOptions:** Configuração de logging.
 - **Enabled:** Quando True, requisições e respostas HTTP são registradas em um arquivo.
 - **FileName:** O caminho do arquivo onde a saída do log é escrita.
- **RetryOptions:** Configuração de retry automático para requisições de API que falharam.
 - **Enabled:** Quando True, as requisições que falharam são repetidas automaticamente.
 - **Retries:** Número máximo de tentativas de retry.
 - **Wait:** Tempo de espera em milissegundos entre as tentativas de retry.

Messages

Envie uma lista estruturada de mensagens de entrada com conteúdo de texto e/ou imagem, e o modelo gerará a próxima mensagem na conversa.

- **CreateMessage:** Cria uma mensagem com o modelo e os parâmetros especificados.
 - **Model:** O modelo a ser utilizado (por exemplo, claude-sonnet-4-20250514).
 - **Message:** O conteúdo da mensagem do usuário.
 - **MaxTokens:** O número máximo de tokens a serem gerados (obrigatório, padrão 4096).
- **CreateMessageWithSystem:** Cria uma mensagem com um system prompt.
 - **System:** System prompt que define o comportamento do assistant.
- **CreateMessageStream:** Cria uma mensagem com streaming (SSE) habilitado. Os eventos são entregues através do manipulador OnHTTPAPISSE.
- **CreateMessageJSON:** Cria uma mensagem que retorna JSON estruturado em conformidade com um JSON Schema fornecido.
 - **Schema:** Uma string de JSON Schema que define o formato de saída.
- **CreateMessageWithThinking:** Cria uma mensagem com extended thinking habilitado.
 - **BudgetTokens:** Orçamento máximo de tokens para o thinking (mínimo 1024).
- **CreateDocumentMessage:** Cria uma mensagem com um documento base64 (PDF ou texto).
 - **DocumentBase64:** Os dados do documento codificados em base64.
 - **MediaType:** O tipo MIME (por exemplo application/pdf).

Vision

O Claude pode entender imagens passadas como blocos de conteúdo codificados em base64 dentro das mensagens.

- **CreateVisionMessage:** Envia uma imagem com um prompt de texto.
 - **ImageBase64:** Os dados da imagem codificados em base64.
 - **MediaType:** O tipo MIME (image/jpeg, image/png, image/gif, image/webp).
 - **Prompt:** O prompt de texto que acompanha a imagem.

Uso de Ferramenta

O Claude pode utilizar ferramentas (function calling) para interagir com sistemas externos. Você define ferramentas com seus nomes, descrições e input schemas, e o Claude gerará blocos de conteúdo tool_use quando quiser chamar uma ferramenta.

Modelos

Lista e descreve os modelos Claude disponíveis.

- **GetModels:** Lista todos os modelos disponíveis.
- **GetModel:** Recupera informações sobre um modelo específico.
 - **ModelId:** O ID do modelo a ser recuperado.

Extended Thinking

O extended thinking permite que o Claude raciocine passo a passo antes de responder, melhorando a qualidade para tarefas complexas como matemática, programação e análise.

- **ThinkingType:** Defina como 'enabled' para ativar o pensamento, 'disabled' para desativá-lo.
- **ThinkingBudgetTokens:** Orçamento de tokens para thinking (mín. 1024, deve ser menor que MaxTokens).
- **CreateMessageWithAdaptiveThinking:** Cria uma mensagem com thinking adaptativo, deixando o Claude decidir automaticamente a profundidade do thinking.

Documents

O Claude pode processar documentos PDF e arquivos de texto enviados como blocos de conteúdo. Suporta os tipos de fonte base64, texto e URL.

- **CreateDocumentMessage:** Envia um documento com um prompt de texto.
 - **DocumentBase64:** Os dados do documento codificados em base64.
 - **MediaType:** O tipo MIME (application/pdf, text/plain).

Prompt Caching

Armazena em cache contexto utilizado com frequência (prompts de sistema, blocos de conteúdo, definições de ferramentas) entre chamadas de API para reduzir custos em até 90% nas leituras de cache.

- **SystemCacheControl:** Defina como True na requisição para fazer cache do system prompt.
- **CacheControl:** Defina como 'ephemeral' em blocos de conteúdo ou definições de tool.
- **BetaHeaders:** Cabeçalhos opcionais de recursos beta em AnthropicOptions.

Citações

Quando os documentos são enviados com citations habilitadas, o Claude inclui referências de origem em sua resposta. Os tipos de citation incluem char_location (texto), page_location (PDF), content_block_location (conteúdo personalizado) e web_search_result_location (busca na web).

Web Search

O Claude pode buscar na web por informações em tempo real usando a ferramenta integrada web_search. Outras ferramentas integradas incluem code_execution e computer use.

- **CreateMessageWithWebSearch:** Cria uma mensagem com a busca web habilitada.
 - **Model:** O modelo a ser usado.
 - **Message:** A consulta do usuário.

Token Counting

Conta o número de tokens em uma mensagem antes de enviá-la.

- **CountTokens:** Conta o número de tokens de entrada de uma mensagem.
 - **Model:** O modelo a ser usado para a contagem de tokens.
 - **Message:** O conteúdo da mensagem para o qual contar tokens.

Message Batches

A Message Batches API permite que você processe grandes volumes de mensagens de forma assíncrona.

- **ListBatches:** Lista todos os lotes de mensagens.
- **GetBatch:** Recupera um lote específico por ID.
- **CancelBatch:** Cancela um lote que ainda está em processamento.
- **GetBatchResults:** Recupera os resultados de um lote concluído.

Saídas Estruturadas

Força o Claude a retornar JSON válido em conformidade com um JSON Schema fornecido. Combine com o parâmetro Effort para controlar a qualidade da saída versus o custo.

- **CreateMessageJSON:** Cria uma mensagem com saída de esquema JSON.

- **Schema:** Uma string JSON Schema que define o formato de saída.
- **OutputFormatSchema:** (propriedade Request) JSON Schema para saída estruturada.
- **Effort:** (propriedade Request) Controla qualidade vs. custo: 'low', 'medium', 'high', 'max'.
- **Strict:** (propriedade Tool) Quando True, as entradas da ferramenta têm garantia de corresponder ao `input_schema`.

Files API

Faça upload, liste, recupere, baixe e exclua arquivos. Os arquivos enviados podem ser referenciados em mensagens utilizando blocos de conteúdo de documento com tipo de origem de arquivo. Requer o header beta `files-api-2025-04-14`.

- **UploadFile:** Faz upload de um arquivo local. Retorna os metadados do arquivo.
- **ListFiles:** Lista os arquivos enviados com paginação.
- **GetFile:** Recupera os metadados de um arquivo específico.
- **DownloadFile:** Baixa o conteúdo do arquivo.
- **DeleteFile:** Exclui permanentemente um arquivo.

Parâmetros da Requisição

Parâmetros de requisição adicionais disponíveis na Messages API.

- **ServiceTier:** Controla o uso do tier de prioridade: 'auto' (padrão) ou 'standard_only'.
- **InferenceGeo:** Controla a geografia de inferência: 'global' (padrão) ou 'us'.
- **Container:** ID de contêiner para reutilizar ambientes de execução de código entre requisições.
- **IsError:** (Propriedade de bloco de conteúdo) Defina True em blocos `tool_result` para indicar falha na execução da ferramenta.
- **CacheTTL:** Tempo de vida estendido do cache: '5m' (padrão) ou '1h'.
- **ThinkingType:** Defina como 'adaptive' para deixar o Claude decidir a profundidade do pensamento automaticamente.

MCP Connector

Conecte o Claude a servidores MCP (Model Context Protocol) externos para acessar ferramentas de terceiros. Requer o header beta `mcp-client-2025-11-20`.

- **CreateMessageWithMCP:** Cria uma mensagem com uma conexão a um servidor MCP.
 - **MCPServerUrl:** URL HTTPS do servidor MCP.
 - **MCPServerName:** Nome exclusivo para o servidor.
- **MCPServers:** (propriedade da Request) Array de definições de servidor MCP com `ServerType`, `Url`, `Name`, `AuthorizationToken`.
- **MCPServerName:** (propriedade Tool) Referencia um servidor MCP quando `ToolType` é 'mcp_toolset'.

TsgcHTTP_API_Anthropic

Cliente da REST API da Anthropic para Claude Messages, Files, Batches e contagem de tokens.

Introdução

Cliente da REST API da Anthropic para Claude Messages, Files, Batches e contagem de tokens.

Introdução

Cliente da REST API da Anthropic para Claude Messages, Files, Batches e contagem de tokens. Configure a propriedade **AnthropicOptions** com sua chave de API e configurações opcionais de HTTP/retentativa/log, depois chame os métodos tipados como `CreateMessage/CreateChatCompletion` com um objeto de requisição e receba uma classe de resposta tipada. Para mais informações sobre o provedor, consulte o guia [Anthropic](#).

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Métodos — 7 métodos públicos.
- Events — 2 eventos que você pode tratar.

Anthropic | Messages

Envia uma lista estruturada de mensagens de entrada com conteúdo de texto, e o modelo gerará a próxima mensagem na conversa.

Exemplo Simples

Envie uma mensagem Hello ao Claude.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->CreateMessage("claude-sonnet-4-20250514", "Hello!"));
```

Exemplo de system prompt

Envia uma mensagem com um system prompt para controlar o comportamento do Claude.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->CreateMessageWithSystem("claude-sonnet-4-20250514",
    "You are a helpful assistant that responds in Spanish.",
    "What is the capital of France?"));
```

Exemplo avançado

Use as classes tipadas de requisição/resposta para controle total sobre os parâmetros da mensagem.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";

TsgcAnthropicClass_Request_Messages *oRequest = new TsgcAnthropicClass_Request_Messages();
try {
    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!";
    oMessages = oRequest->Messages;
    SetLength(oMessages, 1);
    oMessages[0] = oMessage;
    oRequest->Messages = oMessages;

    TsgcAnthropicClass_Response_Messages *oResponse = Anthropic->CreateMessage(oRequest);
    try {
        if (oResponse->Content.Length > 0)
            ShowMessage(oResponse->Content[0]->Text);
    } __finally {
        delete oResponse;
    }
} __finally {
    delete oMessage;
    delete oRequest;
}
```

Exemplo de Streaming

Utilize Server-Sent Events (SSE) para transmitir a resposta em tempo real. Atribua o handler de evento OnHTTPAPISSE para receber os eventos de streaming.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
```

```
Anthropic->OnHTTPAPISSE = OnSSEEvent;  
Anthropic->_CreateMessageStream("claude-sonnet-4-20250514", "Tell me a story.");  
  
void __fastcall TForm1::OnSSEEvent(TObject *Sender, const String aEvent,  
    const String aData, 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

O Claude pode entender e analisar imagens. Você pode enviar imagens como dados codificados em base64 dentro de blocos de conteúdo.

Formatos de Imagem Suportados

- **image/jpeg** - imagens JPEG
- **image/png** - imagens PNG
- **image/gif** - imagens GIF
- **image/webp** - imagens WebP

Exemplo Simples

Envia uma imagem com um prompt pedindo ao Claude para descrevê-la.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";

// Load image and encode to base64
TFileStream *oStream = new TFileStream("photo.png", fmOpenRead);
try {
    TBytesStream *oBytes = new TBytesStream();
    try {
        oBytes->CopyFrom(oStream, 0);
        vBase64 = EncodeBase64(oBytes->Memory, oBytes->Size);
    } __finally {
        oBytes->Free();
    }
} __finally {
    oStream->Free();
}

ShowMessage(Anthropic->_CreateVisionMessage("claude-sonnet-4-20250514",
    "What is in this image?", vBase64, "image/png"));
```

Exemplo avançado

Use blocos de conteúdo para ter mais controle sobre a mensagem de imagem.

```
TsgcAnthropicClass_Request_Messages *oRequest = new TsgcAnthropicClass_Request_Messages(NULL);
try {
    oRequest->Model = "claude-sonnet-4-20250514";
    oRequest->MaxTokens = 4096;

    TsgcAnthropicClass_Request_Message *oMessage = new TsgcAnthropicClass_Request_Message(NULL);
    oMessage->Role = "user";

    // Image content block
    TsgcAnthropicClass_Request_Content_Block *oImageBlock = new TsgcAnthropicClass_Request_Content_Block(NULL);
    oImageBlock->ContentType = "image";
    oImageBlock->MediaType = "image/jpeg";
    oImageBlock->Data = vBase64;

    // Text content block
    TsgcAnthropicClass_Request_Content_Block *oTextBlock = new TsgcAnthropicClass_Request_Content_Block(NULL);
    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;

    TsgcAnthropicClass_Response_Messages *oResponse = Anthropic->CreateMessage(oRequest);
    try {
```

```
    if (Length(oResponse->Content) > 0)
        ShowMessage(oResponse->Content[0]->Text);
} __finally {
    oResponse->Free();
}
} __finally {
    oImageBlock->Free();
    oTextBlock->Free();
    oMessage->Free();
    oRequest->Free();
}
```



```
oTool->Free();  
oRequest->Free();  
}
```


Anthropic | Models

Lista e recupera informações sobre os modelos Claude disponíveis.

List Models

Lista todos os modelos Claude disponíveis.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_GetModels());
```

Get Model

Recupera informações sobre um modelo específico.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_GetModel("claude-sonnet-4-20250514"));
```

Resposta Tipada

Use a classe de resposta tipada para acesso estruturado aos dados do modelo.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";

TsgcAnthropicClass_Response_Models *oModels = Anthropic->GetModels();
try {
    for (int i = 0; i < oModels->Data.Length; i++)
        ShowMessage(oModels->Data[i]->Id + " - " + oModels->Data[i]->DisplayName);
} __finally {
    delete oModels;
}
```

Anthropic | Message Batches

A Message Batches API permite que você processe grandes volumes de mensagens de forma assíncrona. Isso é ideal para tarefas que não requerem respostas imediatas, como geração de conteúdo em massa, análise de dados ou fluxos de trabalho de processamento em lote.

List Batches

Lista todos os lotes de mensagens da sua organização.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_ListBatches());
```

Get Batch

Recupera informações sobre um lote específico.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_GetBatch("batch_id_here"));
```

Cancel Batch

Cancela um lote que ainda está em processamento.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_CancelBatch("batch_id_here"));
```

Get Batch Results

Recupera os resultados de um lote concluído.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_GetBatchResults("batch_id_here"));
```

Resposta Tipada

Utilize a classe de resposta tipada para acesso estruturado aos dados do batch.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";

TsgcAnthropicClass_Response_Batches *oBatches = Anthropic->ListBatches();
try {
    for (int i = 0; i < Length(oBatches->Batches); i++)
        ShowMessage(oBatches->Batches[i]->Id + " - " + oBatches->Batches[i]->ProcessingStatus);
} __finally {
    oBatches->Free();
}
```

Anthropic | Extended Thinking

O extended thinking dá ao Claude a capacidade de pensar em problemas complexos passo a passo antes de fornecer uma resposta. Quando habilitado, o Claude cria raciocínio interno (thinking blocks) que melhora a qualidade das respostas para matemática, programação, análise e outras tarefas complexas.

Exemplo Simples

Envia uma mensagem com extended thinking habilitado usando o método de conveniência. A temperatura é definida automaticamente como 1.0 (exigido pela API quando o thinking está habilitado).

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->CreateMessageWithThinking("claude-sonnet-4-20250514",
    "How many r's are in the word strawberry?", 10000));
```

Exemplo avançado

Use as classes tipadas de requisição/resposta para controle total. Defina ThinkingType como 'enabled' e ThinkingBudgetTokens com o orçamento de tokens desejado (mínimo 1024). A resposta conterá blocos de conteúdo thinking e text.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";

TsgcAnthropicClass_Request_Messages *oRequest = new TsgcAnthropicClass_Request_Messages();
try {
    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.";
    oMessages = oRequest->Messages;
    SetLength(oMessages, 1);
    oMessages[0] = oMessage;
    oRequest->Messages = oMessages;

    TsgcAnthropicClass_Response_Messages *oResponse = Anthropic->CreateMessage(oRequest);
    try {
        for (int i = 0; i < oResponse->Content.Length; i++) {
            if (oResponse->Content[i]->ContentType == "thinking")
                ShowMessage("Thinking: " + oResponse->Content[i]->Thinking);
            else if (oResponse->Content[i]->ContentType == "text")
                ShowMessage("Response: " + oResponse->Content[i]->Text);
        }
    } __finally {
        delete oResponse;
    }
} __finally {
    delete oMessage;
    delete oRequest;
}
```

Multi-turn com Thinking

Ao utilizar extended thinking em conversas com múltiplas interações, os blocos thinking e redacted_thinking de respostas anteriores devem ser repassados na conversa. Utilize o array ContentBlocks para incluir esses blocos.

```
// 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;
```

Properties

- **ThinkingType:** Defina como 'enabled' para ativar o thinking estendido, ou 'disabled' para desativá-lo.
- **ThinkingBudgetTokens:** O número máximo de tokens para thinking (mínimo 1024). Deve ser menor que MaxTokens.

Notas

- A temperatura deve ser 1.0 quando thinking está habilitado (o método de conveniência define isso automaticamente).
- TopK não pode ser utilizado quando o thinking está habilitado.
- BudgetTokens deve ser pelo menos 1024 e menor que MaxTokens.
- Os blocos de conteúdo da resposta podem incluir: tipos thinking, redacted_thinking e text.

Anthropic | Documents

O Claude pode processar documentos PDF e arquivos de texto enviados como blocos de conteúdo dentro das mensagens. Os documentos são enviados como dados codificados em base64 ou via URL, e o Claude pode analisar, resumir e responder perguntas sobre o conteúdo deles.

Exemplo Simples

Envia um documento PDF com uma pergunta usando o método de conveniência.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";

// Load PDF file and encode to base64
AnsiString vBase64 = sgcBase64Encode(LoadFileToBytes("document.pdf"));
ShowMessage(Anthropic->CreateDocumentMessage("claude-sonnet-4-20250514",
    "Summarize this document.", vBase64, "application/pdf"));
```

Exemplo Avançado com Citations

Use as classes tipadas para enviar um documento com citações habilitadas. Quando as citações estão habilitadas, a resposta do Claude incluirá referências a partes específicas do documento de origem.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";

TsgcAnthropicClass_Request_Messages *oRequest = new TsgcAnthropicClass_Request_Messages();
try {
    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";
    oBlocks = oMessage->ContentBlocks;
    SetLength(oBlocks, 2);
    oBlocks[0] = oDocBlock;
    oBlocks[1] = oTextBlock;
    oMessage->ContentBlocks = oBlocks;

    oMessages = oRequest->Messages;
    SetLength(oMessages, 1);
    oMessages[0] = oMessage;
    oRequest->Messages = oMessages;

    TsgcAnthropicClass_Response_Messages *oResponse = Anthropic->CreateMessage(oRequest);
    try {
        if (oResponse->Content.Length > 0)
            ShowMessage(oResponse->Content[0]->Text);
    } __finally {
        delete oResponse;
    }
} __finally {
    delete oRequest;
}
```

Properties

- **ContentType:** Defina como 'document' para blocos de conteúdo de documento.
- **SourceType:** O tipo de origem: 'base64' (padrão), 'text' ou 'url'.
- **MediaType:** O tipo MIME (por exemplo, application/pdf, text/plain).
- **Data:** Os dados do documento codificados em base64, ou a URL quando SourceType é 'url'.
- **Title:** Título opcional do documento para referência.
- **Context:** Contexto ou metadados opcionais sobre o documento.
- **CitationsEnabled:** Defina como True para habilitar citações de fonte na resposta.

Formatos Suportados

- **PDF:** application/pdf (máx 32 MB, máx 100 páginas por requisição)
- **Plain Text:** text/plain

Anthropic | Prompt Caching

O cache de prompts permite que você armazene em cache contexto usado com frequência entre chamadas de API, reduzindo custos em até 90% nas leituras de cache e melhorando a latência. O conteúdo em cache é marcado com um parâmetro `cache_control` e é reutilizado entre requisições dentro da janela TTL do cache.

System Prompt Caching

Faça cache de um system prompt longo para evitar reprocessá-lo a cada requisição. Defina `SystemCacheControl` como `True` para envolver automaticamente o system prompt com `cache_control`.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";

TsgcAnthropicClass_Request_Messages *oRequest = new TsgcAnthropicClass_Request_Messages(NULL);
try {
    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(NULL);
    oMessage->Role = "user";
    oMessage->Content = "Analyze this contract clause.";
    oMessages = oRequest->Messages;
    SetLength(oMessages, 1);
    oMessages[0] = oMessage;
    oRequest->Messages = oMessages;

    TsgcAnthropicClass_Response_Messages *oResponse = Anthropic->CreateMessage(oRequest);
    try {
        // Check cache usage in response
        ShowMessage("Cache created: " +
            IntToStr(oResponse->Usage->CacheCreationInputTokens));
        ShowMessage("Cache read: " +
            IntToStr(oResponse->Usage->CacheReadInputTokens));
        ShowMessage(oResponse->Content[0]->Text);
    } __finally {
        oResponse->Free();
    }
} __finally {
    oMessage->Free();
    oRequest->Free();
}
```

Content Block Caching

Faz cache de blocos de conteúdo específicos (por exemplo, documentos grandes ou contexto) definindo `Cache-Control` como `'ephemeral'` em blocos de conteúdo individuais.

```
TsgcAnthropicClass_Request_Content_Block *oBlock = new TsgcAnthropicClass_Request_Content_Block(NULL);
oBlock->ContentType = "text";
oBlock->Text = "(large reference text to cache)";
oBlock->CacheControl = "ephemeral"; // Mark for caching
```

Cache de Definição de Tool

Armazene em cache as definições de tool para evitar reprocessá-las a cada requisição. Isto é útil quando você tem muitas definições de tool que permanecem constantes entre as requisições.

```
TsgcAnthropicClass_Request_Tool *oTool = new TsgcAnthropicClass_Request_Tool(NULL);
oTool->Name = "search_database";
oTool->Description = "Search the database for records.";
oTool->InputSchema = "{\n\"type\":\n\"object\", \"properties\": {\n\"query\": {\n\"type\":\n\"string\"}\n}}";
oTool->CacheControl = "ephemeral"; // Cache this tool definition
```

Properties

- **SystemCacheControl:** Defina como True na requisição para fazer cache do prompt de system com cache_control efêmero.
- **CacheControl:** Defina como 'ephemeral' em blocos de conteúdo ou definições de ferramentas para marcá-los para cache.
- **CacheCreationInputTokens:** (Response) Número de tokens utilizados para criar entradas de cache.
- **CacheReadInputTokens:** (Resposta) Número de tokens lidos do cache (economia de custo).

Preços

- **Gravações em cache:** 1,25x o preço base do token de entrada (TTL de 5 minutos).
- **Cache reads:** 0,1x o preço base do token de entrada (90% de economia).
- **Cache TTL:** 5 minutos por padrão. Requisições subsequentes atualizam o TTL.

Anthropic | Citations

Quando os documentos são enviados com citações habilitadas, a resposta do Claude inclui referências de volta a partes específicas dos documentos de origem. Isso permite que você verifique afirmações e rastreie informações até sua origem.

Habilitando Citações

Defina `CitationsEnabled` como `True` nos blocos de conteúdo de documento. As citações devem estar habilitadas em TODOS ou em NENHUM dos documentos de uma requisição.

```
TsgcAnthropicClass_Request_Content_Block *oDocBlock = new TsgcAnthropicClass_Request_Content_Block(NULL);
oDocBlock->ContentType = "document";
oDocBlock->SourceType = "base64";
oDocBlock->MediaType = "application/pdf";
oDocBlock->Data = sgcBase64Encode(LoadFileToBytes("report.pdf"));
oDocBlock->CitationsEnabled = true; // Enable citations
```

Lendo Citations da Resposta

Os blocos de conteúdo de texto na resposta podem conter um array `Citations` com referências de origem.

```
TsgcAnthropicClass_Response_Messages *oResponse = Anthropic->CreateMessage(oRequest);
try {
    for (int i = 0; i < Length(oResponse->Content); i++) {
        if (oResponse->Content[i]->ContentType == "text") {
            ShowMessage(oResponse->Content[i]->Text);

            // Process citations
            for (int j = 0; j < Length(oResponse->Content[i]->Citations); j++) {
                TsgcAnthropicClass_Response_Citation *oCitation = oResponse->Content[i]->Citations[j];
                ShowMessage(Format(" Citation [%s]: \"%s\"",
                    ARRAYOFCONST((oCitation->CitationType, oCitation->CitedText))));

                if (oCitation->CitationType == "page_location")
                    ShowMessage(Format(" Pages %d-%d of \"%s\"",
                        ARRAYOFCONST((oCitation->StartPageNumber, oCitation->EndPageNumber,
                            oCitation->DocumentTitle))));
            }
        }
    }
} __finally {
    oResponse->Free();
}
```

Tipos de citação

- **char_location:** Referência em nível de caractere em documentos de texto simples. Campos: `DocumentIndex`, `DocumentTitle`, `StartCharIndex`, `EndCharIndex`.
- **page_location:** Referência no nível da página em documentos PDF. Campos: `DocumentIndex`, `DocumentTitle`, `StartPageNumber`, `EndPageNumber`.
- **content_block_location:** Referência em nível de bloco em documentos de conteúdo personalizado. Campos: `DocumentIndex`, `DocumentTitle`, `StartBlockIndex`, `EndBlockIndex`.
- **web_search_result_location:** Referência a resultados de busca na web. Campos: `Url`, `Title`, `EncryptedIndex`.

Anthropic | Web Search

A ferramenta Web Search permite que o Claude pesquise na web por informações em tempo real durante uma conversa. Esta é uma ferramenta integrada do lado do servidor que a Anthropic hospeda e executa automaticamente.

Exemplo Simples

Use o método de conveniência para criar uma mensagem com web search habilitado.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->CreateMessageWithWebSearch("claude-sonnet-4-20250514",
    "What are the latest news about Delphi programming?"));
```

Exemplo avançado

Use as classes tipadas para controle total sobre os parâmetros da tool de busca na web, como MaxUses.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";

TsgcAnthropicClass_Request_Messages *oRequest = new TsgcAnthropicClass_Request_Messages();
try {
    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
    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.";
    oMessages = oRequest->Messages;
    SetLength(oMessages, 1);
    oMessages[0] = oMessage;
    oRequest->Messages = oMessages;

    TsgcAnthropicClass_Response_Messages *oResponse = Anthropic->CreateMessage(oRequest);
    try {
        for (int i = 0; i < oResponse->Content.Length; i++) {
            if (oResponse->Content[i]->ContentType == "text")
                ShowMessage(oResponse->Content[i]->Text);
        }
    } finally {
        delete oResponse;
    }
} finally {
    delete oTool;
    delete oMessage;
    delete oRequest;
}
```

Tipos de Tool integrados

- **web_search_20250305:** Busca na web do lado do servidor. Propriedades: Name ('web_search'), MaxUses.
- **code_execution_20250825:** Execução de código em sandbox. Propriedades: Name ('code_execution'), MaxUses.
- **computer_20250124:** Uso de computador (mouse, teclado, capturas de tela). Propriedades: Name ('computer'), DisplayWidthPx, DisplayHeightPx.

Tipos de Conteúdo da Resposta

Quando ferramentas integradas são utilizadas, a resposta pode conter tipos de bloco de conteúdo adicionais:

- **server_tool_use:** Indica que o Claude invocou uma ferramenta integrada. Campos: Id, Name, Input.
- **web_search_tool_result:** Contém resultados de web search. Campos: ToolUseId, Data (conteúdo JSON bruto).

Anthropic | Structured Outputs

As Structured Outputs forçam o Claude a retornar respostas em conformidade com um esquema JSON. Isso garante uma saída JSON válida e parseável que corresponde à definição do seu esquema.

Exemplo Simples

Utilize o método de conveniência para criar uma mensagem com saída em JSON schema.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";

String vSchema = "{\n\"type\": \"object\", \"properties\": {\n\"name\": {\n\"type\": \"string\", \"\n\"age\": {\n\"type\": \"integer\"}}, \"required\": [\n\"name\", \"age\"], \"\n\"additionalProperties\": false}";

ShowMessage(Anthropic->CreateMessageJSON("claude-sonnet-4-20250514",
"Extract the name and age: John is 30 years old.", vSchema));
```

Exemplo avançado

Utilize as classes tipadas para combinar saída estruturada com o parâmetro effort.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";

TsgcAnthropicClass_Request_Messages *oRequest = new TsgcAnthropicClass_Request_Messages();
try {
    oRequest->Model = "claude-sonnet-4-20250514";
    oRequest->MaxTokens = 4096;

    // JSON schema for structured output
    oRequest->OutputFormatSchema =
        "{\n\"type\": \"object\", \"properties\": {\n\"sentiment\": {\n\"type\": \"string\", \"\n\"enum\": [\n\"positive\", \"negative\", \"neutral\"], \"confidence\": \"\n\"type\": \"number\"}}, \"required\": [\n\"sentiment\", \"confidence\"], \"\n\"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!";
    oMessages = oRequest->Messages;
    SetLength(oMessages, 1);
    oMessages[0] = oMessage;
    oRequest->Messages = oMessages;

    TsgcAnthropicClass_Response_Messages *oResponse = Anthropic->CreateMessage(oRequest);
    try {
        for (int i = 0; i < oResponse->Content.Length; i++) {
            if (oResponse->Content[i]->ContentType == "text")
                ShowMessage(oResponse->Content[i]->Text);
        }
    } finally {
        delete oResponse;
    }
} finally {
    delete oMessage;
    delete oRequest;
}
```

Strict Tool Use

Habilite o modo estrito nas definições de ferramentas para garantir que as entradas das ferramentas estejam exatamente em conformidade com o 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}";
```

Properties

- **OutputFormatSchema:** Uma string de JSON Schema. O texto de resposta do Claude será um JSON válido em conformidade com este schema.
- **Effort:** Controla a qualidade da saída versus o custo. Valores: 'low', 'medium', 'high' (padrão), 'max' (somente Opus 4.6).
- **Strict:** (em ferramentas) Quando True, as entradas da ferramenta têm garantia de conformidade com o input_schema.

Anthropic | Files API

A Files API permite que você faça upload, liste, recupere, baixe e exclua arquivos. Os arquivos enviados podem ser referenciados em mensagens utilizando blocos de conteúdo document com o tipo de fonte file.

Nota: A Files API requer o header beta files-api-2025-04-14. Defina isto na propriedade **AnthropicOptions.BetaHeaders**.

Upload a File

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
Anthropic->AnthropicOptions->BetaHeaders = "files-api-2025-04-14";

TsgcAnthropicClass_Response_File *oFile = Anthropic->UploadFile("C:\\documents\\report.pdf");
try {
    ShowMessage("File ID: " + oFile->Id);
} __finally {
    delete oFile;
}
```

List Files

```
TsgcAnthropicClass_Response_FileList *oList = Anthropic->ListFiles(50);
try {
    for (int i = 0; i < oList->Data.Length; i++)
        ShowMessage(oList->Data[i]->Id + " - " + oList->Data[i]->Filename +
            " (" + IntToStr(oList->Data[i]->SizeBytes) + " bytes)");
    ShowMessage("Has more: " + BoolToStr(oList->HasMore, true));
} __finally {
    delete oList;
}
```

Usar Arquivo em Mensagens

Referencie arquivos enviados (uploaded) utilizando o bloco de conteúdo document com SourceType definido como '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.";
```

Excluir um arquivo

```
TsgcAnthropicClass_Response_File *oDeleted = Anthropic->DeleteFile("file_abc123");
try {
    ShowMessage("Deleted: " + oDeleted->Id);
} __finally {
    delete oDeleted;
}
```

API Methods

- **UploadFile:** Faz o upload de um arquivo a partir de um caminho local. Retorna os metadados do arquivo (Id, Filename, MimeType, SizeBytes).
- **ListFiles:** Lista todos os arquivos enviados com suporte a paginação (Limit, AfterId, BeforeId).

- **GetFile:** Recupera metadados de um arquivo específico por ID.
- **DownloadFile:** Baixa o conteúdo de um arquivo (apenas arquivos criados pela ferramenta de execução de código podem ser baixados).
- **DeleteFile:** Exclui permanentemente um arquivo por ID.

Anthropic | MCP Connector

O conector MCP (Model Context Protocol) permite que o Claude acesse tools de servidores MCP externos. Isto habilita a integração com serviços de terceiros e provedores de tools personalizados.

Nota: O conector MCP requer o header beta `mcp-client-2025-11-20`. Defina-o na propriedade `AnthropicOptions.BetaHeaders`.

Exemplo Simples

Utilize o método de conveniência para criar uma mensagem com um servidor MCP.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
Anthropic->AnthropicOptions->BetaHeaders = "mcp-client-2025-11-20";

ShowMessage(Anthropic->CreateMessageWithMCP("claude-sonnet-4-20250514",
    "What tools are available?",
    "https://my-mcp-server.example.com/sse",
    "my-mcp-server"));
```

Exemplo avançado

Utilize as classes tipadas para controle total sobre a configuração do servidor MCP, incluindo a autenticação.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
Anthropic->AnthropicOptions->BetaHeaders = "mcp-client-2025-11-20";

TsgcAnthropicClass_Request_Messages *oRequest = new TsgcAnthropicClass_Request_Messages();
try {
    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
    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";
    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.";
    oMessages = oRequest->Messages;
    SetLength(oMessages, 1);
    oMessages[0] = oMessage;
    oRequest->Messages = oMessages;

    TsgcAnthropicClass_Response_Messages *oResponse = Anthropic->CreateMessage(oRequest);
    try {
        for (int i = 0; i < oResponse->Content.Length; i++) {
            if (oResponse->Content[i]->ContentType == "text")
                ShowMessage(oResponse->Content[i]->Text);
            else if (oResponse->Content[i]->ContentType == "mcp_tool_use")
                ShowMessage("MCP tool call: " + oResponse->Content[i]->Name +
                    " on " + oResponse->Content[i]->ServerName);
        }
    }
} finally {
```



```
        delete oResponse;
    }
} __finally {
    delete oServer;
    delete oTool;
    delete oMessage;
    delete oRequest;
}
```

Propriedades do MCP Server

- **ServerType:** Atualmente, apenas 'url' é suportado.
- **Url:** A URL HTTPS do servidor MCP.
- **Name:** Identificador único para este servidor. Deve corresponder ao MCPServerName no toolset.
- **AuthorizationToken:** Token Bearer OAuth opcional para servidores autenticados.

Tipos de Conteúdo da Resposta

- **mcp_tool_use:** O Claude invocou uma ferramenta MCP. Campos: Id, Name, ServerName, Input.
- **mcp_tool_result:** Resultado de uma execução de ferramenta MCP. Campos: ToolUseId, IsError, Data (conteúdo).

Gemini

O Google Gemini é uma família de modelos de IA multimodais desenvolvidos pelo Google DeepMind. Os modelos Gemini suportam geração de texto, visão, saídas estruturadas, embeddings e uso de ferramentas, oferecendo recursos poderosos para construir aplicações com IA.

A biblioteca `sgcWebSockets` fornece um componente Delphi **TsgcHTTP_API_Gemini** para interagir com a API do Gemini.

Gemini API

A API Gemini fornece acesso aos modelos Google Gemini para construir aplicações com IA. A API suporta geração de conteúdo, visão (compreensão de imagens), saídas JSON estruturadas, streaming, contagem de tokens, embeddings, uso de ferramentas (function calling) e listagem de modelos.

Features

- **Messages**
 - [Exemplos de Mensagens do Gemini](#)
- **Vision**
 - [Exemplos de Gemini Vision](#)
- **Models**
 - [Exemplos de Gemini Models](#)
- **Structured Outputs**
 - [Exemplos de Structured Outputs com Gemini](#)
- **Contagem de Tokens**
 - [Exemplos de Contagem de Tokens do Gemini](#)
- **Embeddings**
 - [Exemplos de Embeddings do Gemini](#)
- **Tool Use**
 - [Exemplos de Tool Use do Gemini](#)

Configuração

A API Gemini utiliza API keys para autenticação. Visite sua página [API Keys](#) no Google AI Studio para recuperar a API key que você utilizará em suas requisições.

Lembre-se de que sua API key é um segredo! Não a compartilhe com outras pessoas nem a exponha em nenhum código do lado do cliente.

Esta **API Key** deve ser configurada na propriedade **GeminiOptions.ApiKey** do componente.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "YOUR_API_KEY";
```

Messages

Envia conteúdo a um modelo Gemini e recebe respostas geradas. O modelo gera a próxima mensagem com base na entrada fornecida.

- **_CreateContent:** Cria conteúdo com o modelo e o prompt de usuário especificados.
 - **Model:** O modelo a usar (por exemplo, gemini-2.0-flash).
 - **Message:** O conteúdo da mensagem do usuário.
 - **MaxOutputTokens:** Número máximo de tokens a gerar.
- **_CreateContentWithSystem:** Cria conteúdo com uma instrução de sistema.
 - **Model:** O modelo a ser utilizado.
 - **System:** Instrução de sistema que define o comportamento do modelo.
 - **Message:** O conteúdo da mensagem do usuário.
 - **MaxOutputTokens:** Número máximo de tokens a gerar.
- **_CreateContentStream:** Cria conteúdo com streaming (SSE) habilitado. Os eventos são entregues através do manipulador de evento OnHTTPAPISSE.

Vision

Os modelos Gemini podem entender imagens passadas como conteúdo codificado em base64 junto com prompts de texto.

- **_CreateVisionContent:** Envia uma imagem com um prompt de texto.
 - **Model:** O modelo a ser utilizado.
 - **Prompt:** O prompt de texto que acompanha a imagem.
 - **Base64:** Os dados da imagem codificados em base64.
 - **MediaType:** O tipo MIME (image/jpeg, image/png, image/gif, image/webp).
 - **MaxOutputTokens:** Número máximo de tokens a serem gerados.

Saídas Estruturadas

Gera saída JSON estruturada a partir de um modelo Gemini fornecendo um JSON schema que define o formato de resposta esperado.

- **_CreateContentJSON:** Cria conteúdo com saída JSON estruturada.
 - **Model:** O modelo a utilizar.
 - **Message:** O conteúdo da mensagem do usuário.
 - **Schema:** Um JSON schema definindo a estrutura de saída esperada.
 - **MaxOutputTokens:** Número máximo de tokens a gerar.

Modelos

Lista e recupera detalhes sobre os modelos Gemini disponíveis.

- **_GetModels:** Lista todos os modelos disponíveis.
- **_GetModel:** Obtém os detalhes de um modelo específico.
 - **ModelId:** O identificador do modelo a ser recuperado.

Token Counting

Conta o número de tokens de uma mensagem antes de enviá-la a um modelo.

- **_CountTokens:** Conta tokens de uma mensagem.
 - **Model:** O modelo a utilizar para tokenização.
 - **Message:** O conteúdo de texto para o qual contar tokens.

Embeddings

Gera embeddings vetoriais para conteúdo de texto usando modelos Gemini.

- **_EmbedContent:** Gera embeddings para texto.
 - **Model:** O modelo a utilizar para a geração de embeddings.
 - **Text:** O conteúdo de texto para o qual gerar embeddings.

TsgcHTTP_API_Gemini

Cliente REST API do Google Gemini para geração de conteúdo, contagem de tokens e embeddings.

Introdução

Cliente REST API do Google Gemini para geração de conteúdo, contagem de tokens e embeddings.

Introdução

Cliente REST API do Google Gemini para geração de conteúdo, contagem de tokens e embeddings. Configure a propriedade **GeminiOptions** com sua API key e configurações opcionais de HTTP/retry/log e, em seguida, chame os métodos tipados como `CreateMessage/CreateChatCompletion` com um objeto de requisição e receba uma classe de resposta tipada.

Para mais informações sobre o provedor, consulte o guia [Gemini](#).

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Métodos — 3 métodos públicos.
- Eventos — 2 eventos que você pode tratar.

DeepSeek

A DeepSeek é uma empresa de IA chinesa focada em construir modelos de linguagem de código aberto poderosos. Seus modelos se destacam em codificação, raciocínio e tarefas de propósito geral, oferecendo desempenho robusto a preços competitivos.

A biblioteca `sgcWebSockets` fornece um componente Delphi **TsgcHTTP_API_DeepSeek** para interagir com a API DeepSeek.

DeepSeek API

A API do DeepSeek fornece acesso aos modelos DeepSeek para criar aplicações com IA. A API suporta geração de texto, visão (compreensão de imagens), streaming e listagem de modelos. A API segue um formato compatível com a OpenAI, tornando-a fácil de integrar.

Features

- **Mensagens**
 - [Exemplos de Mensagens DeepSeek](#)
- **Vision**
 - [Exemplos de Vision do DeepSeek](#)
- **Models**
 - [Exemplos de Modelos DeepSeek](#)

Configuração

A API do DeepSeek usa API keys para autenticação. Visite sua página de [API Keys](#) na Plataforma DeepSeek para recuperar a api key que você usará em suas requisições.

Lembre-se de que sua API key é um segredo! Não a compartilhe com outras pessoas nem a exponha em nenhum código do lado do cliente.

Esta **API Key** deve ser configurada na propriedade **DeepSeekOptions.ApiKey** do componente.

```
TsgcHTTP_API_DeepSeek *DeepSeek = new TsgcHTTP_API_DeepSeek(NULL);
DeepSeek->DeepSeekOptions->ApiKey = "YOUR_API_KEY";
```

Messages

Envia uma lista estruturada de mensagens de entrada com conteúdo de texto, e o modelo gerará a próxima mensagem na conversa.

- **_CreateMessage:** Cria uma mensagem com o modelo especificado e o prompt do usuário.
 - **Model:** O modelo a ser usado (por exemplo, deepseek-chat).
 - **Message:** O conteúdo da mensagem do usuário.
- **_CreateMessageWithSystem:** Cria uma mensagem com um system prompt.
 - **System:** System prompt que define o comportamento do assistente.
- **_CreateMessageStream:** Cria uma mensagem com streaming (SSE) habilitado. Os eventos são entregues por meio do handler de evento `OnHTTPAPISSE`.

Vision

O DeepSeek pode entender imagens passadas como conteúdo codificado em base64 dentro das mensagens.

- **_CreateVisionMessage:** Envia uma imagem com um prompt de texto.
 - **ImageBase64:** Os dados da imagem codificados em base64.
 - **MediaType:** O tipo MIME (image/jpeg, image/png, image/gif, image/webp).
 - **Prompt:** O prompt de texto para acompanhar a imagem.

Modelos

Lista os modelos DeepSeek disponíveis.

- **_GetModels:** Lista todos os modelos disponíveis.

TsgcHTTP_API_DeepSeek

Cliente REST API DeepSeek para Chat Completions.

Introdução

Cliente REST API DeepSeek para Chat Completions.

Introdução

Cliente REST API DeepSeek para Chat Completions.

Configure a propriedade **DeepSeekOptions** com sua chave de API e configurações opcionais de HTTP/retry/log, e então chame os métodos tipados, como `createMessage/createChatCompletion`, com um objeto de requisição e receba uma classe de resposta tipada.

Para mais informações sobre o provedor, consulte o guia [DeepSeek](#).

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Métodos — 1 métodos públicos.
- Eventos — 2 eventos que você pode tratar.

DeepSeek | Messages

Envia uma lista estruturada de mensagens de entrada com conteúdo de texto, e o modelo gerará a próxima mensagem na conversa.

Exemplo Simples

Envia uma mensagem Hello ao DeepSeek.

```
TsgcHTTP_API_DeepSeek *DeepSeek = new TsgcHTTP_API_DeepSeek(NULL);
DeepSeek->DeepSeekOptions->ApiKey = "API_KEY";
ShowMessage(DeepSeek->_CreateMessage("deepseek-chat", "Hello!"));
```

Exemplo de system prompt

Envia uma mensagem com um system prompt para controlar o comportamento do DeepSeek.

```
TsgcHTTP_API_DeepSeek *DeepSeek = new TsgcHTTP_API_DeepSeek(NULL);
DeepSeek->DeepSeekOptions->ApiKey = "API_KEY";
ShowMessage(DeepSeek->_CreateMessageWithSystem("deepseek-chat",
    "You are a helpful assistant that responds in Spanish.",
    "What is the capital of France?"));
```

Exemplo de Streaming

Utilize Server-Sent Events (SSE) para transmitir a resposta em tempo real. Atribua o handler de evento OnHTTPAPISSE para receber os eventos de streaming.

```
TsgcHTTP_API_DeepSeek *DeepSeek = new TsgcHTTP_API_DeepSeek(NULL);
DeepSeek->DeepSeekOptions->ApiKey = "API_KEY";
DeepSeek->OnHTTPAPISSE = OnSSEEvent;
DeepSeek->_CreateMessageStream("deepseek-chat", "Tell me a story.");

void __fastcall TForm1::OnSSEEvent(TObject *Sender, const String aEvent,
    const String aData, bool &Cancel)
{
    // aEvent contains the event type
    // aData contains the JSON data for this event
    Memo1->Lines->Add(aData);
}
```


DeepSeek | Vision

O DeepSeek consegue entender e analisar imagens. Você pode enviar imagens como dados codificados em base64 dentro de blocos de conteúdo.

Vision Example

Envia uma imagem com um prompt solicitando que o DeepSeek a descreva.

```
TsgcHTTP_API_DeepSeek *DeepSeek = new TsgcHTTP_API_DeepSeek(NULL);
DeepSeek->DeepSeekOptions->ApiKey = "API_KEY";

// Load image and encode to base64
TFileStream *oStream = new TFileStream("photo.jpg", fmOpenRead);
try {
    TBytesStream *oBytes = new TBytesStream();
    try {
        oBytes->CopyFrom(oStream, 0);
        vImageBase64 = EncodeBase64(oBytes->Memory, oBytes->Size);
    } __finally {
        oBytes->Free();
    }
} __finally {
    oStream->Free();
}

ShowMessage(DeepSeek->_CreateVisionMessage("deepseek-chat",
    "Describe this image", vImageBase64, "image/jpeg"));
```

DeepSeek | Modelos

Lista os modelos DeepSeek disponíveis.

List Models

Lista todos os modelos DeepSeek disponíveis.

```
TsgcHTTP_API_DeepSeek *DeepSeek = new TsgcHTTP_API_DeepSeek(NULL);  
DeepSeek->DeepSeekOptions->ApiKey = "API_KEY";  
ShowMessage(DeepSeek->_GetModels());
```

Gemini

O Google Gemini é uma família de modelos de IA multimodais desenvolvidos pelo Google DeepMind. Os modelos Gemini suportam geração de texto, visão, saídas estruturadas, embeddings e uso de ferramentas, oferecendo recursos poderosos para construir aplicações com IA.

A biblioteca `sgcWebSockets` fornece um componente Delphi **TsgcHTTP_API_Gemini** para interagir com a API do Gemini.

Gemini API

A API Gemini fornece acesso aos modelos Google Gemini para construir aplicações com IA. A API suporta geração de conteúdo, visão (compreensão de imagens), saídas JSON estruturadas, streaming, contagem de tokens, embeddings, uso de ferramentas (function calling) e listagem de modelos.

Features

- **Messages**
 - [Exemplos de Mensagens do Gemini](#)
- **Vision**
 - [Exemplos de Gemini Vision](#)
- **Models**
 - [Exemplos de Gemini Models](#)
- **Structured Outputs**
 - [Exemplos de Structured Outputs com Gemini](#)
- **Contagem de Tokens**
 - [Exemplos de Contagem de Tokens do Gemini](#)
- **Embeddings**
 - [Exemplos de Embeddings do Gemini](#)
- **Tool Use**
 - [Exemplos de Tool Use do Gemini](#)

Configuração

A API Gemini utiliza API keys para autenticação. Visite sua página [API Keys](#) no Google AI Studio para recuperar a API key que você utilizará em suas requisições.

Lembre-se de que sua API key é um segredo! Não a compartilhe com outras pessoas nem a exponha em nenhum código do lado do cliente.

Esta **API Key** deve ser configurada na propriedade **GeminiOptions.ApiKey** do componente.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "YOUR_API_KEY";
```

Messages

Envia conteúdo a um modelo Gemini e recebe respostas geradas. O modelo gera a próxima mensagem com base na entrada fornecida.

- **_CreateContent**: Cria conteúdo com o modelo e o prompt de usuário especificados.
 - **Model**: O modelo a usar (por exemplo, gemini-2.0-flash).
 - **Message**: O conteúdo da mensagem do usuário.
 - **MaxOutputTokens**: Número máximo de tokens a gerar.
- **_CreateContentWithSystem**: Cria conteúdo com uma instrução de sistema.
 - **Model**: O modelo a ser utilizado.
 - **System**: Instrução de sistema que define o comportamento do modelo.
 - **Message**: O conteúdo da mensagem do usuário.
 - **MaxOutputTokens**: Número máximo de tokens a gerar.
- **_CreateContentStream**: Cria conteúdo com streaming (SSE) habilitado. Os eventos são entregues através do manipulador de evento OnHTTPAPISSE.

Vision

Os modelos Gemini podem entender imagens passadas como conteúdo codificado em base64 junto com prompts de texto.

- **_CreateVisionContent:** Envia uma imagem com um prompt de texto.
 - **Model:** O modelo a ser utilizado.
 - **Prompt:** O prompt de texto que acompanha a imagem.
 - **Base64:** Os dados da imagem codificados em base64.
 - **MediaType:** O tipo MIME (image/jpeg, image/png, image/gif, image/webp).
 - **MaxOutputTokens:** Número máximo de tokens a serem gerados.

Saídas Estruturadas

Gera saída JSON estruturada a partir de um modelo Gemini fornecendo um JSON schema que define o formato de resposta esperado.

- **_CreateContentJSON:** Cria conteúdo com saída JSON estruturada.
 - **Model:** O modelo a utilizar.
 - **Message:** O conteúdo da mensagem do usuário.
 - **Schema:** Um JSON schema definindo a estrutura de saída esperada.
 - **MaxOutputTokens:** Número máximo de tokens a gerar.

Modelos

Lista e recupera detalhes sobre os modelos Gemini disponíveis.

- **_GetModels:** Lista todos os modelos disponíveis.
- **_GetModel:** Obtém os detalhes de um modelo específico.
 - **ModelId:** O identificador do modelo a ser recuperado.

Token Counting

Conta o número de tokens de uma mensagem antes de enviá-la a um modelo.

- **_CountTokens:** Conta tokens de uma mensagem.
 - **Model:** O modelo a utilizar para tokenização.
 - **Message:** O conteúdo de texto para o qual contar tokens.

Embeddings

Gera embeddings vetoriais para conteúdo de texto usando modelos Gemini.

- **_EmbedContent:** Gera embeddings para texto.
 - **Model:** O modelo a utilizar para a geração de embeddings.
 - **Text:** O conteúdo de texto para o qual gerar embeddings.

Gemini | Mensagens

Envia um prompt a um modelo Gemini e recebe conteúdo gerado. A API Gemini usa o endpoint generateContent para processar entrada de texto e retornar respostas do modelo.

Exemplo Simples

Envia uma mensagem Hello ao Gemini.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";
ShowMessage(Gemini->_CreateContent("gemini-2.0-flash", "Hello!"));
```

System Instruction Example

Envia uma mensagem com uma instrução de sistema para controlar o comportamento do Gemini.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";
ShowMessage(Gemini->_CreateContentWithSystem("gemini-2.0-flash",
    "You are a helpful assistant that responds in Spanish.",
    "What is the capital of France?"));
```

Exemplo de Streaming

Utilize Server-Sent Events (SSE) para transmitir a resposta em tempo real. Atribua o handler de evento OnHTTPAPISSE para receber os eventos de streaming.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";
Gemini->OnHTTPAPISSE = OnSSEEvent;
Gemini->_CreateContentStream("gemini-2.0-flash", "Tell me a story.");

void __fastcall TForm1::OnSSEEvent(TObject *Sender, const String aEvent,
    const String aData, bool &Cancel)
{
    Memo1->Lines->Add(aData);
}
```

Exemplo avançado

Utilize as classes tipadas de request/response para controle total sobre os parâmetros de geração de conteúdo.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";

TsgcGeminiClass_Request_GenerateContent *oRequest = new TsgcGeminiClass_Request_GenerateContent();
try {
    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;

    TsgcGeminiClass_Response_GenerateContent *oResponse = Gemini->CreateContent(oRequest);
    try {
```

```
    if (Length(oResponse->Candidates) > 0)
        if (Length(oResponse->Candidates[0]->Parts) > 0)
            ShowMessage(oResponse->Candidates[0]->Parts[0]->Text);
} __finally {
    delete oResponse;
}
} __finally {
    delete oParts[0];
    delete oContents[0];
    delete oRequest;
}
```

Gemini | Vision

O Gemini pode entender e analisar imagens. Você pode enviar imagens como dados codificados em base64 dentro de partes de conteúdo.

Vision Example

Envia uma imagem codificada em base64 ao Gemini para análise.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";

// Load image and encode to base64
TFileStream *oStream = new TFileStream("photo.jpg", fmOpenRead);
try {
    TBytesStream *oBytes = new TBytesStream();
    try {
        oBytes->CopyFrom(oStream, 0);
        vImageBase64 = EncodeBase64(oBytes->Memory, oBytes->Size);
    } __finally {
        oBytes->Free();
    }
} __finally {
    oStream->Free();
}

ShowMessage(Gemini->_CreateVisionContent("gemini-2.0-flash",
    "Describe this image", vImageBase64, "image/jpeg"));
```

Gemini | Modelos

Lista e recupera informações sobre os modelos Gemini disponíveis.

List Models

Lista todos os modelos disponíveis.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";
ShowMessage(Gemini->_GetModels());
```

Get Model

Obtém detalhes de um modelo específico.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";
ShowMessage(Gemini->_GetModel("gemini-2.0-flash"));
```

Exemplo avançado

Use as classes de resposta tipadas para controle total sobre os dados do modelo.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";

TsgcGeminiClass_Response_Models *oModels = Gemini->GetModels();
try {
    for (int i = 0; i < oModels->Models.Length; i++)
        ShowMessage(oModels->Models[i]->DisplayName + " - " + oModels->Models[i]->Description);
} __finally {
    delete oModels;
}
```


Gemini | Structured Outputs

O Gemini pode retornar respostas em formato JSON estruturado, em conformidade com um schema que você define. Isso garante uma saída JSON válida e parseável que corresponde à definição do seu schema.

Exemplo Simples

Utilize o método de conveniência para criar conteúdo com saída em JSON schema.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";

String vSchema = "{\"type\":\"object\",\"properties\":{\"name\":{\"type\":\"string\"}, \"age\":{\"type\":\"integer\"}}, \"required\": [\"name\", \"age\"]}";

ShowMessage(Gemini->CreateContentJSON("gemini-2.0-flash",
    "Extract the name and age: John is 30 years old.", vSchema));
```

Exemplo avançado

Utilize as classes tipadas para configurar a saída estruturada com ResponseMimeType e ResponseSchema.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";

TsgcGeminiClass_Request_GenerateContent *oRequest = new TsgcGeminiClass_Request_GenerateContent();
try {
    oRequest->Model = "gemini-2.0-flash";
    oRequest->MaxOutputTokens = 4096;
    oRequest->ResponseMimeType = "application/json";
    oRequest->ResponseSchema =
        "{\"type\":\"object\",\"properties\":{\"sentiment\":{\"type\":\"string\"}, \"confidence\":{\"type\":\"number\"}}, \"required\": [\"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;

    TsgcGeminiClass_Response_GenerateContent *oResponse = Gemini->CreateContent(oRequest);
    try {
        if (Length(oResponse->Candidates) > 0)
            if (Length(oResponse->Candidates[0]->Parts) > 0)
                ShowMessage(oResponse->Candidates[0]->Parts[0]->Text);
    } __finally {
        delete oResponse;
    }
} __finally {
    delete oParts[0];
    delete oContents[0];
    delete oRequest;
}
```

Properties

- **ResponseMimeType:** Defina como 'application/json' para habilitar a saída JSON estruturada.
- **ResponseSchema:** Uma string JSON Schema definindo a estrutura de saída esperada.

Gemini | Token Counting

Conta o número de tokens em uma mensagem antes de enviá-la à API. Isto ajuda a estimar custos e garantir que as mensagens caibam dentro dos limites do modelo.

Exemplo Simples

Conta os tokens de uma mensagem.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";
ShowMessage(Gemini->_CountTokens("gemini-2.0-flash", "Hello, how are you?"));
```

Exemplo avançado

Utilize as classes tipadas de request/response para controle total sobre os parâmetros de contagem de tokens.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";

TsgcGeminiClass_Request_CountTokens *oRequest = new TsgcGeminiClass_Request_CountTokens();
try {
    oRequest->Model = "gemini-2.0-flash";
    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 = "Explain quantum computing in simple terms.";
    oContents[0]->Parts = oParts;
    oRequest->Contents = oContents;

    TsgcGeminiClass_Response_CountTokens *oResponse = Gemini->CountTokens(oRequest);
    try {
        ShowMessage("Total tokens: " + IntToStr(oResponse->TotalTokens));
    } __finally {
        delete oResponse;
    }
} __finally {
    delete oParts[0];
    delete oContents[0];
    delete oRequest;
}
```

Gemini | Embeddings

Gera embeddings vetoriais para conteúdo de texto. Os embeddings capturam o significado semântico do texto e podem ser usados para busca por similaridade, clustering e outras tarefas de NLP.

Exemplo Simples

Gera embeddings para texto.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";
ShowMessage(Gemini->EmbedContent("text-embedding-004", "Hello world"));
```

Exemplo avançado

Use as classes de resposta tipadas para controle total sobre os dados de embedding.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";

TsgcGeminiClass_Response_Embedding *oEmbedding = Gemini->EmbedContent("text-embedding-004", "Hello world");
try {
    ShowMessage("Dimensions: " + IntToStr(oEmbedding->Values.Length));
    for (int i = 0; i < oEmbedding->Values.Length; i++)
        ShowMessage(FloatToStr(oEmbedding->Values[i]));
} __finally {
    delete oEmbedding;
}
```

Gemini | Uso de Ferramentas

O Gemini suporta chamada de função (uso de ferramentas), permitindo que você defina funções que o modelo pode invocar durante uma conversa. Quando o Gemini decide chamar uma função, ele retorna uma parte function-Call em sua resposta. Você então executa a função e envia o resultado de volta.

Fluxo de Function Calling

1. Defina declarações de função com nome, descrição e parâmetros (JSON Schema).
2. Envia uma requisição de geração de conteúdo com declarações de função.
3. O Gemini responde com uma parte **functionCall** contendo o nome da função e os argumentos.
4. Executa a função com os argumentos fornecidos.
5. Envie uma nova requisição com uma parte **functionResponse** contendo o resultado.
6. O Gemini responde com a resposta final.

Exemplo

Define uma função de clima e trata a chamada de função.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";

// Step 1: Create request with function declarations
TsgcGeminiClass_Request_GenerateContent *oRequest = new TsgcGeminiClass_Request_GenerateContent();
try {
    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 =
        "{\n\"type\": \"object\", \"properties\": {\n\"location\": {\n\"type\": \"string\", \"\n\"description\": \"The city and state\"}}, \"required\": [\n\"location\"]}";
    oRequest->FunctionDeclarations = oFunctions;

    // Step 2: Send request
    TsgcGeminiClass_Response_GenerateContent *oResponse = Gemini->CreateContent(oRequest);
    try {
        // Step 3: Check for function call
        if (Length(oResponse->Candidates) > 0) {
            if (Length(oResponse->Candidates[0]->Parts) > 0) {
                if (oResponse->Candidates[0]->Parts[0]->FunctionCallName != "") {
                    vFunctionName = oResponse->Candidates[0]->Parts[0]->FunctionCallName;
                    vFunctionArgs = oResponse->Candidates[0]->Parts[0]->FunctionCallArgs;
                    // Step 4: Execute function and get result
                    vResult = "72 degrees and sunny";
                }
            }
        }
    } finally {
        delete oResponse;
    }
} finally {
    delete oParts[0];
    delete oContents[0];
    delete oFunctions[0];
    delete oRequest;
}
```

Properties

- **FunctionDeclarations:** Array de TsgcGeminiClass_Request_FunctionDeclaration definindo as funções disponíveis.
 - **Name:** O nome da função.
 - **Description:** Uma descrição do que a função faz.
 - **Parameters:** String JSON Schema definindo os parâmetros da função.
- **ToolChoice:** Controla o comportamento de function calling. Defina o modo (por exemplo, 'AUTO', 'ANY', 'NONE').

Ollama

O Ollama é uma ferramenta de código aberto para executar modelos de linguagem grandes localmente. Ele suporta uma ampla variedade de modelos, incluindo Llama, Mistral, Gemma, Phi e muitos outros, permitindo inferência de IA local sem exigir acesso a uma API na nuvem.

A biblioteca `sgcWebSockets` fornece um componente Delphi **TsgcHTTP_API_Ollama** para interagir com a Ollama API.

Ollama API

A API Ollama fornece acesso a modelos rodando localmente para aplicações com IA. A API suporta geração de texto, streaming, gerenciamento de modelos e embeddings. Nenhuma chave de API é necessária por padrão, já que os modelos rodam localmente. O host e a porta são configuráveis.

Features

- **Messages**
 - [Exemplos de Ollama Messages](#)
- **Models**
 - [Exemplos de Ollama Models](#)
- **Embeddings**
 - [Exemplos de Embeddings do Ollama](#)

Configuração

O Ollama roda localmente e por padrão escuta em **http://localhost:11434**. Configure o host na propriedade **OllamaOptions.Host**. Uma api key é opcional e só é necessária se você tiver configurado autenticação na sua instância Ollama.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
```

Messages

Envia uma lista estruturada de mensagens de entrada com conteúdo de texto, e o modelo gerará a próxima mensagem na conversa.

- **_CreateMessage**: Cria uma mensagem com o modelo e o prompt de usuário especificados.
 - **Model**: O modelo a ser utilizado (por exemplo, llama3).
 - **Message**: O conteúdo da mensagem do usuário.
- **_CreateMessageWithSystem**: Cria uma mensagem com um system prompt.
 - **System**: System prompt que define o comportamento do assistente.
- **_CreateMessageStream**: Cria uma mensagem com streaming (SSE) habilitado. Os eventos são entregues por meio do handler de evento `OnHTTPAPISSE`.

Modelos

Gerencia e consulta os modelos disponíveis localmente.

- **_GetModels**: Lista todos os modelos disponíveis.
- **_GetTags**: Lista todas as tags de modelo disponíveis localmente.
- **_ShowModel**: Recupera informações detalhadas sobre um modelo específico.
 - **Model**: O nome do modelo a consultar.
- **_PullModel**: Baixa um modelo da biblioteca de modelos do Ollama.
 - **Model**: O nome do modelo a baixar.
- **_DeleteModel**: Exclui um modelo disponível localmente.
 - **Model**: O nome do modelo a excluir.

Embeddings

Gera embeddings vetoriais a partir de entrada de texto utilizando modelos executados localmente.

- **_CreateEmbeddings:** Gera embeddings para o texto fornecido.
 - **Model:** O modelo a utilizar para os embeddings (por exemplo llama3).
 - **Input:** O texto para o qual gerar embeddings.

TsgcHTTP_API_Ollama

Cliente Ollama REST API para Chat Completions e Embeddings locais.

Introdução

Cliente Ollama REST API para Chat Completions e Embeddings locais.

Introdução

Cliente Ollama REST API para Chat Completions e Embeddings locais.

Configure a propriedade **OllamaOptions** com sua chave de API e configurações opcionais de HTTP/retry/log, depois chame os métodos tipados como `CreateMessage/CreateChatCompletion` com um objeto de requisição e receba uma classe de resposta tipada.

Para informações sobre o provedor, consulte o guia [Ollama](#).

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Métodos — 2 métodos públicos.
- Eventos — 2 eventos que você pode tratar.

Ollama | Messages

Envia uma lista estruturada de mensagens de entrada com conteúdo de texto, e o modelo gerará a próxima mensagem na conversa.

Exemplo Simples

Envia uma mensagem Hello a um modelo Ollama local.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
ShowMessage(Ollama->_CreateMessage("llama3", "Hello!"));
```

Exemplo de system prompt

Envia uma mensagem com um system prompt para controlar o comportamento do modelo.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
ShowMessage(Ollama->_CreateMessageWithSystem("llama3",
    "You are a helpful assistant that responds in Spanish.",
    "What is the capital of France?"));
```

Exemplo de Streaming

Utilize Server-Sent Events (SSE) para transmitir a resposta em tempo real. Atribua o handler de evento OnHTTPAPISSE para receber os eventos de streaming.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
Ollama->OnHTTPAPISSE = OnSSEEvent;
Ollama->_CreateMessageStream("llama3", "Tell me a story.");

void __fastcall TForm1::OnSSEEvent(TObject *Sender, const String aEvent,
    const String aData, bool &Cancel)
{
    // aEvent contains the event type
    // aData contains the JSON data for this event
    Memo1->Lines->Add(aData);
}
```

Ollama | Modelos

Gerencie e consulte os modelos Ollama disponíveis localmente.

List Models

Lista todos os modelos disponíveis.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);  
Ollama->OllamaOptions->Host = "http://localhost:11434";  
ShowMessage(Ollama->_GetModels());
```

Get Tags

Lista todas as tags de modelo disponíveis localmente.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);  
Ollama->OllamaOptions->Host = "http://localhost:11434";  
ShowMessage(Ollama->_GetTags());
```

Show Model

Recupera informações detalhadas sobre um modelo específico.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);  
Ollama->OllamaOptions->Host = "http://localhost:11434";  
ShowMessage(Ollama->_ShowModel("llama3"));
```

Pull Model

Baixa um modelo da biblioteca de modelos Ollama.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);  
Ollama->OllamaOptions->Host = "http://localhost:11434";  
ShowMessage(Ollama->_PullModel("llama3"));
```

Ollama | Embeddings

Gera embeddings vetoriais a partir de entrada de texto usando modelos Ollama executados localmente. Os embeddings podem ser usados para busca semântica, clustering e outras tarefas de NLP.

Create Embeddings

Gera embeddings para a entrada de texto fornecida.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);  
Ollama->OllamaOptions->Host = "http://localhost:11434";  
ShowMessage(Ollama->_CreateEmbeddings("llama3", "Hello world"));
```

Grok

Grok é um assistente de IA conversacional desenvolvido pela xAI, projetado para fornecer respostas úteis e informativas com conhecimento em tempo real. Os modelos Grok são conhecidos por suas fortes capacidades de raciocínio e acesso a informações atualizadas.

A biblioteca `sgcWebSockets` fornece um componente Delphi **TsgcHTTP_API_Grok** para interagir com a API do Grok.

Grok API

A Grok API fornece acesso aos modelos Grok para construir aplicações com IA. A API suporta geração de texto, vision (compreensão de imagens), streaming e listagem de modelos.

Features

- **Messages**
 - [Exemplos de Grok Messages](#)
- **Vision**
 - [Grok Vision Examples](#)
- **Models**
 - [Exemplos de Modelos Grok](#)

Configuração

A API Grok utiliza chaves de API para autenticação. Visite seu [Console xAI](#) para recuperar a chave de API que você utilizará em suas requisições.

Lembre-se de que sua API key é um segredo! Não a compartilhe com outras pessoas nem a exponha em nenhum código do lado do cliente.

Esta **API Key** deve ser configurada na propriedade **GrokOptions.ApiKey** do componente.

```
TsgcHTTP_API_Grok *Grok = new TsgcHTTP_API_Grok(NULL);
Grok->GrokOptions->ApiKey = "YOUR_API_KEY";
```

Messages

Envia uma lista estruturada de mensagens de entrada com conteúdo de texto, e o modelo gerará a próxima mensagem na conversa.

- **_CreateMessage:** Cria uma mensagem com o modelo e o prompt de usuário especificados.
 - **Model:** O modelo a ser utilizado (por exemplo, grok-3).
 - **Message:** O conteúdo da mensagem do usuário.
- **_CreateMessageWithSystem:** Cria uma mensagem com um system prompt.
 - **System:** System prompt que define o comportamento do assistente.
- **_CreateMessageStream:** Cria uma mensagem com streaming (SSE) habilitado. Os eventos são entregues por meio do handler de evento `OnHTTPAPISSE`.

Vision

O Grok pode entender imagens passadas como conteúdo codificado em base64 dentro das mensagens.

- **_CreateVisionMessage:** Envia uma imagem com um prompt de texto.
 - **ImageBase64:** Os dados da imagem codificados em base64.
 - **MediaType:** O tipo MIME (image/jpeg, image/png, image/gif, image/webp).
 - **Prompt:** O prompt de texto para acompanhar a imagem.

Modelos

Lista os modelos Grok disponíveis.

- **_GetModels:** Lista todos os modelos disponíveis.

TsgcHTTP_API_Grok

Componente *TsgcHTTP_API_Grok*.

Introdução

Cliente REST API do xAI Grok para Chat Completions e listagem de modelos.

Introdução

Cliente REST API do xAI Grok para Chat Completions e listagem de modelos.

Configure a propriedade **GrokOptions** com sua chave de API e configurações opcionais de HTTP/retry/log e, em seguida, chame os métodos tipados como `CreateMessage/CreateChatCompletion` com um objeto de requisição e receba uma classe de resposta tipada.

Para mais informações sobre o provedor, consulte o guia [Grok](#).

Referência

- Propriedades — 7 propriedades disponíveis no componente.
- Métodos — 1 método público.
- Events — 0 eventos que você pode tratar.

Grok | Messages

Envia uma lista estruturada de mensagens de entrada com conteúdo de texto, e o modelo gerará a próxima mensagem na conversa.

Exemplo Simples

Envia uma mensagem Hello para o Grok.

```
TsgcHTTP_API_Grok *Grok = new TsgcHTTP_API_Grok(NULL);
Grok->GrokOptions->ApiKey = "API_KEY";
ShowMessage(Grok->_CreateMessage("grok-3", "Hello!"));
```

Exemplo de system prompt

Envia uma mensagem com um system prompt para controlar o comportamento do Grok.

```
TsgcHTTP_API_Grok *Grok = new TsgcHTTP_API_Grok(NULL);
Grok->GrokOptions->ApiKey = "API_KEY";
ShowMessage(Grok->_CreateMessageWithSystem("grok-3",
    "You are a helpful assistant that responds in Spanish.",
    "What is the capital of France?"));
```

Exemplo de Streaming

Utilize Server-Sent Events (SSE) para transmitir a resposta em tempo real. Atribua o handler de evento OnHTTPAPISSE para receber os eventos de streaming.

```
TsgcHTTP_API_Grok *Grok = new TsgcHTTP_API_Grok(NULL);
Grok->GrokOptions->ApiKey = "API_KEY";
Grok->OnHTTPAPISSE = OnSSEEvent;
Grok->_CreateMessageStream("grok-3", "Tell me a story.");

void __fastcall TForm1::OnSSEEvent(TObject *Sender, const String aEvent,
    const String aData, bool &Cancel)
{
    Memo1->Lines->Add(aData);
}
```

Grok | Vision

O Grok pode entender imagens passadas como conteúdo codificado em base64 dentro das mensagens. Utilize o endpoint de visão para enviar uma imagem junto com um prompt de texto e receber uma descrição ou análise.

Vision Example

Envia uma imagem codificada em base64 ao Grok para análise.

```
TsgcHTTP_API_Grok *Grok = new TsgcHTTP_API_Grok(NULL);
Grok->GrokOptions->ApiKey = "API_KEY";
ShowMessage(Grok->_CreateVisionMessage("grok-2-vision",
    "Describe this image.", vBase64, "image/jpeg"));
```


Grok | Models

Lista os modelos Grok disponíveis e suas informações básicas.

Exemplo de Listar Modelos

Recupera a lista de todos os modelos Grok disponíveis.

```
TsgcHTTP_API_Grok *Grok = new TsgcHTTP_API_Grok(NULL);  
Grok->GrokOptions->ApiKey = "API_KEY";  
ShowMessage(Grok->_GetModels);
```

Mistral AI

A Mistral AI é uma empresa francesa de inteligência artificial que desenvolve modelos de linguagem eficientes e poderosos. Seus modelos são conhecidos pelo forte desempenho em tarefas de raciocínio, programação e multilíngues, mantendo uma eficiência competitiva.

A biblioteca `sgcWebSockets` fornece um componente Delphi **TsgcHTTP_API_Mistral** para interagir com a API Mistral AI.

Mistral API

A API da Mistral fornece acesso aos modelos da Mistral para criar aplicações com IA. A API suporta geração de texto, visão (compreensão de imagens), streaming, embeddings, modo JSON e listagem de modelos.

Features

- **Mensagens**
 - [Exemplos de Mensagens Mistral](#)
- **Vision**
 - [Exemplos de Vision do Mistral](#)
- **Models**
 - [Exemplos de Models Mistral](#)
- **Embeddings**
 - [Exemplos de Embeddings Mistral](#)

Configuração

A API da Mistral usa chaves de API para autenticação. Visite sua página de [API Keys](#) no Console da Mistral para recuperar a chave de API que você usará em suas requisições.

Lembre-se de que sua API key é um segredo! Não a compartilhe com outras pessoas nem a exponha em nenhum código do lado do cliente.

Esta **API Key** deve ser configurada na propriedade **MistralOptions.ApiKey** do componente.

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);
Mistral->MistralOptions->ApiKey = "YOUR_API_KEY";
```

Messages

Envia uma lista estruturada de mensagens de entrada com conteúdo de texto, e o modelo gerará a próxima mensagem na conversa.

- **_CreateMessage:** Cria uma mensagem com o modelo e o prompt de usuário especificados.
 - **Model:** O modelo a ser utilizado (por exemplo, mistral-large-latest).
 - **Message:** O conteúdo da mensagem do usuário.
- **_CreateMessageWithSystem:** Cria uma mensagem com um system prompt.
 - **System:** System prompt que define o comportamento do assistente.
- **_CreateMessageStream:** Cria uma mensagem com streaming (SSE) habilitado. Os eventos são entregues por meio do handler de evento `OnHTTPAPISSE`.
- **_CreateMessageJSON:** Cria uma mensagem com o modo JSON habilitado. O modelo retornará uma resposta JSON válida.
 - **Model:** O modelo a ser utilizado.
 - **Message:** O conteúdo da mensagem do usuário.

Vision

O Mistral consegue entender imagens passadas como conteúdo codificado em base64 dentro das mensagens.

- **_CreateVisionMessage:** Envia uma imagem com um prompt de texto.
 - **ImageBase64:** Os dados da imagem codificados em base64.

- **MediaType:** O tipo MIME (image/jpeg, image/png, image/gif, image/webp).
- **Prompt:** O prompt de texto para acompanhar a imagem.

Modelos

Lista os modelos Mistral disponíveis.

- **_GetModels:** Lista todos os modelos disponíveis.

Embeddings

Obtém uma representação vetorial de uma entrada que pode ser usada para busca semântica, clustering e outras tarefas de aprendizado de máquina.

- **_CreateEmbeddings:** Cria um vetor de embedding representando o texto de entrada.
 - **Model:** O modelo a utilizar (por exemplo, mistral-embed).
 - **Input:** Texto de entrada para o qual obter embeddings.

TsgcHTTP_API_Mistral

Cliente REST API da Mistral para Chat Completions e Embeddings.

Introdução

Cliente REST API da Mistral para Chat Completions e Embeddings.

Introdução

Cliente REST API da Mistral para Chat Completions e Embeddings.

Configure a propriedade **MistralOptions** com sua API key e configurações opcionais de HTTP/retry/log, então chame os métodos tipados como `CreateMessage/CreateChatCompletion` com um objeto de requisição e receba uma classe de resposta tipada.

Para informações sobre o provider, consulte o guia [Mistral](#).

Referência

- Propriedades — 5 propriedades disponíveis no componente.
- Métodos — 2 métodos públicos.
- Eventos — 2 eventos que você pode tratar.

Mistral | Messages

Envia uma lista estruturada de mensagens de entrada com conteúdo de texto, e o modelo gerará a próxima mensagem na conversa.

Exemplo Simples

Envie uma mensagem Hello ao Mistral.

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);
Mistral->MistralOptions->ApiKey = "API_KEY";
ShowMessage(Mistral->_CreateMessage("mistral-large-latest", "Hello!"));
```

Exemplo de system prompt

Envia uma mensagem com um system prompt para controlar o comportamento do Mistral.

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);
Mistral->MistralOptions->ApiKey = "API_KEY";
ShowMessage(Mistral->_CreateMessageWithSystem("mistral-large-latest",
    "You are a helpful assistant that responds in Spanish.",
    "What is the capital of France?"));
```

Exemplo de Streaming

Utilize Server-Sent Events (SSE) para transmitir a resposta em tempo real. Atribua o handler de evento OnHTTPAPISSE para receber os eventos de streaming.

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);
Mistral->MistralOptions->ApiKey = "API_KEY";
Mistral->OnHTTPAPISSE = OnSSEEvent;
Mistral->_CreateMessageStream("mistral-large-latest", "Tell me a story.");

void __fastcall TForm1::OnSSEEvent(TObject *Sender, const String aEvent,
    const String aData, bool &Cancel)
{
    Memo1->Lines->Add(aData);
}
```

JSON Mode Example

Solicita uma resposta formatada em JSON da Mistral.

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);
Mistral->MistralOptions->ApiKey = "API_KEY";
ShowMessage(Mistral->_CreateMessageJSON("mistral-large-latest",
    "List 3 colors as JSON"));
```

Mistral | Vision

O Mistral consegue entender imagens passadas como conteúdo codificado em base64 dentro das mensagens. Use o endpoint de vision para enviar uma imagem junto com um prompt de texto e receber uma descrição ou análise.

Vision Example

Envia uma imagem codificada em base64 para a Mistral para análise.

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);
Mistral->MistralOptions->ApiKey = "API_KEY";
ShowMessage(Mistral->_CreateVisionMessage("mistral-large-latest",
    "Describe this image.", vBase64, "image/jpeg"));
```

Mistral | Models

Lista os modelos Mistral disponíveis e suas informações básicas.

Exemplo de Listar Modelos

Recupera a lista de todos os modelos Mistral disponíveis.

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);  
Mistral->MistralOptions->ApiKey = "API_KEY";  
ShowMessage(Mistral->_GetModels);
```

Mistral | Embeddings

Obtém uma representação vetorial de uma entrada que pode ser usada para busca semântica, clustering e outras tarefas de aprendizado de máquina.

Embeddings Example

Cria um vetor de embedding para uma entrada de texto.

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);  
Mistral->MistralOptions->ApiKey = "API_KEY";  
ShowMessage(Mistral->_CreateEmbeddings("mistral-embed", "Hello world"));
```


IoT

A Internet das coisas (IoT) refere-se ao conceito de estender a conectividade com a Internet para além das plataformas computacionais convencionais, como computadores pessoais e dispositivos móveis, alcançando qualquer gama de dispositivos físicos tradicionalmente "burros" ou não habilitados para a internet e objetos do cotidiano. Equipados com eletrônica, conectividade com a Internet e outras formas de hardware (como sensores), esses dispositivos podem se comunicar e interagir com outros pela Internet, e podem ser monitorados e controlados remotamente.

O pacote `sgcWebSockets` implementa os seguintes clientes IoT:

1. Amazon AWS IoT: O AWS IoT fornece comunicação bidirecional segura entre dispositivos conectados à Internet, como sensores, atuadores, microcontroladores embarcados ou eletrodomésticos inteligentes, e a Nuvem AWS. Isto permite que você colete dados de telemetria de vários dispositivos, e armazene e analise os dados. Você também pode criar aplicações que permitam que seus usuários controlem esses dispositivos a partir de seus celulares ou tablets.

2. Azure IoT Hub: o IoT Hub é um serviço gerenciado, hospedado na nuvem, que atua como um hub central de mensagens para comunicação bidirecional entre sua aplicação IoT e os dispositivos que ela gerencia. Você pode utilizar o Azure IoT Hub para criar soluções IoT com comunicações confiáveis e seguras entre milhões de dispositivos IoT e um backend de solução hospedado na nuvem.

IoT Amazon MQTT Client

O que é o AWS IoT?

O AWS IoT fornece comunicação segura e bidirecional entre dispositivos conectados à Internet, como sensores, atuadores, microcontroladores embarcados ou eletrodomésticos inteligentes, e a Nuvem AWS. Isto permite que você colete dados de telemetria de múltiplos dispositivos, e armazene e analise os dados. Você também pode criar aplicações que permitam que seus usuários controlem esses dispositivos a partir de seus telefones ou tablets.

Message broker

Fornecer um mecanismo seguro para que dispositivos e aplicações AWS IoT publiquem e recebam mensagens entre si. Você pode usar o protocolo MQTT diretamente ou MQTT sobre WebSocket para publicar e assinar.

O message broker do AWS IoT é um serviço de broker publish/subscribe que permite o envio e o recebimento de mensagens de e para o AWS IoT. Ao se comunicar com o AWS IoT, um cliente envia uma mensagem endereçada a um tópico como Sensor/temp/room1.

O message broker, por sua vez, envia a mensagem a todos os clientes que se registraram para receber mensagens daquele tópico. O ato de enviar a mensagem é chamado de publicação. O ato de registrar-se para receber mensagens de um filtro de tópico é chamado de assinatura.

O namespace de tópicos é isolado para cada par de conta e região AWS. Por exemplo, o tópico Sensor/temp/room1 de uma conta AWS é independente do tópico Sensor/temp/room1 de outra conta AWS. Isso também vale para as regiões. O tópico Sensor/temp/room1 na mesma conta AWS em us-east-1 é independente do mesmo tópico em us-east-2. O AWS IoT não suporta o envio e recebimento de mensagens entre contas e regiões AWS.

O message broker mantém uma lista de todas as sessões de cliente e as assinaturas de cada sessão. Quando uma mensagem é publicada em um tópico, o broker verifica as sessões com assinaturas que mapeiam para o tópico. O broker então encaminha a mensagem de publicação para todas as sessões que têm um cliente atualmente conectado.

Cliente MQTT

TsgcIoTAmazon_MQTT_Client é o componente usado para conectar ao AWS IoT. Um cliente pode conectar a apenas um dispositivo. O cliente conecta usando o protocolo MQTT simples e autentica usando um Certificado de Cliente X.509.

Para conectar ao AWS IoT, o cliente precisa das seguintes propriedades:

Amazon.ClientId: identificação do cliente, opcional.

Amazon.Endpoint: nome do servidor onde o cliente MQTT se conectará.

Amazon.Port: por padrão utiliza a porta 8883. Se a porta for 443, utiliza ALPN automaticamente para conectar (Requer versão Indy personalizada).

O AWS IoT Core suporta dispositivos e clientes que usam os protocolos MQTT e MQTT over WebSocket Secure (WSS) para publicar e assinar mensagens. A tabela a seguir lista os protocolos que os endpoints de dispositivo do AWS IoT suportam e os métodos de autenticação e portas que utilizam.

Protocol	Authenticati- tion	Port	ALPN Pro- tocol Na- me
MQTT sobre WebSocket	Signature Version 4	443	

MQTT sobre WebSocket	Custom Authentication	443	
MQTT	Certificado de cliente X.509	443	x-amzn-mqtt-ca
MQTT	Certificado de cliente X.509	8883	
MQTT	Custom Authentication	443	mqtt

Autenticação por certificados

Você precisa criar certificados em seu console Amazon AWS e definir o caminho onde eles estão armazenados.

Utilizando **OpenSSL** como IOHandler, você deve definir o certificado nos seguintes caminhos

Certificate.Enabled: defina como True se você quiser utilizar certificados.

Certificate.CertFile: caminho para o certificado de cliente X.509.

Certificate.KeyFile: caminho para o arquivo de chave cliente X.509.

Usando **SChannel** como IOHandler, primeiro converta o Certificado PEM + Chave para um certificado PFX. Isso requer os binários do OpenSSL:

```
openssl pkcs12 -inkey 884ccf73ff-private.pem.key -in 884ccf73ff-certificate.pem.crt -export -out 884ccf73ff-certi
```

Em seguida, defina os seguintes caminhos (não é necessário definir o arquivo de chave porque ele já está incluído no certificado).

Certificate.Enabled: defina como True se você quiser utilizar certificados.

Certificate.CertFile: caminho para o certificado PFX

SignatureV4 Authentication

Você precisa criar um usuário no seu console Amazon AWS e salvar as chaves Access e Secret, que serão usadas para assinar a requisição WebSocket.

SignatureV4.Enabled: defina como True se você quiser utilizar este tipo de autenticação.

SignatureV4.Region: a região onde seu dispositivo está localizado (exemplo: us-east-1).

SignatureV4.AccessKey: a access key criada em seu console Amazon ou obtida como credencial temporária.

SignatureV4.SecretKey: a chave secreta criada no seu console da amazon ou obtida como credencial temporária

SignatureV4.SessionToken: (condicional) se você estiver utilizando Temporary Security Credentials, defina aqui o token de segurança.

OpenSSL_Options: configuração das bibliotecas openssl.

APIVersion: permite definir qual API OpenSSL será usada.

oslAPI_1_0: usa a API 1.0 do OpenSSL, é a mais recente suportada pelo Indy

oslAPI_1_1: usa a API 1.1 do OpenSSL, requer nossa biblioteca Indy personalizada e permite usar as bibliotecas OpenSSL 1.1.1 (com suporte a TLS 1.3).

oslAPI_3_0: utiliza a API 3.0 do OpenSSL, requer nossa biblioteca Indy personalizada e permite utilizar as bibliotecas OpenSSL 3.0.0 (com suporte a TLS 1.3).

LibPath: aqui você pode configurar onde estão localizadas as bibliotecas openssl

oslpNone: este é o padrão, as bibliotecas openssl devem estar na mesma pasta onde está o binário ou em um caminho conhecido.

oslpDefaultFolder: define automaticamente o caminho openSSL onde as bibliotecas devem estar localizadas para todas as personalidades do IDE.

oslpCustomFolder: se esta for a opção selecionada, defina o caminho completo na propriedade LibPathCustom.

LibPathCustom: quando LibPath = oslpCustomFolder, defina aqui o caminho completo onde estão localizadas as bibliotecas openSSL.

UnixSymLinks: habilita ou desabilita o carregamento de SymLinks em sistemas Unix (por padrão está habilitado, exceto no OSX64):

oslsSymLinksDefault: por padrão estão habilitados, exceto sob OSX64 (após o MacOS Monterey falha ao tentar carregar a biblioteca sem versão.).

oslsSymLinksLoadFirst: Carrega os SymLinks e faz isso antes de tentar carregar as bibliotecas de versão.

oslsSymLinksLoad: Carrega os SymLinks após tentar carregar as bibliotecas de versão.

oslsSymLinksDontLoad: não carrega os SymLinks.

**SignatureV4 requer Indy 10.5.7+*

Custom Authentication

A autenticação personalizada permite que você defina como autenticar e autorizar clientes utilizando recursos authorize. O dispositivo passa as credenciais nos campos de cabeçalho da requisição ou em parâmetros de query (para os protocolos MQTT over WebSockets) ou nos campos de nome de usuário e senha da mensagem MQTT CONNECT (para os protocolos MQTT e MQTT over WebSockets).

CustomAuthentication.Enabled: defina como True se você quiser utilizar este tipo de Autenticação.

CustomAuthentication.Parameters: defina aqui os parâmetros de query que serão passados ao servidor (por padrão é /mqtt)

CustomAuthentication.Headers: aqui você pode colocar os campos de cabeçalho personalizados.

CustomAuthentication.WebSockets: se definido como true, a conexão funcionará sobre o protocolo WebSocket, caso contrário funcionará sobre TCP puro.

MQTTAuthentication.Enabled: se você precisar passar o nome de usuário/senha na conexão mqtt, habilite esta propriedade

MQTTAuthentication.Username: nome de usuário da conexão mqtt

MQTTAuthentication.Password: segredo da conexão mqtt.

O cliente pode enviar opcionalmente um ClientId para identificar a conexão do cliente; então, outros clientes podem se inscrever para receber uma notificação toda vez que este cliente conectar, se inscrever, desconectar...

Autorização

Se você não conseguir conectar utilizando a porta 8883 e utilizar TCP como transporte (que é o padrão), a Amazon utiliza a "AWS IoT Core policy" para fornecer ou não autorização a clientes e inscrições. Muito provavelmente você deve autorizar seu client id.

Entre no seu console Amazon AWS, vá para o IoT Core e acesse o menu "Secure/Policies", lá selecione a política anexada ao seu IoT Thing e verifique no final como a conexão está configurada. Exemplo:

```
{
  "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-*"
  ]
}
```

Esta configuração significa que apenas clientes com ID: sdk-java, basicPubSub e sdk-nodejs-* terão permissão para conectar. Altere de acordo e tente novamente.

Se ainda não funcionar, habilite o log e verifique no cloudwatch o motivo pelo qual você não consegue conectar.

Outras propriedades

MQTTHeartBeat: se habilitado, tenta manter a conexão MQTT ativa enviando um ping a cada x segundos.

Interval: número de segundos entre cada ping.

MQTTAuthentication: se habilitado, inclui na conexão MQTT o nome de usuário e a senha

UserName: nome do usuário

Senha: secret string

WatchDog: se habilitado, quando uma desconexão inesperada é detectada, tenta reconectar-se ao servidor automaticamente.

Interval: segundos antes das tentativas de reconexão.

Attempts: número máximo de tentativas de reconexão; zero significa ilimitado.

LogFile: se habilitado, salva as mensagens do socket em um arquivo de log (útil para depuração). O acesso ao arquivo de log não é thread safe se for acessado a partir de várias threads.

Enabled: se habilitado, cada vez que uma mensagem é recebida e enviada pelo socket ela será salva em um arquivo.

FileName: caminho completo para o nome do arquivo.

Implementation

A implementação do Amazon MQTT é baseada na versão MQTT 3.1.1, mas desvia da especificação da seguinte forma:

- No AWS IoT, assinar um tópico com Quality of Service (QoS) 0 significa que uma mensagem é entregue zero ou mais vezes. Uma mensagem pode ser entregue mais de uma vez. Mensagens entregues mais de uma vez podem ser enviadas com um packet ID diferente. Nesses casos, a flag DUP não é definida.
- O AWS IoT não suporta publicação e assinatura com QoS 2. O broker de mensagens do AWS IoT não envia um PUBACK ou SUBACK quando QoS 2 é solicitado.
- Ao responder a uma requisição de conexão, o message broker envia uma mensagem CONNACK. Esta mensagem contém uma flag para indicar se a conexão está retomando uma sessão anterior. O valor dessa flag pode estar incorreto se dois clientes MQTT se conectarem com o mesmo client ID simultaneamente.
- Quando um cliente assina um tópico, pode haver um atraso entre o momento em que o message broker envia um SUBACK e o momento em que o cliente começa a receber novas mensagens correspondentes.
- A especificação MQTT fornece uma disposição para que o publicador solicite que o broker retenha a última mensagem enviada a um tópico e a envie a todos os futuros assinantes do tópico. O AWS IoT não suporta mensagens retidas. Se for feita uma requisição para reter mensagens, a conexão é desconectada.
- O message broker usa o client ID para identificar cada cliente. O client ID é passado do cliente para o message broker como parte do payload MQTT. Dois clientes com o mesmo client ID não podem estar conectados simultaneamente ao message broker. Quando um cliente se conecta ao message broker usando um client ID que outro cliente está usando, uma mensagem CONNACK é enviada a ambos os clientes e o cliente atualmente conectado é desconectado.
- Em raras ocasiões, o message broker pode reenviar a mesma mensagem PUBLISH lógica com um id de pacote diferente.
- O message broker não garante a ordem em que mensagens e ACK são recebidos.

Conectar ao AWS IoT

Primeiro, você deve fazer login no seu console AWS, registrar um novo dispositivo e criar um certificado X.509 para esse dispositivo. Uma vez feito isso, você pode criar um novo `TsgcIoTAmazon_MQTT_Client` e conectar-se ao AWS IoT Server. Por exemplo:

```
oClient = new TsgcIoTAmazon_MQTT_Client();
oClient->Amazon->Endpoint = "a2ohgdjqitsmij-ats.iot.us-west-2.amazonaws.com";
oClient->Amazon->ClientId = "sgcWebSockets";
oClient->Certificate->CertFile = "amazon-certificate.pem.crt";
oClient->Certificate->KeyFile = "amazon-private.pem.key";
oClient->OnMQTTConnect = OnMQTTConnectEvent;
oClient->Active = true;

void OnMQTTConnect(TsgcWSConnection *Connection, const bool Session, const TmqttConnReturnCode
ReturnCode)
{
    ShowMessage("Connected to AWS");
}
```

Tópicos

O message broker usa tópicos para rotear mensagens dos clientes publicadores para os clientes assinantes. A barra (/) é usada para separar a hierarquia de tópicos. A tabela a seguir lista os wildcards que podem ser usados no filtro de tópico quando você assina. # Deve ser o último caractere no tópico que você está assinando. Funciona como um wildcard correspondendo à árvore atual e a todas as subárvores.

Por exemplo, uma inscrição em `Sensor/#` recebe mensagens publicadas em `Sensor/`, `Sensor/temp`, `Sensor/temp/room1`, mas não as mensagens publicadas em `Sensor`.

+ Corresponde exatamente a um item na hierarquia de tópicos. Por exemplo, uma assinatura de `Sensor/+room1` recebe mensagens publicadas em `Sensor/temp/room1`, `Sensor/moisture/room1` e assim por diante.

```
oClient = new TsgcIoTAmazon_MQTT_Client();
...
oClient->OnSubscribe = OnSubscribeEvent;

vPacketIdentifier = oClient->Subscribe("Sensor/moisture/room1");

void OnMQTTSubscribe(TsgcWSConnection *Connection, Word aPacketIdentifier, TsgcWSSUBACKS *aCodes)
{
    if (vPacketIdentifier == aPacketIdentifier)
    {
        ShowMessage("Subscribed to topic Sensor/moisture/room1");
    }
}

// Client, can send a message using Publish method.
oClient->Publish("Sensor/moisture/room1", "{\"temp\"=10}");

// Messages received from server, are dispatched OnMQTTPublishEvent.
// For extended payload access (string, bytes or stream), use OnMQTTPublishEx.
void OnMQTTPublish(TsgcWSConnection *Connection, string aTopic, string aText)
{
    DoLog("Received Message: " + aTopic + " " + aText);
}
```

Tópicos reservados

Os métodos a seguir são utilizados para assinar / publicar em tópicos reservados.

Subscribe_ClientConnected(const aClientId: String): O AWS IoT publica neste tópico quando um cliente MQTT com o client ID especificado se conecta ao AWS IoT

Subscribe_ClientDisconnected(const aClientId: String): o AWS IoT publica neste tópico quando um cliente MQTT com o client ID especificado se desconecta do AWS IoT

Subscribe_ClientSubscribed(const aClientId: String): o AWS IoT publica neste tópico quando um cliente MQTT com o client ID especificado assina um tópico MQTT

Subscribe_ClientUnsubscribed(const aClientId: String): O AWS IoT publica neste tópico quando um cliente MQTT com o client ID especificado cancela a assinatura de um tópico MQTT

Publish_Rule(const aRuleName, aText: String): Um dispositivo ou uma aplicação publica neste tópico para acionar regras diretamente

Publish_DeleteShadow(const aThingName, aText: String): Um dispositivo ou uma aplicação publica neste tópico para excluir um shadow

Subscribe_DeleteShadow(const aThingName: String): Um dispositivo ou uma aplicação se inscreve neste tópico para excluir um shadow

Subscribe_ShadowDeleted(const aThingName: String): O serviço Device Shadow envia mensagens para este tópico quando um shadow é excluído

Subscribe_ShadowRejected(const aThingName: String): O serviço Device Shadow envia mensagens para este tópico quando uma requisição para excluir um shadow é rejeitada

Publish_ShadowGet(const aThingName, aText: String): Uma aplicação ou um thing publica uma mensagem vazia neste tópico para obter um shadow

Subscribe_ShadowGet(const aThingName: String): Uma aplicação ou um thing assina este tópico para obter um shadow

Subscribe_ShadowGetAccepted(const aThingName: String): O serviço Device Shadow envia mensagens para este tópico quando uma requisição por um shadow é feita com sucesso

Subscribe_ShadowGetRejected(const aThingName: String): O serviço Device Shadow envia mensagens para este tópico quando uma requisição de shadow é rejeitada

Publish_ShadowUpdate(const aThingName, aText: String): Um thing ou aplicação publica neste tópico para atualizar um shadow

Subscribe_ShadowUpdateAccepted(const aThingName: String): O serviço Device Shadow envia mensagens para este tópico quando uma atualização é feita com sucesso em um shadow

Subscribe_ShadowUpdateRejected(const aThingName: String): O serviço Device Shadow envia mensagens a este tópico quando uma atualização de um shadow é rejeitada

Subscribe_ShadowUpdateDelta(const aThingName: String): O serviço Device Shadow envia mensagens para este tópico quando uma diferença é detectada entre as seções reported e desired de um shadow

Subscribe_ShadowUpdateDocuments(const aThingName: String): O AWS IoT publica um documento de estado neste tópico sempre que uma atualização do shadow é realizada com sucesso

Sessões Persistentes

Uma sessão persistente representa uma conexão contínua com um broker de mensagens MQTT. Quando um cliente conecta ao broker de mensagens do AWS IoT usando uma sessão persistente, o broker de mensagens salva todas as assinaturas que o cliente faz durante a conexão. Quando o cliente se desconecta, o broker de mensagens armazena as mensagens QoS 1 não confirmadas e as novas mensagens QoS 1 publicadas em tópicos aos quais o cliente está assinado. Quando o cliente reconecta à sessão persistente, todas as assinaturas são restabelecidas e todas as mensagens armazenadas são enviadas ao cliente a uma taxa máxima de 10 mensagens por segundo.

Você cria uma sessão MQTT persistente definindo o parâmetro **cleanSession** como False no evento **OnMQTTBeforeConnect**. Se não existir sessão para o cliente, uma nova sessão persistente é criada. Se já existir uma sessão para o cliente, ela é retomada.

Os dispositivos precisam olhar o atributo **Session** no evento **OnMQTTConnect** para determinar se uma sessão persistente está presente. Se **Session for True**, uma sessão persistente está presente e as mensagens armazenadas são entregues ao cliente. Se **Session for False**, nenhuma sessão persistente está presente e o cliente deve assinar novamente seus filtros de tópico.

As sessões persistentes têm um período de expiração padrão de 1 hora. O período de expiração começa quando o message broker detecta que um cliente se desconecta (desconexão MQTT ou timeout). O período de expiração da sessão persistente pode ser aumentado por meio do processo padrão de aumento de limite. Se um cliente não retomar sua sessão dentro do período de expiração, a sessão é encerrada e quaisquer mensagens armazenadas associadas são descartadas. O período de expiração é aproximado; as sessões podem ser persistidas por até 30 minutos a mais (mas não menos) do que a duração configurada.

Credenciais temporárias

O AWS IoT Core pode trabalhar com Credenciais Temporárias obtidas através de Identity Pools, há 2 tipos de Identidades:

- **UnAuthenticated:** requer apenas a definição do tipo de policy no IAM

- **Authenticated:** requer definir o tipo de política nas políticas do IAM e do AWS IoT Core

Não autenticado

Se você estiver utilizando credenciais não autenticadas, basta anexar a policy à Role UnAuthenticated criada automaticamente no menu IAM. Em seguida, configure o cliente definindo a Access, a Secret Key e o Token retornados pelo serviço Cognito.

Encontre abaixo um código em .NET para obter credenciais não autenticadas

```
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;
```

Authenticated

Credenciais autenticadas, requer anexar a política na Authenticated Role criada automaticamente no menu IAM e anexar a política do usuário nas políticas do AWS IoT Core.

Portanto, crie uma nova política no menu de políticas do IoT Core e, toda vez que um novo usuário se autenticar, anexe esta política a esse usuário.

Você pode utilizar o seguinte comando da AWS para anexar uma policy ou criar uma função lambda.

```
aws iot attach-policy --policy-name PolicyName --target us-east-1:XXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Provisionamento de Dispositivo

O serviço Fleet Provisioning suporta as seguintes operações da API MQTT:

- **CreateCertificateFromCsr:** Cria um certificado a partir de uma certificate signing request (CSR). O AWS IoT fornece certificados de cliente assinados pela autoridade certificadora (CA) Amazon Root. O novo certificado tem o status PENDING_ACTIVATION. Quando você chama RegisterThing para provisionar um thing com este certificado, o status do certificado muda para ACTIVE ou INACTIVE, conforme descrito no template.
- **CreateKeysAndCertificate:** Cria novas chaves e um certificado. O AWS IoT fornece certificados de cliente assinados pela Amazon Root certificate authority (CA). O novo certificado tem um status PENDING_ACTIVATION. Quando você chama RegisterThing para provisionar um thing com este certificado, o status do certificado muda para ACTIVE ou INACTIVE conforme descrito no template.
- **RegisterThing:** Provisiona um thing usando um template predefinido.

[CreateCertificateFromCsr](#)

Utilize o método `CreateCertificateFromCsr` passando o `CertificateSigningRequest` como parâmetro para criar o certificado. Para receber a resposta a esta requisição, primeiro inscreva-se nos seguintes métodos: `SubscribeCreateCertificateFromCsrResponse` e `SubscribeCreateCertificateFromCsrError`

CreateKeysAndCertificate

Use o método `CreateKeysAndCertificate` para criar um novo certificado e chaves. Para receber a resposta a esta requisição, primeiro assine os seguintes métodos `SubscribeCreateKeysAndCertificateResponse` e `SubscribeCreateKeysAndCertificateError`

RegisterThing

Utilize o método `RegisterThing` para registrar uma nova thing passando como parâmetro o `Template Name` e o `Payload` em formato JSON. Para receber a resposta a esta requisição, primeiro assine os seguintes métodos `SubscribeRegisterThingResponse` e `SubscribeRegisterThingError`.

IoT Azure MQTT Client

O que é o Azure IoT Hub?

O IoT Hub é um serviço gerenciado, hospedado na nuvem, que atua como um hub central de mensagens para comunicação bidirecional entre sua aplicação IoT e os dispositivos que ela gerencia. Você pode utilizar o Azure IoT Hub para construir soluções IoT com comunicações confiáveis e seguras entre milhões de dispositivos IoT e um backend de solução hospedado na nuvem. Você pode conectar praticamente qualquer dispositivo ao IoT Hub.

O IoT Hub suporta comunicações tanto do dispositivo para a nuvem quanto da nuvem para o dispositivo. O IoT Hub suporta múltiplos padrões de mensageria, como telemetria dispositivo-para-nuvem, upload de arquivos de dispositivos e métodos request-reply para controlar seus dispositivos a partir da nuvem. O monitoramento do IoT Hub ajuda você a manter a integridade da sua solução rastreando eventos como criação de dispositivos, falhas de dispositivos e conexões de dispositivos.

Os recursos do IoT Hub ajudam você a construir soluções de IoT escaláveis e com todos os recursos, como gerenciar equipamentos industriais utilizados na fabricação, rastrear ativos valiosos na área da saúde e monitorar o uso de prédios de escritórios.

Message broker

O IoT Hub fornece um canal de comunicação seguro para seus dispositivos enviarem dados. O IoT Hub e os SDKs de dispositivo suportam os seguintes protocolos para conectar dispositivos:

- **MQTT**
- **MQTT** sobre **WebSockets**

Múltiplos tipos de autenticação suportam uma variedade de capacidades de dispositivo:

- Autenticação baseada em token **SAS** para começar rapidamente com a sua solução IoT.
- Autenticação individual por **certificado X.509** para uma autenticação segura e baseada em padrões.

Cliente MQTT

TsgcIoTazure_MQTT_Client é o componente utilizado para conectar ao Azure IoT. Um cliente pode conectar a apenas um dispositivo. O cliente conecta utilizando o protocolo MQTT simples e autentica utilizando SAS / X.509 Client Certificate.

Para conectar ao Azure IoT Hub, o cliente precisa das seguintes propriedades:

Azure.IoTHub: nome do servidor ao qual o cliente MQTT conectará.

Azure.Deviceld: nome do dispositivo no Azure IoT Hub.

O Azure permite múltiplos tipos de autenticação, por padrão usa tokens SAS.

SAS Authentication

SAS.Enabled: habilite se a autenticação utilizar SAS.

SAS.SecretKey: o SAS Token da sua conta do Azure IoT.

SAS.KeyName: o Shared Access Key Name.

SAS.Expiry: define o número de minutos antes de o SAS Token expirar. O valor padrão é 1440 (24 horas).

Se você tiver uma connection string, pode ler os valores dela automaticamente utilizando o método **ReadConnectionString**. Exemplo:

```
ReadConnectionString('HostName=yourhub.azure-
devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=Yj7RRPnkSDTv+UCFLgwIP/
FrbDymZv4qVAIoTLHUFR8=');
```

Certificados X509

Utilizando **OpenSSL** como IOHandler

Certificate.Enabled: habilite se a autenticação utilizar certificados.

Certificate.CertFile: caminho para o certificado de cliente X.509.

Certificate.KeyFile: caminho para o arquivo de chave cliente X.509.

Certificate.Password: se o certificado tiver uma senha, defina-a aqui.

Version: versão TLS, por padrão utiliza TLS 1.0

Utilizando **SChannel** como IOHandler

Certificate.Enabled: habilite se a autenticação utilizar certificados.

Certificate.CertFile: caminho para o certificado PFX (primeiro o certificado deve ser convertido para PFX).

[Saiba Mais.](#)

Certificate.Password: se o certificado tiver uma senha, defina-a aqui.

Version: versão TLS, por padrão utiliza TLS 1.0

Outras propriedades:

MQTTHeartBeat: se habilitado, tenta manter a conexão MQTT ativa enviando um ping a cada x segundos.

Interval: número de segundos entre cada ping.

WatchDog: se habilitado, quando uma desconexão inesperada é detectada, tenta reconectar-se ao servidor automaticamente.

Interval: segundos antes das tentativas de reconexão.

Attempts: número máximo de tentativas de reconexão; zero significa ilimitado.

LogFile: se habilitado, salva as mensagens do socket em um arquivo de log (útil para depuração). O acesso ao arquivo de log não é thread safe se for acessado a partir de várias threads.

Enabled: se habilitado, cada vez que uma mensagem é recebida e enviada pelo socket ela será salva em um arquivo.

FileName: caminho completo para o nome do arquivo.

A implementação Azure MQTT é baseada na versão 3.1.1 do MQTT, mas desvia da especificação da seguinte forma:

- O IoT Hub não suporta mensagens QoS 2. Se uma aplicação de dispositivo publicar uma mensagem com QoS 2, o IoT Hub fecha a conexão de rede.
- O IoT Hub não persiste mensagens Retain. Se um dispositivo enviar uma mensagem com a flag RETAIN definida como 1, o IoT Hub adiciona a propriedade de aplicação x-opt-retain à mensagem. Neste caso, em vez de persistir a mensagem retain, o IoT Hub a passa para a aplicação backend.
- O IoT Hub suporta apenas uma conexão MQTT ativa por dispositivo. Qualquer nova conexão MQTT em nome do mesmo ID de dispositivo faz com que o IoT Hub derrube a conexão existente.

Conectar ao Azure IoT Hub

Primeiro, você deve fazer login na sua conta Azure, registrar um novo dispositivo e criar um método de autenticação para este dispositivo. Feito isso, você pode criar um novo `TsgcloTAzure_MQTT_Client` e conectar ao Azure IoT Hub.

Por exemplo:

```
TsgcIoTAzure_MQTT_Client *oClient = new TsgcIoTAzure_MQTT_Client();
oClient->Azure->IoTHub = "youriothub.azure-devices.net";
oClient->Azure->DeviceId = "YourDeviceId";
oClient->SAS->Enabled = true;
oClient->SAS->SecretKey = "YourSecretKey";
oClient->OnMQTTConnect = OnMQTTConnectEvent;
oClient->Active = true;

private void OnMQTTConnect(TsgcWSConnection *Connection, const bool Session, const TmqttConnReturnCode *ReturnCoc
{
    ShowMessage("Connected to Azure IoT Hub");
}
```

Device To Cloud

Ao enviar informações do app do dispositivo para o back end da solução, o IoT Hub expõe as seguintes opções:

1. **Mensagens dispositivo-para-nuvem** para telemetria de séries temporais e alertas.

```
oClient->Send_DeviceToCloud("{\"temp\": 10}", azuIoTQoS1);
```

Você pode enviar propriedades de chave-valor usando uma TStringList; basta preencher a TStringList com as propriedades de mensagem desejadas e passá-las como argumento.

```
TStringList *oProperties = new TStringList;
try
{
    oProperties->AddPair("prop_name1", "prop_value1");
    oProperties->AddPair("prop_name2", "prop_value2");
    oClient->Send_DeviceToCloud("{\"temp\": 10}", oProperties, azuIoTQoS1);
}
__finally
{
    oProperties->Free();
}
```

Se você precisar definir o **ContentType** e o **ContentEncoding** da mensagem, você deve adicionar esses valores à Lista de Properties. Os nomes dessas propriedades são definidos pela Azure.

Name	Value
\$.ct	application/json
\$.ce	utf-8

```
TStringList *oProperties = new TStringList();
try
{
    oProperties->AddPair("$.ct", "application/json");
    oProperties->AddPair("$.ce", "utf-8");
    oClient->Send_DeviceToCloud("{\"temp\": 10}", oProperties, azuIoTQoS1);
}
__finally
{
    oProperties->Free();
}
```

2. **Propriedades reportadas do device twin** para reportar informações de estado do dispositivo, como capacidades disponíveis, condições ou o estado de workflows de longa duração. Por exemplo, configuração e atualizações de software.

```
oClient->Set_DeviceTwinsProperties("1", "{\"sgc\":1}");
```

Cloud To Device

O IoT Hub fornece três opções para que os apps de dispositivo exponham funcionalidades a um app de back-end:

1. **Métodos diretos** para comunicações que exigem confirmação imediata do resultado. Métodos diretos são frequentemente usados para o controle interativo de dispositivos, como ligar um ventilador.

```
oClient->Subscribe_DirectMethod();
```

Você pode responder a métodos públicos usando o método a seguir.

```
oClient->RespondPublicMethod(RequestId, Status, "Your Response", azuIoTQoS1);
```

2. **Propriedades desejadas do Twin** para comandos de longa duração destinados a colocar o dispositivo em um determinado estado desejado. Por exemplo, definir o intervalo de envio de telemetria para 30 minutos. Você pode obter as propriedades utilizando o método a seguir.

```
oClient->Get_DeviceTwinsProperties("1");
```

3. **Mensagens cloud-to-device** para notificações unidirecionais ao aplicativo do dispositivo. Para obter mensagens, primeiro você deve assinar.

```
oClient->Subscribe_CloudToDevice;
```

As mensagens são recebidas no evento **OnMQTTPublish** (payload de texto) ou no evento **OnMQTTPublishEx** (payload como `TsgcWSMQTTPublishData` com as propriedades `Value`, `Bytes` e `Stream`).

```
void AzureIoTMQTTPublish(TsgcWSConnection *Connection, string aTopic, string aText)
{
    DoLog("Received Message: " + aTopic + " " + aText);
}
```

Upload Files

O IoT hub facilita os uploads de arquivos a partir de dispositivos conectados, fornecendo-lhes URIs de shared access signature (SAS) ou certificados X509.

Se você selecionar SAS, deve definir as seguintes propriedades:

- **Azure.IoTHub:** exemplo youriothub.azure-devices.net
- **Azure.Deviceld:** exemplo: myDevice
- **SAS.SecretKey:** exemplo: Yj7RRPnkSDTv+UCFLgwIP/FrbDymZv4qVAIoTLHUFR8=
- **SAS.KeyName:** exemplo: iothubowner
- **SAS.Enabled:** o valor deve ser definido como true.

Se você selecionar certificados X509, deve definir as seguintes propriedades:

- **Certificate.CertFile:** o caminho para o seu certificado PEM.
- **Certificate.KeyFile:** o caminho para o seu arquivo de chave PEM.
- **Certificate.Enabled:** o valor deve ser definido como true.

Utilize o método **UploadFile** para fazer upload de um arquivo para os Servidores Azure. Se o parâmetro `Overwrite` estiver definido como true, ele substituirá o arquivo existente. Se o parâmetro `Overwrite` estiver definido como false, ele só fará o upload se o arquivo não existir e gerará um erro se ele existir (esta é a opção padrão).

```
void UploadFileToAzure()
{
    TOpenDialog* oDialog = new TOpenDialog(NULL);
    try
    {
```

```

        if (oDialog->Execute())
        {
            AnsiString fileName = oDialog->FileName;
            AzureIoT::UploadFile(fileName.c_str());
        }
    }__finally
    {
        oDialog->Free();
    }
}

```

Device Provisioning Service

O Azure IoT permite que você registre dispositivos via código usando o DPS. Atualmente, a biblioteca suporta o registro de um dispositivo passando o Scope Id e o Registration Id como parâmetros.

```

TsgcIoTAzure_MQTT_Client* oClient = new TsgcIoTAzure_MQTT_Client(NULL);
try
{
    oClient->Certificate->CertFile = L"cert.pem";
    oClient->Certificate->KeyFile = L"key.pem";
    oClient->Certificate->Enabled = true;
    TsgcIoT_Azure_OperationRegistrationState* oResponse = new TsgcIoT_Azure_OperationRegistrationState();
    try
    {
        if (oClient->ProvisioningDeviceClient_Register(L"scope_id", L"registration_id", oResponse))
            ShowMessage(L"#Provisioning Register OK: " + oResponse->Status);
        else
            ShowMessage(L"#Provisioning Register Error: " + oResponse->Status);
    }
    __finally
    {
        delete oResponse;
    }
}__finally
{
    delete oClient;
}

```

Azure IoT Explorer

Você pode utilizar a aplicação **Azure IoT Explorer** para interagir com os dispositivos conectados ao seu IoT Hub. Você pode ver as mensagens de telemetria recebidas, os dispositivos registrados e muito mais. A aplicação é gratuita e pode ser baixada em:

<https://github.com/Azure/azure-iot-explorer/releases>

HTTP

O protocolo HTTP permite buscar recursos de servidores, como imagens e documentos HTML. É um protocolo cliente-servidor, o que significa que o cliente solicita ao servidor os recursos de que necessita.

Quando um cliente deseja se conectar a um servidor, ele segue os próximos passos:

1. Abra uma nova conexão TCP
2. Envie uma mensagem ao servidor com os dados solicitados

```
GET / HTTP/1.1
Host: server.com
Accept-Language: en-us
```

3. Leia a resposta enviada pelo servidor

```
HTTP/1.1 200 OK
Server: Apache
Content-Length: 120
Content-Type: text/html
...
```

Componentes

- **HTTP/2:** HTTP/2 (ou h2) é um protocolo binário que traz push, multiplexação de streams e controle de frames para a web.
- **Cliente HTTP/1:** um componente não-visual que herda do componente cliente TIdHTTP.
- **OAuth2:** OAuth2 permite que aplicações de terceiros recebam acesso limitado a um serviço HTTP.
- **JWT:** O JWT permite criar dados com assinatura e/ou criptografia opcionais, cujo payload contém JSON que afirma um determinado número de claims.
- **Amazon SQS:** um serviço de enfileiramento de mensagens totalmente gerenciado para microsserviços, sistemas distribuídos e aplicações serverless.
- **Google Cloud Pub/Sub:** fornece mensageria entre aplicações e foi projetado para fornecer mensageria assíncrona, confiável e de muitos-para-muitos entre aplicações.
- **Google Calendar:** permite que você utilize a Google Calendar API V3: obter Calendars, eventos, sincronizar com seu próprio calendário...
- **Google FCM:** envia notificações usando o Firebase Cloud Messaging.

HTTP/2

O HTTP/2 é uma evolução do protocolo HTTP 1.1; basicamente tenta ser mais eficiente ao usar redes. A semântica é a mesma, portanto é projetado para ser compatível com protocolos antigos.

HTTP 1.1 Limitations

HTTP 1.1 é limitado a processar uma requisição por conexão, então geralmente os clientes utilizam mais de uma conexão para solicitar arquivos dos servidores. Mas isso gera um problema, porque quando há muitas conexões TCP abertas, há uma disputa entre os clientes para utilizar os recursos do servidor e o desempenho fica cada vez menor à medida que mais clientes conectam aos servidores.

Recursos principais

- HTTP/2 é um protocolo binário (lembre-se de que HTTP 1.1 é um protocolo de texto).
- HTTP/2 funciona sobre TLS e ALPN.
- É multiplexado (permite que você envie mais de uma requisição sobre uma única Conexão TCP).
- O servidor pode fazer push de respostas aos clientes.
- Reduz os Round Trip Times, de modo que os clientes possam carregar mais rapidamente.

O HTTP/2 introduz outras melhorias, mais detalhes: [RFC7540](#).

HTTP/2 requer nossa versão personalizada do Indy porque ele requer o protocolo ALPN.

Componentes

- **TsgcHTTP2Client**: componente cliente que suporta totalmente o protocolo HTTP/2 (sgcWebSockets 100% código Pascal, sem bibliotecas externas).
- **TsgcWebsocketHTTPServer**: componente de servidor que suporta totalmente o protocolo HTTP/2 (código sgcWebSockets 100% Pascal, sem bibliotecas externas). Por padrão, o HTTP/2 está desabilitado; você pode habilitá-lo utilizando a propriedade HTTPOptions e definindo Enable = true.
- **TsgcWebsocketServer_HTTPAPI**: componente de servidor que suporta o protocolo HTTP/2 (implementação Microsoft e requer Windows 2016+ ou Windows 10+).
- **Servidores DataSnap**: o servidor datasnap também pode suportar o protocolo HTTP/2.

APIs

- **Apple Push Notifications**: envia notificações voltadas ao usuário para o dispositivo do usuário a partir de um provedor de servidor.

TsgcHTTP2Client

O *TsgcHTTP2Client* implementa um Componente Cliente HTTP/2 e pode conectar a servidores HTTP/2.

Introdução

O *TsgcHTTP2Client* implementa o Componente Cliente HTTP/2 e pode conectar a servidores HTTP/2. Siga os passos abaixo para configurar este componente:

1. Crie uma nova instância do componente **TsgcHTTP2Client**.
2. Envie a requisição ao servidor e processe a resposta utilizando o evento *OnHTTP2Response*. exemplo:

```
TsgcHTTP2Client *oClient = new TsgcHTTP2Client();
oClient->OnHTTP2Response = OnHTTP2ResponseEvent;
oClient->Get("https://www.google.com");

void OnHTTP2ResponseEvent(TObject *Sender, const
    TsgcHTTP2ConnectionClient *Connection, const TsgcHTTP2RequestProperty *Request,
    const TsgcHTTP2ResponseProperty *Response)
{
    ShowMessage(Response->DataString);
}
```

Referência

- Propriedades — 19 propriedades disponíveis no componente.
- Métodos — 19 métodos públicos.
- Eventos — 13 eventos que você pode tratar.

Guides

- **Requisições**
 - [Método de Requisição HTTP/2](#)
 - [HTTP/2 Server Push](#)
 - [Baixar Arquivo](#)
 - [Respostas Parciais HTTP/2](#)
 - [Cabeçalhos HTTP/2](#)
- **Conexão**
 - [Cliente Fechar Conexão](#)
 - [Cliente Manter Conexão Ativa](#)
 - [HTTP/2 Motivo da Desconexão](#)
 - [Requisições Pendentes do Cliente](#)
- **Authentication**
 - [Client Authentication](#)
 - [HTTP/2 and OAuth2](#)
- **Classes**
 - [TsgcHTTP2ConnectionClient](#)
 - [TsgcHTTP2RequestProperty](#)

- [TsgcHTTP2ResponseProperty](#)

TsgcHTTP2Client | Request HTTP/2 Method

O HTTP/2 Client pode trabalhar em modo bloqueante e não bloqueante; internamente o componente trabalha em um thread secundário e as requisições são processadas de forma assíncrona, mas você pode fazer uma chamada de requisição e aguardar até que esta requisição seja concluída.

Veja abaixo um exemplo de como o cliente pode solicitar uma página HTML a um servidor HTTP/2 e como pode funcionar em ambos os modos.

Asynchronous Mode

Obtenha a seguinte url: <https://www.google.com> e seja notificado quando o cliente receber a resposta completa. Depois de chamar o método **GETASYNC**, o processo continua e o evento OnHTTP2Response é chamado quando a resposta é recebida.

```
TsgcHTTP2Client *oClient = new TsgcHTTP2Client();
oClient->OnHTTP2Response = OnHTTP2ResponseEvent;
oClient->GetAsync("https://www.google.com");
void OnHTTP2ResponseEvent(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection,
    const TsgcHTTP2RequestProperty *Request, const TsgcHTTP2ResponseProperty *Response)
{
    ShowMessage(Response->Headers->Text + #13#10 + Response->DataString);
}
```

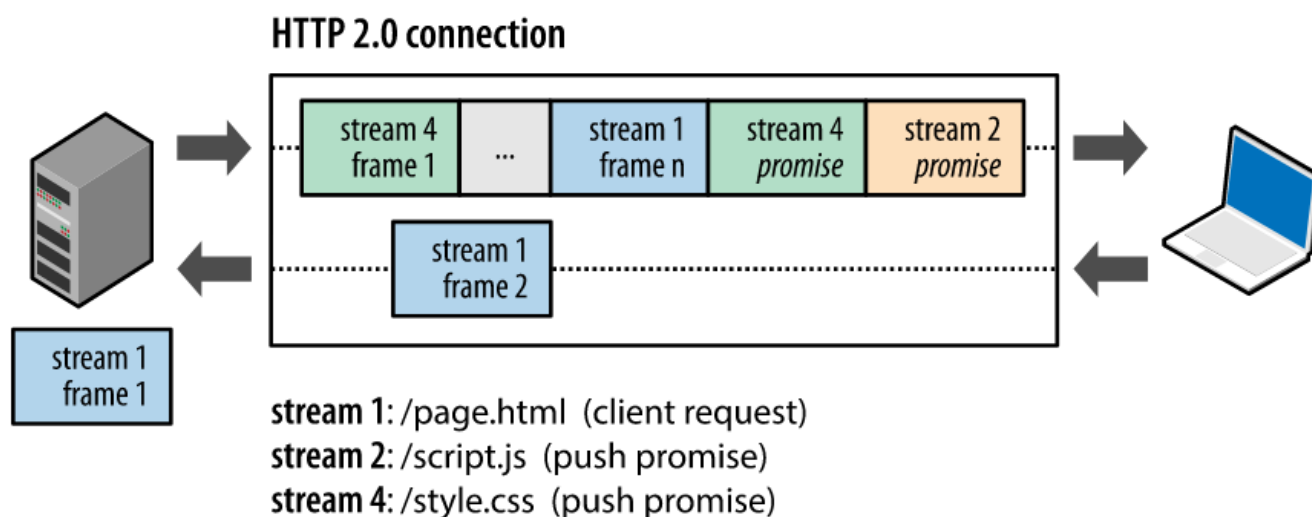
Modo de bloqueio

Obtém a seguinte url: <https://www.google.com> e aguarda até que o cliente receba a resposta completa. Depois de chamar o método **GET**, o processo aguarda até que a resposta seja recebida ou o tempo limite seja atingido. Você pode acessar os dados da Resposta Bruta usando a propriedade Response do cliente HTTP/2. Aqui você pode acessar os Cabeçalhos Brutos, o código de status da resposta, o Charset e mais.

```
TsgcHTTP2Client *oClient = new TsgcHTTP2Client();
vResponse = oClient->Get("https://www.google.com");
if (oClient->Response->Status == 200)
    ShowMessage("Response from server: " + vResponse);
else
    ShowMessage("Response Code: " + IntToStr(oClient->Response->Status));
```

Requisições | HTTP/2 Server Push

Server Push é a capacidade do servidor de enviar múltiplas respostas para uma única requisição do cliente. Ou seja, além da resposta à requisição original, o servidor pode enviar via push recursos adicionais ao cliente, sem que o cliente tenha que solicitar cada um explicitamente.



Cada vez que o servidor envia uma mensagem PushPromise ao cliente, o evento OnHTTP2PushPromise é chamado. Quando o cliente recebe um PushPromise, significa que o servidor enviará esse recurso nos próximos pacotes, então o cliente pode aceitá-lo ou rejeitá-lo.

```
TsgcHTTP2Client *oClient = new TsgcHTTP2Client();
oClient->OnHTTP2PushPromise = OnHTTP2PushPromiseEvent;
oClient->Get("https://http2.golang.org/serverpush");
...
void OnHTTP2PushPromiseEvent(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection,
    const TsgcHTTP2_Frame_PushPromise *PushPromise, bool &Cancel)
{
    if (PushPromise->URL == "/serverpush/static/godocs.js")
    {
        Cancel = true;
    }
    else
    {
        Cancel = false;
    }
}
```

TsgcHTTP2Client | HTTP/2 Download File

Quando o cliente requisita um arquivo do servidor, utilize o evento `OnHTTP2Response` para carregar a resposta em stream.

Downloads de Arquivos Grandes

Ao baixar arquivos grandes (centenas de MB ou mais), defina `HTTP2Options.ReadTimeout` como `0` (sem timeout) para garantir que a transferência seja concluída sem ser interrompida pelo timeout padrão de 60 segundos:

```
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(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection,
    const TsgcHTTP2RequestProperty *Request, const TsgcHTTP2ResponseProperty *Response)
{
    TFileStream *oStream = new TFileStream("file", fmOpenWrite | fmCreate);
    try
    {
        oStream->CopyFrom(Response->Data, Response->Data->Size);
    }
    finally
    {
        oStream->Free();
    }
}
```

TsgcHTTP2Client | HTTP/2 Partial Responses

Normalmente, quando você envia uma Requisição HTTP, o servidor envia uma resposta com o arquivo solicitado. Às vezes, em vez de enviar uma única resposta, o servidor pode enviar múltiplas respostas como um stream. Nesses casos, você pode usar o evento **OnHTTP2ResponseFragment** para capturar essas respostas e exibi-las ao usuário.

Example: envia uma requisição para <https://http2.golang.org/clockstream> e o servidor enviará uma resposta em stream a cada segundo.

```
TsgcHTTP2Client *oClient = new TsgcHTTP2Client();
oClient->OnHTTP2ResponseFragment = OnHTTP2ResponseFragmentEvent;
oClient->Get("https://http2.golang.org/clockstream");
...
void OnHTTP2ResponseFragmentEvent(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection,
    const TsgcHTTP2RequestProperty *Request, const TsgcHTTP2ResponseFragmentProperty *Fragment)
{
    ShowMessage(Fragment->DataString);
}
```

TsgcHTTP2Client | Cabeçalhos HTTP/2

TsgcHTTP2Client permite personalizar os Headers enviados ao servidor quando o cliente conecta

Exemplo: se você precisar adicionar este Cabeçalho HTTP "Client: sgcWebSockets"

```
void OnHTTP2BeforeRequest(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection,  
    ref TStringList *Headers)  
{  
    Headers->Add("Client: sgcWebSockets");  
}
```

Você pode utilizar Request.CustomHeaders para adicionar seus cabeçalhos personalizados também.

TsgcHTTP2Client | Client Close Connection

A conexão pode ser fechada utilizando a propriedade Active ou utilizando os métodos Close/Disconnect.

Propriedade Active

Quando a conexão está ativa e você define Active := False, a conexão será fechada imediatamente sem enviar nenhuma mensagem ao servidor sobre a desconexão.

Disconnect

Você pode usar o método Disconnect (de TsgcHTTP2Client ou TsgcHTTP2ConnectionClient) para desconectar o socket.

Close

Este método envia uma mensagem ao servidor informando que a conexão será fechada e você pode enviar opcionalmente alguma informação sobre o motivo da desconexão. É uma maneira limpa de fechar uma conexão HTTP/2.

O método Close pode ser chamado a partir dos objetos TsgcHTTP2Client ou TsgcHTTP2ConnectionClient.

Os seguintes motivos de erro podem ser enviados:

- no error
- erro de protocolo
- internal error
- flow control error
- settings timeout
- stream closed
- erro de tamanho de frame
- refused stream
- cancel
- erro de compressão
- connect error
- acalme-se
- segurança inadequada
- required

TsgcHTTP2Client | Client Keep Connection Active

Depois que seu cliente se conectou ao servidor, às vezes a conexão pode ser fechada devido a sinal fraco, erros de conexão... há 2 propriedades que ajudam a manter a conexão ativa.

HeartBeat

A propriedade **HeartBeat** permite que você **envie um Ping** a cada **X segundos** para **manter a conexão ativa**. Alguns servidores fecham conexões TCP se não houver dados trocados entre os peers. O HeartBeat resolve esse problema, enviando um ping a cada intervalo específico. Geralmente isso é suficiente para manter uma conexão ativa.

A propriedade HeartBeatType permite personalizar como o HeartBeat funciona:

1. **hbtAlways**: envia um ping a cada x segundos definidos em Interval.
2. **hbtOnlyIfNoMsgRcvInterval**: envia um ping a cada x segundos somente se nenhuma mensagem tiver sido recebida durante os últimos x segundos definidos na propriedade Interval.

Exemplo: envia um ping a cada 30 segundos

```
oClient = new TsgcHTTP2Client();
oClient->HeartBeat->Interval = 30;
oClient->HeartBeat->Enabled = true;
oClient->Active = true;
```

WatchDog

Se o WatchDog estiver habilitado, quando o cliente detecta uma desconexão, o WatchDog tenta reconectar novamente a cada X segundos até que a conexão esteja ativa de novo.

Exemplo: reconecta a cada 10 segundos após uma desconexão com tentativas ilimitadas.

```
oClient = new TsgcHTTP2Client();
oClient->WatchDog->Interval = 10;
oClient->WatchDog->Attempts = 0;
oClient->WatchDog->Enabled = true;
oClient->Active = true;
```

TsgcHTTP2Client | HTTP/2 Reason Disconnection

O servidor HTTP/2 pode desconectar um cliente por vários motivos. Quando o servidor deseja informar ao cliente o motivo pelo qual está desconectando, ele envia uma mensagem **GoAway** ao cliente com informações sobre a desconexão.

Use o evento `OnHTTP2GoAway` para capturar o motivo pelo qual o servidor desconectou (se o cliente quiser fechar uma conexão, pode usar o método `close` para enviar o motivo pelo qual está fechando a conexão).

O objeto **TsgcHTTP2GoAwayProperty** contém as informações sobre a desconexão

- **LastStreamId**: é o último stream processado pelo servidor.
- **ErrorCode**: inteiro que identifica o código de erro.
- **ErrorDescription**: descrição do erro, uma das seguintes:
 - no error
 - erro de protocolo
 - internal error
 - flow control error
 - settings timeout
 - stream closed
 - erro de tamanho de frame
 - refused stream
 - cancel
 - erro de compressão
 - connect error
 - acalme-se
 - segurança inadequada
 - required
- **AdditionalDebugData**: string opcional que oferece mais informações sobre a desconexão.

TsgcHTTP2Client | Requisições pendentes do cliente

Quando o cliente envia várias requisições, estas são processadas em uma thread secundária; às vezes a conexão pode ser fechada por qualquer motivo e ainda há requisições pendentes. Use o evento `OnHTTP2PendingRequests` para tratar essas requisições pendentes. Este evento é chamado quando o cliente detecta uma desconexão e ainda há requisições pendentes. Este evento tem 2 parâmetros:

1. **Reconnect:** desabilitado por padrão, se você definir como `true`, o cliente reconectará automaticamente.
2. **Clear:** habilitado por padrão; se você definir como `false`, quando o cliente se conectar novamente, ele tentará reenviar as requisições pendentes ao servidor.

TsgcHTTP2Client | Autenticação de Cliente

O cliente HTTP/2 suporta 2 tipos de autenticação: Basic Authentication e OAuth2 Authentication.

Utilize o evento **OnHTTP2Authorization** para tratar ambos os tipos de autenticação.

Basic Authentication

Se o servidor retornar um cabeçalho solicitando Basic Authentication, defina em OnHTTP2Authorization o nome de usuário e a senha.

```
TsgcHTTP2Client *oClient = new TsgcHTTP2Client();
oClient->OnHTTP2Authorization = OnHTTP2AuthorizationEvent;
...
void OnHTTP2AuthorizationEvent(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection, const string AuthType)
{
    if (AuthType == "Basic")
    {
        UserName = "user";
        Password = "secret";
    }
}
```

Bearer Token

Se o servidor retornar um cabeçalho requisitando autenticação Bearer Token, defina o token em OnHTTP2Authorization.

```
TsgcHTTP2Client *oClient = new TsgcHTTP2Client();
oClient->OnHTTP2Authorization = OnHTTP2AuthorizationEvent;
...
void OnHTTP2AuthorizationEvent(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection, const string AuthType)
{
    if (AuthType == "Bearer")
    {
        aToken = "bearer token";
    }
}
```

Valor Bearer de Third-party

Se você já conhece o valor do Bearer, porque o obteve usando outro método, você pode passar o valor do Bearer como um cabeçalho HTTP usando as seguintes propriedades da requisição, basta defini-las antes de chamar qualquer método de requisição HTTP:

```
TsgcHTTP2Client.Request.BearerAuthentication
= true

TsgcHTTP2Client.Request.BearerToken = "< value of the token >"
```

OAuth2

Leia o artigo a seguir se quiser usar nosso [componente OAuth2 com o cliente HTTP/2](#).

TsgcHTTP2Client | HTTP/2 and OAuth2

O OAuth2 é um método de autorização comum usado por várias empresas como o Google. Quando você deseja autenticar-se nos servidores do Google para usar qualquer uma de suas APIs, geralmente é necessária uma autenticação usando OAuth2.

O `sgcWebSockets` suporta OAuth2 no cliente HTTP/2, há uma propriedade chamada **Authentication.Token.OAuth** onde você deve atribuir uma instância de `TsgcHTTP_OAuth2`.

Como conectar à API Google do GMail

Para conectar às APIs do Google, precisaremos criar uma instância de `TsgcHTTP_OAuth2` e preencher os seguintes dados:

```
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';
```

Depois de preencher o componente cliente OAuth2, crie uma nova instância de `TsgcHTTP2Client` e atribua o componente OAuth2 ao cliente HTTP/2.

```
TsgcHTTP2Client1.Authentication.Token.OAuth := TsgcHTTP_OAuth1;
```

Por fim, faça uma requisição para obter uma lista de mensagens da conta `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 é um wrapper das conexões HTTP/2 do cliente. Você pode acessar este objeto nos eventos do Cliente.

Métodos

- **Ping:** envia um ping ao servidor para manter a conexão ativa.
- **Close:** envia uma mensagem ao servidor com informações sobre o motivo da desconexão.
- **Disconnect:** fecha a conexão sem enviar nenhuma mensagem informativa ao servidor.
- **HTTP/2 Methods:**

GET: O método GET solicita uma representação do recurso especificado. Requisições utilizando GET devem apenas recuperar dados.

HEAD: O método HEAD solicita uma resposta idêntica à de uma requisição GET, mas sem o corpo da resposta.

POST: O método POST é utilizado para submeter uma entidade ao recurso especificado, frequentemente causando uma mudança de estado ou efeitos colaterais no servidor.

PUT: O método PUT substitui todas as representações atuais do recurso alvo pelo payload da requisição.

DELETE: O método HEAD solicita uma resposta idêntica à de uma requisição GET, mas sem o corpo da resposta.

CONNECT: O método CONNECT estabelece um túnel para o servidor identificado pelo recurso alvo.

OPTIONS: O método OPTIONS é usado para descrever as opções de comunicação para o recurso de destino.

TRACE: O método TRACE realiza um teste de loop-back de mensagem ao longo do caminho até o recurso de destino.

PATCH: O método PATCH é utilizado para aplicar modificações parciais a um recurso.

TsgcHTTP2RequestProperty

Este objeto é recebido como argumento do evento OnHTTP2Response, ele permite conhecer a requisição original da resposta enviada pelo servidor.

Properties

- **Method:** identifica o método HTTP/2 (GET, POST...)
- **URL:** é a URL solicitada.
- **Request:** contém os campos da requisição.

TsgcHTTP2ResponseProperty

Este objeto é recebido como argumento do evento OnHTTP2Response, ele permite conhecer a resposta enviada pelo servidor ao cliente.

Properties

- **Headers:** contém uma lista de cabeçalhos brutos recebidos do servidor.
- **Data:** contém o corpo bruto enviado pelo servidor como resposta à requisição.
- **DataString:** é a conversão de Data para string.
- **DataUTF8:** é a conversão para string UTF8 de Data.
- **PushPromise:** se atribuído, contém o objeto PushPromise enviado pelo servidor ao cliente (significa que este objeto de resposta não foi solicitado pelo cliente).

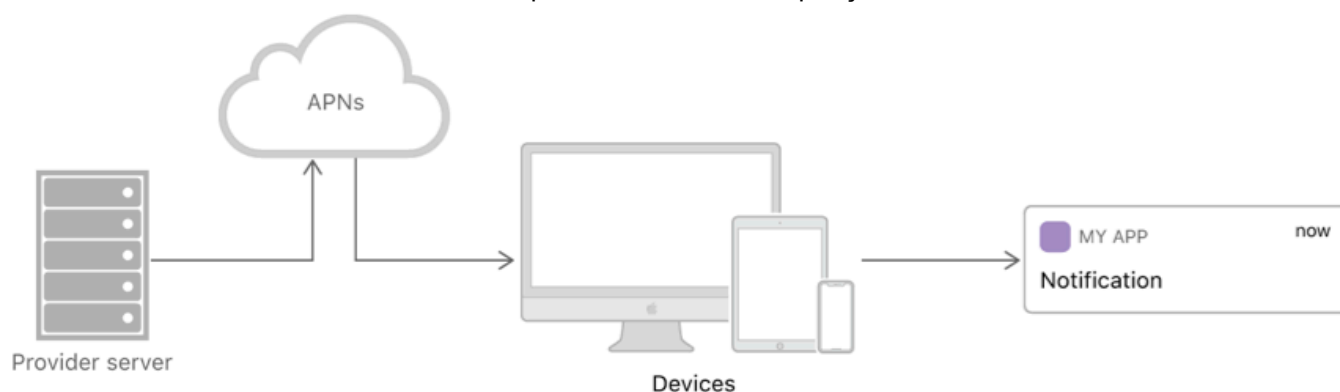
HTTP2 | Apple Push Notifications

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server

A Apple permite o envio de push notifications a dispositivos Apple utilizando o Apple Push Notification Service (APNs).

Quando você quiser enviar uma notificação a um dispositivo, o provedor deve enviar um HTTP/2 POST à APNs incluindo as seguintes informações:

- Payload JSON com as informações que você deseja enviar.
- Um Device Token que identifica o dispositivo do usuário.
- Alguns Cabeçalhos HTTP sobre como entregar a notificação.
- Um certificado SSL ou um token JWT para autenticar sua requisição contra o APNs



O que é necessário para enviar notificações

Para enviar notificações ao seu dispositivo usando o Rad Studio, você deve seguir os próximos passos

- Registre seu APP com o APNs
- [Gerar uma Remote Notification](#)
- [Enviando Requisições de Notificação para o APNs](#)
 - [Conexão Baseada em Token para APNs](#)
 - [Conexão Baseada em Certificado ao APNs](#)

APN | Generate a Remote Notification APNs

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/generating_a_remote_notification

As Notificações da Apple utilizam um payload JSON para enviar o objeto de notificação. O tamanho máximo do payload é de 4096 bytes.

JSON Payload Samples

Mensagem de alerta simples

```
{
  "aps": {
    "alert": "Alert from sgcWebSockets!"
  }
}
```

Alerta com título e subtítulo.

```
{
  "aps" : {
    "alert" : {
      "title" : "Game Request",
      "subtitle" : "Five Card Draw",
      "body" : "Bob wants to play poker",
    },
    "category" : "GAME_INVITATION"
  },
  "gameID" : "12345678"
}
```

APN | Sending Notification Requests to APNs

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server

Envie o payload da sua notificação remota e as informações do device token ao Apple Push Notification service (APNs).

Como se Conectar à APNs

Você deve utilizar o protocolo HTTP/2 e pelo menos TLS 1.2 ou posterior para estabelecer uma conexão bem-sucedida entre o seu Server Provider e um dos seguintes servidores:

Development Server: <https://api.sandbox.push.apple>

Servidor de produção: <https://api.push.apple>

Código de Exemplo

Crie uma nova instância de `TsgcHTTP2Client` e chame o método `POST` para enviar uma notificação ao APNs.

```
TsgcHTTP2Client *oHTTP = new TsgcHTTP2Client();
try
{
    // ... requires authorization code
    TStringStream *oStream = new TStringStream("{\"aps\":{\"alert\":\"Alert from sgcWebSockets!\"}}",
        TEncoding::UTF8);
    try
    {
        oHTTP->Post("https://api.push.apple/3/device/device_token", oStream);
        if (oHTTP->Response->Status == 200)
        {
            ShowMessage("Notification Sent Successfully");
        }
        else
        {
            ShowMessage("Notification error");
        }
    }
    finally
    {
        oHTTP->Free();
    }
}
finally
{
    oHTTP->Free();
}
```

Para enviar notificações, você deve estabelecer confiança com a APNs, seja **baseada em token** ou **baseada em certificado**, usando o protocolo HTTP/2 e TLS 1.2 ou posterior.

- [Token-Based Connection to APNs](#)
- [Certificate-Based Connection to APNs](#)

APNs Trusted | Token-Based Connection to APNs

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/establishing_a_token-based_connection_to_apns

Proteja suas comunicações com o Apple Push Notification service (APNs) usando Tokens de autenticação stateless.

Primeiro você deve obter uma **Encryption Key** e um **Key ID** da Apple Developer Account. Uma vez concluído um registro bem-sucedido, você obterá uma string de 10 caracteres com o Key ID e uma chave de assinatura de Authentication Token como um arquivo com extensão .p8.

Você deve usar o **Cliente JWT** do sgWebSockets para gerar um JWT usando **ES256** como algoritmo. O token não deve ser gerado para cada requisição HTTP/2, o token não deve ser atualizado antes de 20 minutos e não depois de 60 minutos.

Configurar Cliente JWT

Configure o Cliente JWT com os seguintes valores:

- **JWTOptions.Header.Algorithm:** é o algoritmo de criptografia que você utilizou para criptografar o token. O APNs suporta apenas o algoritmo ES256.
- **JWTOptions.Header.kid:** é o Key ID de 10 caracteres obtido da sua conta de desenvolvedor.
- **JWTOptions.Payload.iss:** o valor para o qual é o Team ID de 10 caracteres que você usa para desenvolver os aplicativos da sua empresa. Obtenha esse valor a partir da sua conta de desenvolvedor.
- **JWTOptions.Payload.iat:** O horário de emissão ("issued at"), cujo valor indica o momento em que este token JSON foi gerado. Especifique o valor como o número de segundos desde o Epoch, em UTC. O valor não deve ser superior a uma hora a partir do horário atual.
- **JWTOptions.RefreshTokenAfter:** define o valor em segundos para 40 minutos (60*40).

Utilizar conexões baseadas em Token requer o envio do **apns-topic** com o valor do bundle ID/app id do seu app (exemplo: com.example.application).

```
TsgcHTTP2Client *oHTTP = new TsgcHTTP2Client(NULL);
oHTTP->TLSOptions->IOHandler = iohOpenSSL;

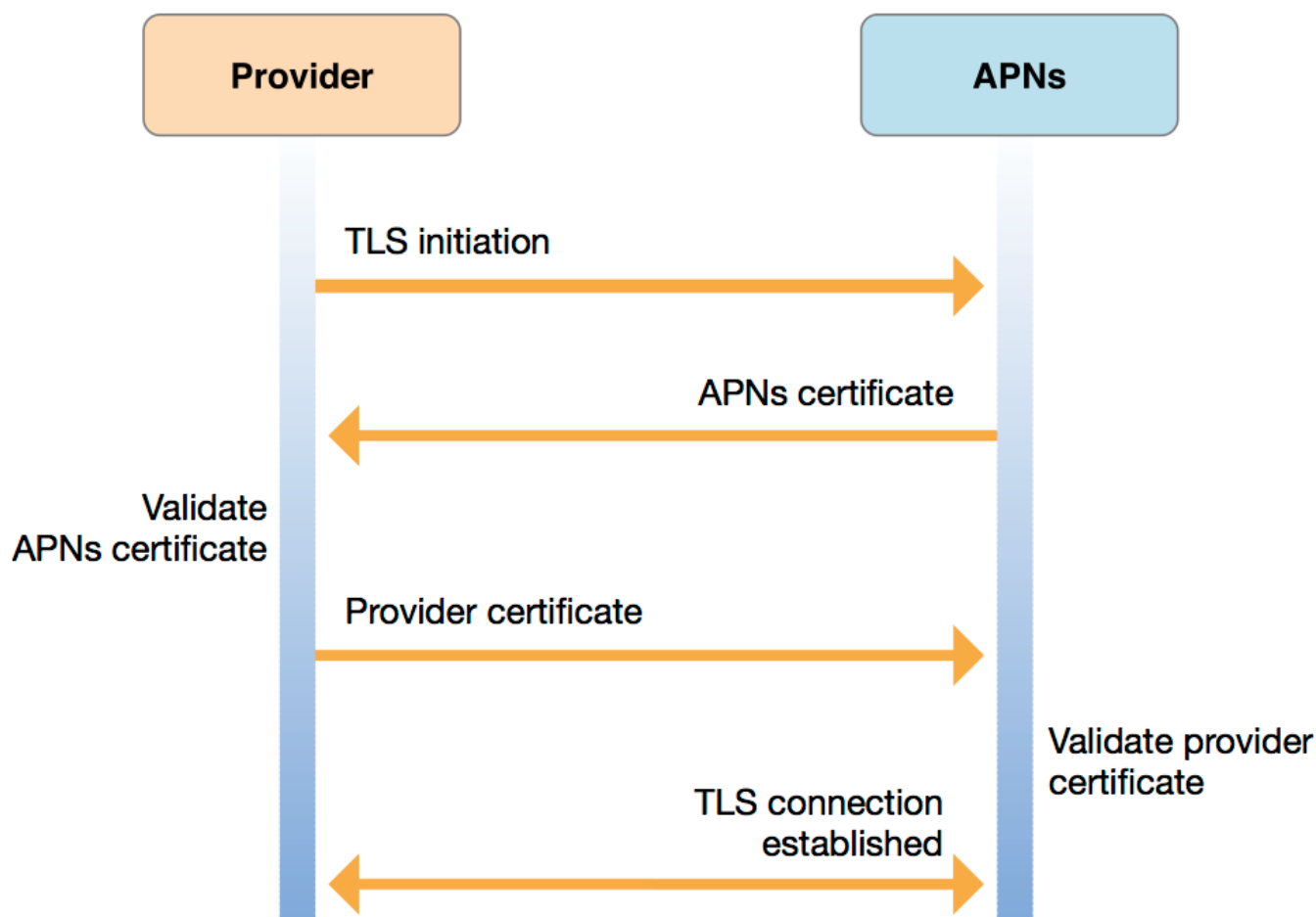
TsgcHTTP_JWT_Client *oJWT = new TsgcHTTP_JWT_Client(NULL);
oHTTP->Authentication->Token->JWT = oJWT;
oJWT->JWTOptions->Header->alg = jwtES256;
oJWT->JWTOptions->Header->kid = "apple key id";
oJWT->JWTOptions->Payload->iss = "issuer";
oJWT->JWTOptions->Payload->iat = StrToInt64(GetDateTimeUnix(Now, False));
oJWT->JWTOptions->Algorithms->ES->PrivateKey->LoadFromFile("AuthKey_*.p8");
oJWT->JWTOptions->RefreshTokenAfter = 60*40;

oHTTP->Request->CustomHeaders->Clear();
oHTTP->Request->CustomHeaders->Add("apns-topic: com.example.application");
```

Conexão Baseada em Certificado com APNs

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/establishing_a_certificate-based_connection_to_apns

Você pode proteger suas comunicações com o Apple Push Notification service (APNs) utilizando um certificado obtido da Apple.



Primeiro entre em sua conta de desenvolvedor e **crie um novo certificado** para o serviço Apple Push Notification

Depois de baixar o seu certificado, o cliente HTTP/2 do `sgcWebSockets` permite que você utilize 2 IOHandlers de segurança (apenas para Windows; para outras personalidades, somente o OpenSSL é suportado).

- OpenSSL
- SChannel (apenas para windows)

OpenSSL

Se você usar OpenSSL, deve implantar as bibliotecas OpenSSL com sua aplicação. Antes de definir o certificado com o `TsgcHTTP2Client`, este certificado deve primeiro ser convertido para o formato PEM porque o OpenSSL não permite importar certificados P12 diretamente.

Utilize os comandos a seguir para converter um único certificado P12 em um certificado no formato PEM e um arquivo de chave privada

criar arquivo de certificado PEM

```
openssl pkcs12 -in INFILE.p12 -out OUTFILE.crt -nokeys
```

Criar arquivo de Chave Privada

```
openssl pkcs12 -in INFILE.p12 -out OUTFILE.key -nodes -nocerts
```

Uma vez que você tenha seu certificado e chave privada no formato PEM, você pode configurar o [TsgcHTTP2Client](#) da seguinte forma.

```
TsgcHTTP2Client *oHTTP = new TsgcHTTP2Client(NULL);
oHTTP->TLSOptions->IOHandler = iohOpenSSL;
oHTTP->TLSOptions->CertFile = "certificate_file.pem";
oHTTP->TLSOptions->KeyFile = "private_key.pem";
oHTTP->TLSOptions->Password = "certificate password";
oHTTP->TLSOptions->Version = tls1_2;
```

SChannel

Se você usar SChannel, não há necessidade de implantar nenhuma biblioteca e o certificado baixado da Apple pode ser importado diretamente sem a necessidade de uma conversão prévia para o formato PEM. Defina a propriedade `UseLegacyCredentials` como `true` ao usar SChannel como IOHandler.

```
TsgcHTTP2Client *oHTTP = new TsgcHTTP2Client(NULL);
oHTTP->TLSOptions->IOHandler = iohSChannel;
oHTTP->TLSOptions->SChannel_Options->UseLegacyCredentials = true;
oHTTP->TLSOptions->CertFile = "certificate_file.p12";
oHTTP->TLSOptions->Password = "certificate password";
oHTTP->TLSOptions->Version = tls1_2;
```

Erros

Se você receber o erro **"missing topic"**, provavelmente está usando um certificado universal (certificados que podem ser usados para push notifications, voip...) que requer definir o nome do tópico com o valor do bundle ID/app id do seu app (exemplo: `com.example.application`). Basta definir o cabeçalho `apns-topic` com o valor correto na propriedade `Request` do cliente HTTP/2.

```
oHTTP->Request->CustomHeaders->Clear();
oHTTP->Request->CustomHeaders->Add("apns-topic: com.example.application");
```

HTTP/1

TsgcHTTP1Client é um componente não visual que herda do componente indy **TIdHTTP** e adiciona algumas novas propriedades. Este componente está localizado na unit **sgcHTTP**.

TLSOptions

Permite que você configure como conectar a servidores SSL/TLS seguros utilizando o protocolo HTTP/1.

ALPNProtocols: lista dos protocolos ALPN que serão enviados ao servidor.

RootCertFile: caminho para o arquivo de certificado raiz.

CertFile: caminho para o arquivo de certificado.

KeyFile: caminho para o arquivo de chave do certificado.

Password: se o certificado estiver protegido por senha, defina-a aqui.

VerifyCertificate: se o certificado deve ser verificado, habilite esta propriedade. Utilize o evento **OnSSLVerifyPeer** para personalizar a verificação SSL.

VerifyDepth: é uma propriedade Integer que representa o número máximo de links permitidos quando a verificação é realizada para o certificado X.509.

Version: por padrão utiliza TLS 1.0. Se o servidor exigir uma versão TLS superior, ela pode ser selecionada aqui.

Proxy: aqui você pode definir se deseja conectar através de um servidor proxy; você pode conectar aos seguintes servidores proxy:

pxyHTTP: Servidor HTTP Proxy.

pxySocks4: SOCKS4 Proxy Server.

pxySocks4A: Servidor Proxy SOCKS4A.

pxySocks5: Servidor SOCKS5 Proxy.

IOHandler: selecione qual biblioteca você utilizará para conectar usando TLS.

iohOpenSSL: usa a biblioteca OpenSSL e é o padrão para os componentes Indy. Requer a implantação das bibliotecas OpenSSL para win32/win64.

iohSChannel: utiliza o Secure Channel, que é um protocolo de segurança implementado pela Microsoft para Windows. Não requer a implantação das bibliotecas OpenSSL. Funciona somente no Windows 32/64 bits.

OpenSSL_Options: configuração das bibliotecas openssl.

APIVersion: permite definir qual API OpenSSL será usada.

oslAPI_1_0: usa a API 1.0 do OpenSSL, é a mais recente suportada pelo Indy

oslAPI_1_1: usa a API 1.1 do OpenSSL, requer nossa biblioteca Indy personalizada e permite usar as bibliotecas OpenSSL 1.1.1 (com suporte a TLS 1.3).

oslAPI_3_0: utiliza a API 3.0 do OpenSSL, requer nossa biblioteca Indy personalizada e permite utilizar as bibliotecas OpenSSL 3.0.0 (com suporte a TLS 1.3).

LibPath: aqui você pode configurar onde as bibliotecas OpenSSL estão localizadas

oslpNone: este é o padrão. As bibliotecas OpenSSL devem estar na mesma pasta do binário ou em um caminho conhecido.

oslpDefaultFolder: define automaticamente o caminho do OpenSSL onde as bibliotecas devem estar localizadas para todas as personalidades da IDE.

oslpCustomFolder: se esta for a opção selecionada, defina o caminho completo na propriedade **LibPathCustom**.

LibPathCustom: quando **LibPath** = **oslpCustomFolder**, defina aqui o caminho completo onde as bibliotecas OpenSSL estão localizadas.

UnixSymLinks: habilita ou desabilita o carregamento de SymLinks em sistemas Unix (por padrão está habilitado, exceto no OSX64):

oslsSymLinksDefault: por padrão, os symlinks são habilitados, exceto em OSX64 (o macOS Monterey e posteriores falham ao tentar carregar a biblioteca sem uma versão).

oslsSymLinksLoadFirst: carrega os symlinks primeiro, antes de tentar carregar as bibliotecas versionadas.

oslsSymLinksLoad: carrega os symlinks após tentar carregar as bibliotecas versionadas.

oslsSymLinksDontLoad: não carrega os SymLinks.

MinVersion: defina aqui a versão mínima que o cliente utilizará para se conectar a um servidor seguro. Por padrão, o valor é **tlsUndefined**, o que significa que a versão mínima é a mesma definida na propriedade **Version**. Exemplo: se você quiser que o cliente conecte apenas utilizando TLS 1.2 ou TLS 1.3, defina os seguintes valores.

SSLOptions.Version := tls1_3;
SSLOptions.OpenSSL_Options.MinVersion := tls1_2;
X509Checks: utilize esta propriedade para habilitar validações adicionais de certificado X509:
Mode: selecione quais opções serão validadas
oslx509chHostName: verifica o hostname do certificado.
oslx509chIPAddress: verifica o endereço ip do certificado.
HostName: define o hostname se ele for diferente do da requisição.
IPAddress: defina o endereço ip se for diferente do da requisição.

SChannel_Options: permite que você utilize um certificado do Windows Certificate Store.
CertHash: é o Hash do certificado. Você pode encontrar o Hash do certificado executando um comando dir no powershell.
CipherList: aqui você pode definir quais Ciphers serão usados (separados por ":"). Exemplo: CALG_AES_256:CALG_AES_128
CertStoreName: o nome do store onde o certificado está armazenado. Selecione um dos seguintes:
 scsnMY (o padrão)
 scsnCA
 scsnRoot
 scsnTrust
CertStorePath: o caminho de armazenamento onde o certificado está armazenado. Selecione um dos seguintes:
 scspStoreCurrentUser (o padrão)
 scspStoreLocalMachine

Log

Se a propriedade Log estiver habilitada, salva as mensagens do socket em um arquivo de log especificado, útil para depuração.

LogOptions.FileName: caminho completo para o nome do arquivo.

Autenticação

Permite que você se autentique utilizando [OAuth2](#) ou [JWT](#).

Requisições Assíncronas

Por padrão, o HTTP1Client utiliza requisições bloqueantes, portanto, após chamar um método de requisição HTTP, o cliente aguarda a resposta do servidor. Alternativamente, você pode utilizar métodos assíncronos para executar essas requisições HTTP em uma thread secundária, evitando bloquear a thread onde a requisição é chamada. Os seguintes métodos assíncronos estão implementados:

- GetAsync
- PostAsync
- PutAsync
- OptionsAsync
- DeleteAsync

Após chamar estes métodos, em vez de aguardar a resposta, o código continua para a próxima linha, e a resposta pode ser tratada utilizando o evento **OnAsyncResponse**.

```
void __fastcall OnAsyncResultEvent(TObject* Sender, const TsgcHTTPAsyncRequest* aRequest, const TIdHTTPResponse*
```

Se houver qualquer erro ao processar a requisição assíncrona, a exceção será gerada no evento **OnAsyncException**.

Exemplos

Solicita um método **GET** a um servidor HTTPs usando **TLS 1.2**

```
TsgcHTTP1Client *oHTTP = new TsgcHTTP1Client();
try
{
    oHTTP->TLSOptions->Version = tls1_2;
    ShowMessage(oHTTP->Get("https://www.google.es"));
}
__finally
{
    oHTTP->Free();
}
```

Requisita um método **GET** a um servidor HTTPs usando **openssl 1.1** e **TLS 1.3**

```
TsgcHTTP1Client *oHTTP = new TsgcHTTP1Client();
try
{
    oHTTP->TLSOptions->OpenSSL_Options->APIVersion = sslAPI_1_1;
    oHTTP->TLSOptions->Version = tls1_3;
    ShowMessage(oHTTP->Get("https://www.google.es"));
}
__finally
{
    oHTTP->Free();
}
```

Solicite um método **POST** assíncrono e leia a resposta utilizando o **OnAsyncResultEvent**.

```
void __fastcall OnAsyncExceptionEvent(TObject* Sender, const TsgcThread* aThread, const Exception* E)
{
    Log(E->Message);
}
void __fastcall OnAsyncResultEvent(TObject* Sender, const TsgcHTTPAsyncRequest* aRequest, const TIdHTTPResponse*
{
    if (aResponse->ResponseCode == 200)
        Log("ok", aRequest->Response);
    else
        Log("error", aRequest->Response);
}
TsgcHTTP1Client* oHTTP = new TsgcHTTP1Client(NULL);
oHTTP->OnAsyncResult = OnAsyncResultEvent;
oHTTP->OnAsyncException = OnAsyncExceptionEvent;
TStringStream* oRequest = new TStringStream("body");
TStringStream* oResponse = new TStringStream("");
oHTTP->PostAsync("https://localhost/test", oRequest, oResponse);
```

Faça uma requisição com o método **GET** a um servidor HTTPs usando **SChannel** para **Windows**.

```
TsgcHTTP1Client *oHTTP = new TsgcHTTP1Client();
try
{
    oHTTP->TLSOptions->IOHandler = iohSChannel;
    oHTTP->TLSOptions->Version = tls1_2;
    ShowMessage(oHTTP->Get("https://www.google.es"));
}
__finally
{
    oHTTP->Free();
}
```

Solicite o método **SSE** para obter eventos de dados

```
TsgcHTTP1Client *oHTTP = new TsgcHTTP1Client();
oHTTP->OnSSEMessage = OnSSEMessageEvent;
oHTTP->GetSSE("https://www.yoursite.com/sse");

void OnSSEMessageEvent(TObject *Sender, const string aMessage, bool &Cancel)
{
}
```

```
ShowMessage(aMessage);  
}
```

Eventos

OnSSEMessage

O evento é chamado quando uma nova mensagem SSE é recebida.

OnSSLVerifyPeer

Se a verificação de certificado estiver habilitada, neste evento você pode verificar e decidir se aceita o certificado do servidor.

OnSSLGetHandler

Este evento é gerado antes da criação do handler SSL. Você pode criar seu próprio handler SSL aqui (ele precisa ser herdado de `TIdServerIOHandlerSSLBase` ou `TIdIOHandlerSSLBase`) e definir as propriedades necessárias.

OnSSLAfterCreateHandler

Se nenhum objeto SSL personalizado foi criado, um padrão é criado usando o handler OpenSSL. Você pode acessar as propriedades do handler SSL e modificá-las se necessário.

OnAsyncResult

O evento é chamado após solicitar um método assíncrono (utilizando os métodos `GetAsync`, `PutAsync`...). Utilize o parâmetro `Response` para saber o resultado da requisição.

OnAsyncException

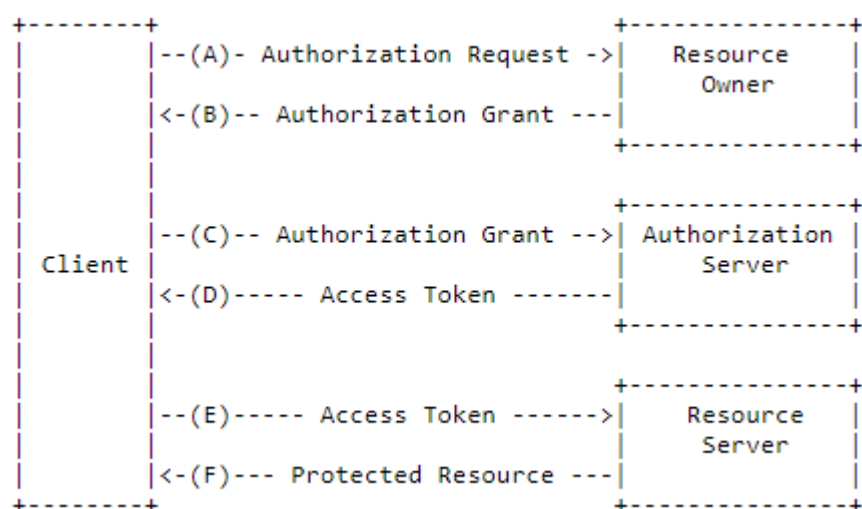
Se houver algum erro ao processar uma requisição assíncrona, este evento é chamado com a exceção gerada.

HTTP | OAuth2

O OAuth2 permite que aplicações de terceiros recebam acesso limitado a um serviço HTTP, seja em nome de um resource owner ou permitindo que uma aplicação de terceiros obtenha acesso em seu próprio nome. Graças ao OAuth2, provedores de serviço e aplicações consumidoras podem interagir entre si de forma segura.

No OAuth2, há 4 papéis:

- **Resource Owner:** o usuário.
- **Resource Server:** o servidor que hospeda os recursos protegidos e fornece acesso a eles com base no access token.
- **Client:** a aplicação externa que solicita a permissão.
- **Authorization Server:** emite o token de acesso após ter autenticado o usuário.



Componentes

- **TsgcHTTP_OAuth2_Client:** é um cliente com suporte para OAuth2, portanto pode conectar a servidores OAuth2 para solicitar uma autenticação como Google, Facebook...
- **TsgcHTTP_OAuth2_Server:** é a implementação de servidor do protocolo OAuth2, permite que você proteja os recursos do Servidor.
- **TsgcHTTP_OAuth2_Server_Provider:** permite implementar Provedores OAuth2 externos (como Azure AD, Google, Facebook...) no seu Servidor, para que o usuário possa fazer login usando as credenciais de usuário do Azure, Google, Facebook...

Os componentes OAuth2 de **Servidor** e **Cliente** suportam **PKCE** (Proof Key for Code Exchange), que é uma extensão do fluxo Authorization Code para prevenir ataques de CSRF e de injeção de código de autorização (RFC 7636).

TsgcHTTP_OAuth2_Client

Quando um cliente precisa de um novo Access Token, inicia automaticamente um servidor HTTP para processar a resposta do servidor de Autorização.

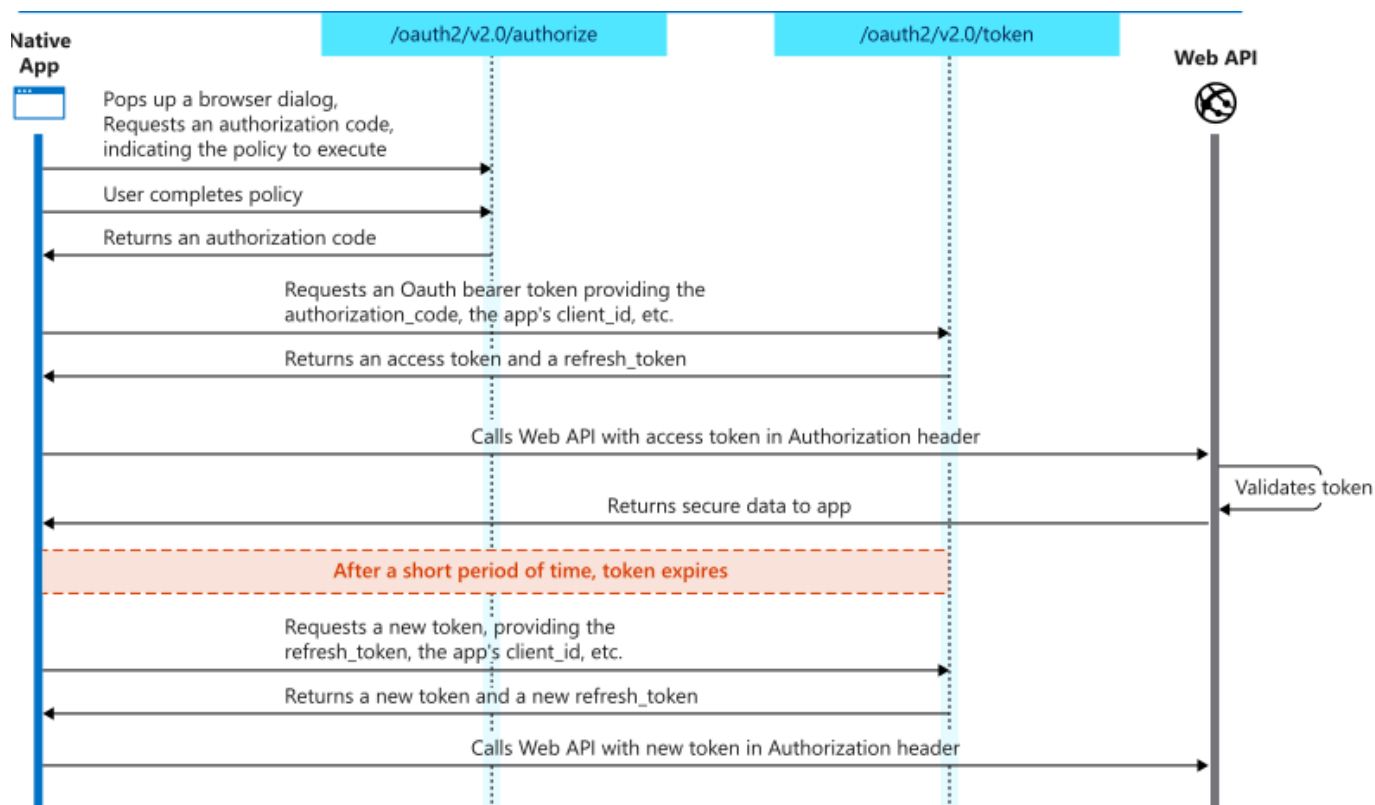
Introdução

Este componente permite que você gerencie o fluxo entre o cliente e os outros papéis. Basicamente, quando você define `Active := True`, ele abre um novo navegador web e solicita ao usuário a concessão de autorização; se bem-sucedido, o servidor de autorização envia um token à aplicação, que é processado e, com este token, o cliente pode se conectar ao servidor de recursos. Este componente inicia um servidor HTTP simples que trata as respostas do servidor de autorização e usa um cliente HTTP para solicitar Access Tokens.

GrantType

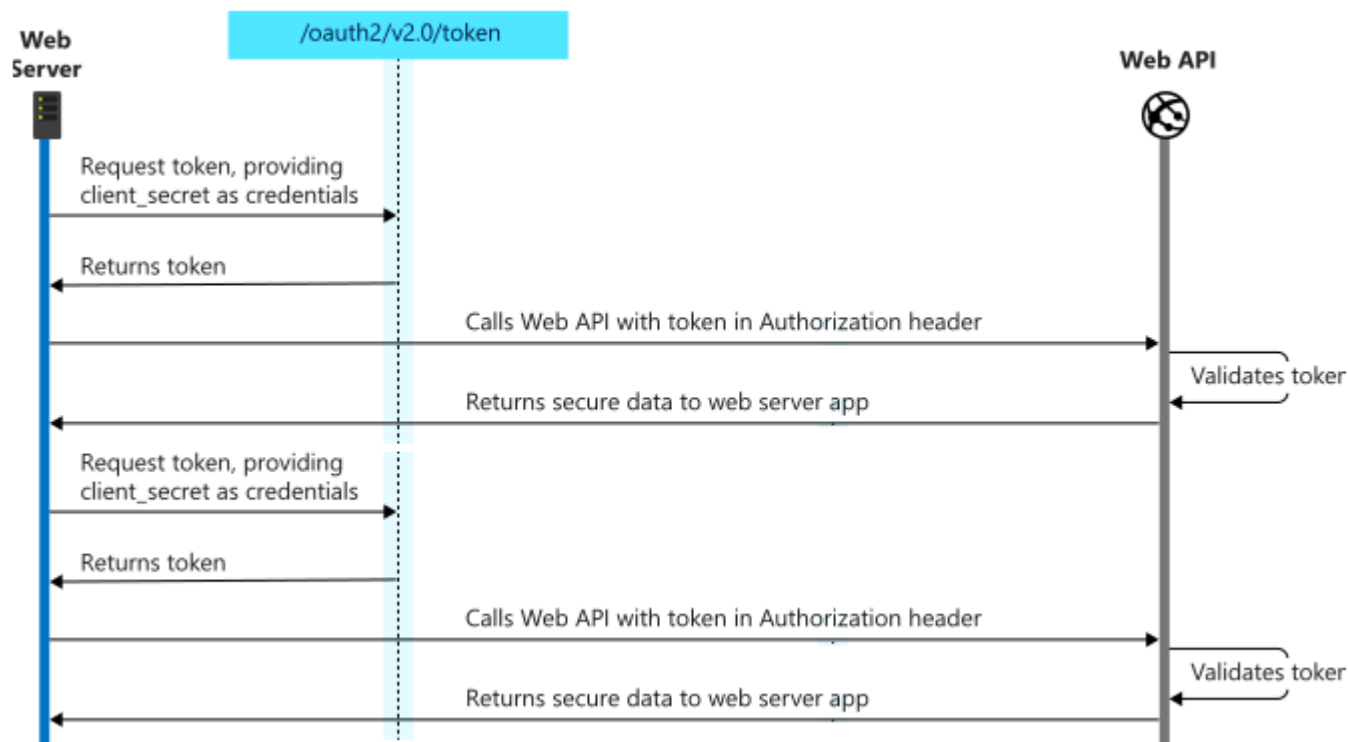
O cliente suporta os seguintes tipos de autorização:

auth2Code: É utilizado para realizar autenticação e autorização na maioria dos tipos de aplicações, incluindo aplicações de página única, aplicações web e aplicações instaladas nativamente. O fluxo permite que as aplicações adquiram com segurança `access_tokens` que podem ser utilizados para acessar recursos protegidos, bem como `refresh tokens` para obter `access_tokens` adicionais, e `ID tokens` para o usuário autenticado.



auth2CodePKCE: é o mesmo fluxo de autenticação que `auth2Code` com PKCE habilitado. PKCE (Proof Key for Code Exchange) é uma extensão de segurança para OAuth 2.0, projetada para aprimorar a segurança de fluxos de autorização para aplicações nativas e single-page. Ele mitiga o risco de ataques de interceptação, especialmente em clientes públicos onde o `authorization code` pode ser exposto a interceptação em trânsito. Geralmente esta opção é utilizada em apps nativos e mobile.

auth2ClientCredentials: Este tipo de grant é comumente utilizado para interações servidor-a-servidor que devem ser executadas em segundo plano, sem interação imediata com um usuário. Esses tipos de aplicações são frequentemente chamados de daemons ou contas de serviço.



auth2DeviceCode: Device Authorization Grant (RFC 8628) para dispositivos com entrada limitada, como smart TVs, consoles de mídia e dispositivos IoT que não possuem navegador ou têm capacidades de entrada limitadas. O dispositivo exibe um código de usuário e uma URI de verificação; o usuário visita a URI em um dispositivo secundário (celular ou computador) e insere o código para autorizar.

LocalServerOptions

Quando um cliente precisa de um novo Access Token, ele inicia automaticamente um servidor HTTP para processar a resposta do servidor de autorização. Este servidor é transparente para o usuário e geralmente funciona em localhost. Por padrão utiliza a porta 8080, mas você pode alterar se necessário.

- **IP:** IP no qual o servidor escuta, exemplo: 127.0.0.1
- **Port:** por padrão 8080. Ao usar `GrantType = auth2CodePKCE` (para aplicações desktop e nativas), você pode definir o valor da porta como zero e o servidor escolherá uma porta aleatória.
- **RedirectURL:** (opcional) permite personalizar a redirect URL, exemplo: `http://localhost:8080/oauth/`.
- **SSL:** habilite esta propriedade se o servidor local rodar em uma porta segura (*suportado somente pelas edições Professional e Enterprise).
- **SSLOptions:** permite personalizar as propriedades SSL do servidor (*suportado somente pelas Edições Professional e Enterprise).
- **LogOptions:** permite salvar o log das requisições/respostas recebidas e enviadas pelo HTTP Internal Server (*Apenas para as edições Professional e Enterprise).
 - **Enabled:** defina como True para habilitar o log em arquivo.
 - **FileName:** defina o nome do arquivo para armazenar o arquivo de log.

AuthorizationServerOptions

Aqui você deve definir a URL para Authorization e Access Token, normalmente fornecidas na especificação da API. Scope é uma lista de todos os escopos solicitados pelo cliente. Exemplo:

- **AuthURL:** `https://accounts.google.com/o/oauth2/auth`
- **TokenURL:** `https://accounts.google.com/o/oauth2/token`
- **Scope:** `https://mail.google.com/`
- **RevocationURL:** URL para o endpoint de revogação de token (obrigatório para o método Revoke). Exemplo: `https://accounts.google.com/o/oauth2/revoke`
- **IntrospectionURL:** URL do endpoint de introspecção de token (obrigatório para o método Introspect). Exemplo: `https://accounts.google.com/o/oauth2/introspect`

OAuth2Options

ClientId é um campo obrigatório que informa ao servidor qual é a identificação do cliente. Verifique a especificação da sua API para saber como obter um ClientId. O mesmo se aplica ao client secret.

Às vezes, o servidor exige um usuário e uma senha para conectar usando Basic Authentication; se for o caso, você pode configurar isso nos campos Username/Password. Exemplo:

- **GrantType:** Tipo de fluxo de autorização
 - **auth2Code:** aplicações confiáveis, como um web-server.
 - **auth2CodePKCE:** aplicações nativas ou móveis não confiáveis.
 - **auth2ClientCredentials:** aplicações automatizadas sem interação do usuário.
 - **auth2ResourceOwnerPassword:** permite que uma aplicação autentique o usuário tratando diretamente sua senha
 - **auth2DeviceCode:** Device Authorization Grant (RFC 8628) para dispositivos com restrições de entrada, sem navegador ou com capacidades de entrada limitadas.
- **ClientId:** 180803918307-eqjtm20gqfhcs6gjk1bbrreng022mqqc.apps.googleusercontent.com
- **ClientSecret:** _by1iYYrvVHxC2Z8TbtNEYJN
- **Username:**
- **Password:**

HTTPClientOptions

Aqui você pode personalizar as Opções do Cliente quando ele se conecta ao Servidor HTTP para solicitar um novo token.

TLSOptions: se o TLS estiver habilitado, aqui você pode personalizar algumas propriedades TLS.

ALPNProtocols: lista dos protocolos ALPN que serão enviados ao servidor.

RootCertFile: caminho para o arquivo de certificado raiz.

CertFile: caminho para o arquivo de certificado.

KeyFile: caminho para o arquivo de chave do certificado.

Password: se o certificado está protegido com uma senha, defina-a aqui.

VerifyCertificate: se o certificado deve ser verificado, habilite esta propriedade.

VerifyDepth: é uma propriedade Integer que representa o número máximo de links permitidos quando a verificação é realizada para o certificado X.509.

Version: por padrão usa TLS 1.0; se o servidor exigir uma versão TLS superior, ela pode ser selecionada aqui.

IOHandler: selecione qual biblioteca você utilizará para conectar usando TLS.

iohOpenSSL: utiliza a biblioteca OpenSSL e é o padrão para componentes Indy. Requer implantar as bibliotecas openssl para win32/win64.

iohSChannel: utiliza o Secure Channel, que é um protocolo de segurança implementado pela Microsoft para Windows, não requer a implantação de bibliotecas openssl. Funciona apenas no Windows 32/64 bits.

OpenSSL_Options: permite definir qual API OpenSSL será utilizada.

APIVersion: permite definir qual API OpenSSL será usada.

oslAPI_1_0: usa a API 1.0 do OpenSSL, é a mais recente suportada pelo Indy

oslAPI_1_1: usa a API 1.1 do OpenSSL, requer nossa biblioteca Indy personalizada e permite usar as bibliotecas OpenSSL 1.1.1 (com suporte a TLS 1.3).

oslAPI_3_0: utiliza a API 3.0 do OpenSSL, requer nossa biblioteca Indy personalizada e permite utilizar as bibliotecas OpenSSL 3.0.0 (com suporte a TLS 1.3).

LibPath: aqui você pode configurar onde estão localizadas as bibliotecas openssl

oslpNone: este é o padrão, as bibliotecas openssl devem estar na mesma pasta onde está o binário ou em um caminho conhecido.

oslpDefaultFolder: define automaticamente o caminho openssl onde as bibliotecas devem estar localizadas para todas as personalidades do IDE.

oslpCustomFolder: se esta for a opção selecionada, defina o caminho completo na propriedade LibPathCustom.

LibPathCustom: quando LibPath = oslpCustomFolder, defina aqui o caminho completo onde estão localizadas as bibliotecas openssl.

UnixSymLinks: habilita ou desabilita o carregamento de SymLinks em sistemas Unix (por padrão está habilitado, exceto no OSX64):

oslsSymLinksDefault: por padrão estão habilitados, exceto sob OSX64 (após o MacOS Monterey falha ao tentar carregar a biblioteca sem versão.).

oslsSymLinksLoadFirst: Carrega os SymLinks e faz isso antes de tentar carregar as bibliotecas de versão.

oslsSymLinksLoad: Carrega os SymLinks após tentar carregar as bibliotecas de versão.

oslsSymLinksDontLoad: não carrega os SymLinks.

SChannel_Options: permite que você utilize um certificado do Windows Certificate Store.

CertHash: é o Hash do certificado. Você pode encontrar o Hash do certificado executando um comando dir no powershell.

CertStoreName: o nome do store onde o certificado está armazenado. Selecione um dos abaixo:

scsnMY (o padrão)

scsnCA

scsnRoot

scsnTrust

CertStorePath: o caminho do store onde o certificado está armazenado. Selecione um dos abaixo:

scspStoreCurrentUser (o padrão)

scspStoreLocalMachine

LogOptions: se um nome de arquivo for definido, salvará um log das requisições/respostas HTTP do cliente HTTP

Referência

- Propriedades — 6 propriedades disponíveis no componente.
- Methods — 7 métodos públicos.
- Events — 19 eventos que você pode tratar.

OAuth2 Client for Web Applications

Quando o cliente OAuth2 precisa obter um token autorizado conectando a uma aplicação web, o servidor local (utilizado para obter o token autorizado) deve ser configurado com os parâmetros da aplicação web. Defina as seguintes propriedades de **LocalServerOptions** :

- **IP:** Endereço IP previamente configurado na Configuração OAuth2.
- **Port:** Porta de escuta configurada anteriormente na OAuth2 Configuration.
- **SSL:** se o servidor estiver utilizando uma conexão segura, habilite esta opção.

OAuth2 Client for Desktop Applications

Quando o Cliente OAuth2 é um aplicativo Desktop ou nativo, o Servidor Local (usado para obter o token autorizado) pode escutar em um endereço IP local (exemplo: 127.0.0.1) e a porta pode ser escolhida aleatoriamente. Portanto, o **LocalServerOptions** deve ser configurado da seguinte forma

- **IP:** 127.0.0.1
- **Port:** defina como zero, e o servidor escolherá automaticamente uma porta aleatória.

Ao criar um OAuth2 Client para aplicações Desktop ou Native, defina o **OAuth2Options.GrantType** como **auth2CodePKCE** para adicionar uma segurança extra.

OAuth2 | TsgcHTTP_OAuth2_Client_Google

Este componente permite que você faça login com sua Conta Google de forma fácil.

Configuração

O módulo requer primeiro **configurar sua Aplicação OAuth2** na sua Conta Google. Uma vez configurada, basta adicionar algumas linhas em sua aplicação para permitir que os usuários façam login com qualquer Conta Google.

O **Servidor Local** usado para ler a resposta do Google, por padrão escuta no **Endereço IP 127.0.0.1** e na **porta 0** (porta aleatória). Portanto, você deve configurar a URL de Callback na Aplicação Google. É claro que você pode modificar o Endereço IP e a porta.

Uma vez configurada a Aplicação OAuth2 na Conta Google, basta criar uma instância de **TsgcHTTP_OAuth2_Client_Google** e chamar o método **Authenticate** passando como parâmetros o **Client_Id** e o **Client_Secret**. Este **método aguarda** (por padrão, até 60 segundos) até que o usuário tenha **feito login com sucesso**. Retorna um objeto onde você pode verificar se o usuário se autenticou ou não, o Name, Id... e mais dados do perfil do usuário.

Exemplo

```
void GoogleSignIn()
{
    TsgcHTTP_OAuth2_Client_Google *oClient = new TsgcHTTP_OAuth2_Client_Google.Create(this);
    Try
    {
        TsgcOAuth2_Google_Data *oData = oClient.Authenticate("client_id", "client_secret");
        if oData->Authenticated() then
            ShowMessage(oData->UserProfile->_Name);
    }
    __Finally
        oClient->Free();
}
```

TsgcHTTP_OAuth2_Client_Microsoft

Este componente permite que você faça login com sua conta Microsoft de maneira fácil.

Configuração

O módulo requer primeiro **configurar sua Aplicação OAuth2** na sua Conta Microsoft; uma vez configurada, basta adicionar algumas linhas na sua aplicação para permitir que os usuários façam login com qualquer Conta Microsoft.

O **Servidor Local** utilizado para ler a resposta da Microsoft, por padrão escuta no **Endereço IP 127.0.0.1** e na **porta 8080** e utiliza SSL. Portanto, você deve configurar a URL de Callback como **https://localhost:8080** (a Microsoft permite apenas localhost como Endereço IP local) na Aplicação Microsoft. É claro que você pode modificar o Endereço IP e a porta.

Uma vez configurada a aplicação OAuth2 na conta Microsoft, basta criar uma instância de **TsgcHTTP_OAuth2_Client_Microsoft** e chamar o método **Authenticate** passando como parâmetros o **TenantId**, **Client_Id** e **Client_Secret**. Este **método aguarda** (por padrão, até 60 segundos) até que o usuário tenha **feito login com sucesso**. Retorna um objeto onde você pode verificar se o usuário se autenticou ou não, o Name, o Id... e mais dados do perfil do usuário.

Exemplo

```
void GoogleSignIn()
{
    TsgcHTTPComponentClient_OAuth2_Microsoft *oClient = new TsgcHTTPComponentClient_OAuth2_Microsoft.Create(this);
    Try
    {
        TsgcOAuth2_Microsoft_Data *oData = oClient.Authenticate("tenant_id", "client_id", "client_secret");
        if oData->Authenticated() then
            ShowMessage(oData->UserProfile->DisplayName);
    }
    __Finally
        oClient->Free();
    }
}
```

OAuth2 | Authorization Code Grant (RFC 6749)

Visão geral

A concessão Authorization Code é o fluxo OAuth2 mais comum. Ele é projetado para aplicações web, aplicações desktop e aplicações móveis em que o cliente pode armazenar com segurança um client secret. O fluxo envolve redirecionar o usuário ao servidor de autorização, onde ele se autentica e concede permissão. O servidor então retorna um código de autorização a uma redirect URI, e o cliente troca esse código por um token de acesso.

Fluxo

1. O cliente redireciona o usuário para a **AuthURL** com os parâmetros `client_id`, `redirect_uri`, `response_type=code`, `scope` e `state`.
2. O usuário autentica com o servidor de autorização e concede permissão ao cliente.
3. O servidor de autorização redireciona o usuário de volta para a redirect URI com um parâmetro `code`.
4. O cliente faz POST do código de autorização para a **TokenURL** junto com `client_id`, `client_secret` e `redirect_uri` para trocá-lo por um access token.
5. O servidor retorna `access_token`, `token_type`, `expires_in` e, opcionalmente, um `refresh_token`.

Configuração

Property	Descrição
<code>OAuth2Options.GrantType</code>	Defina como auth2Code .
<code>OAuth2Options.ClientId</code>	O identificador do cliente emitido pelo servidor de autorização.
<code>OAuth2Options.ClientSecret</code>	O client secret emitido pelo servidor de autorização.
<code>AuthorizationServerOptions.AuthURL</code>	A URL do endpoint de autorização para o qual o usuário é redirecionado para se autenticar.
<code>AuthorizationServerOptions.TokenURL</code>	A URL do endpoint de token onde o authorization code é trocado por um access token.
<code>AuthorizationServerOptions.Scope</code>	O escopo da requisição de acesso (por exemplo, openid, profile, email).
<code>LocalServerOptions.IP</code>	O endereço IP do servidor de redirecionamento local (por exemplo, 127.0.0.1).
<code>LocalServerOptions.Port</code>	A porta do servidor de redirecionamento local (por exemplo, 8080).

Eventos

Event	Descrição

<code>OnBeforeAuthorizeCode</code>	Disparado antes de o cliente redirecionar o usuário para o endpoint de autorização. Permite a personalização da requisição de autorização.
<code>OnAfterAuthorizeCode</code>	Disparado após o código de autorização ser recebido do servidor.
<code>OnErrorAuthorizeCode</code>	Disparado quando ocorre um erro durante a etapa de código de autorização.
<code>OnBeforeAccessToken</code>	Disparado antes do cliente trocar o código de autorização por um access token.
<code>OnAfterAccessToken</code>	Disparado após o access token ser recebido. Utilize este evento para recuperar o valor do token.
<code>OnErrorAccessToken</code>	Disparado quando ocorre um erro durante a etapa de troca de token.

Exemplo

```
OAuth2->OAuth2Options->GrantType = 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();
```

Token Refresh

Quando um access token expira, você pode usar o refresh token para obter um novo sem exigir interação do usuário. Chame o método `Refresh` com o valor do refresh token:

O evento `OnAfterAccessToken` será disparado com o novo token de acesso. Se a atualização falhar, o evento `OnErrorAccessToken` será disparado.

Quando Utilizar

Utilize a concessão `Authorization Code` para:

- Aplicações web com um backend do lado do servidor que pode armazenar com segurança o client secret.
- Aplicações renderizadas no lado do servidor (por exemplo, ASP.NET, PHP, Java).
- Aplicações desktop em que o client secret pode ser razoavelmente protegido.

Para clientes públicos (apps nativos, SPAs) onde o client secret não pode ser armazenado com segurança, considere usar [Authorization Code com PKCE](#) em vez disso.

OAuth2 | Authorization Code with PKCE (RFC 7636)

Visão geral

O grant Authorization Code com PKCE (Proof Key for Code Exchange) estende o fluxo padrão de Authorization Code com uma camada de segurança adicional. Ele é projetado para aplicações nativas, aplicações móveis e single-page applications (SPAs) onde o client secret não pode ser armazenado com segurança. Em vez de depender de um client secret, o cliente gera um code verifier e um challenge criptográficos que comprovam que a entidade que troca o código de autorização é a mesma entidade que iniciou o fluxo.

Como o PKCE Funciona

1. O cliente gera um **code_verifier** aleatório (32 bytes de dados criptograficamente aleatórios, codificados em Base64URL).
2. O cliente computa o **code_challenge** como o hash SHA-256 do code_verifier, codificado em Base64URL.
3. O cliente envia o code_challenge e code_challenge_method=S256 com a requisição de autorização.
4. Após a autorização do usuário, o cliente envia o code_verifier original junto com a requisição de troca de token.
5. O servidor de autorização verifica se $\text{SHA256}(\text{code\_verifier}) == \text{code\_challenge}$ antes de emitir o token.

Isto evita ataques de interceptação de authorization code, porque um atacante que intercepta o código não consegue trocá-lo sem o code_verifier original.

Configuração

Property	Descrição
<code>OAuth2Options.GrantType</code>	Defina como auth2CodePKCE .
<code>OAuth2Options.ClientId</code>	O identificador do cliente emitido pelo servidor de autorização.
<code>OAuth2Options.ClientSecret</code>	Opcional. Alguns provedores o exigem mesmo com PKCE; outros não.
<code>AuthorizationServerOptions.AuthURL</code>	A URL do endpoint de autorização para o qual o usuário é redirecionado para se autenticar.
<code>AuthorizationServerOptions.TokenURL</code>	A URL do endpoint de token onde o authorization code é trocado por um access token.
<code>AuthorizationServerOptions.Scope</code>	O escopo da requisição de acesso.
<code>LocalServerOptions.IP</code>	O endereço IP do servidor de redirecionamento local (por exemplo, 127.0.0.1).
<code>LocalServerOptions.Port</code>	A porta do servidor de redirect local. Defina como 0 para uma porta disponível aleatória.

Exemplo

```
OAuth2->OAuth2Options->GrantType = 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();
```

Random Port

Quando `LocalServerOptions.Port` é definido como **0**, o componente seleciona automaticamente uma porta disponível aleatória. Isso é recomendado para aplicações desktop e móveis, pois evita conflitos de porta. A porta selecionada é incluída no redirect URI enviado ao servidor de autorização.

Token Refresh

Os fluxos PKCE normalmente retornam um token de atualização. Use o método `Refresh` para obter um novo token de acesso:

Quando Utilizar

Utilize o grant Authorization Code com PKCE para:

- Aplicações desktop (Windows, macOS, Linux nativos).
- Aplicações móveis (iOS, Android).
- Single-page applications (SPAs) executadas no navegador.
- Qualquer cliente público que não possa armazenar com segurança um client secret.
- Como uma alternativa mais segura à concessão [Authorization Code](#) padrão, mesmo para clientes confidenciais.

OAuth2 | Client Credentials Grant (RFC 6749)

Visão geral

O grant Client Credentials é utilizado para autenticação máquina-a-máquina (M2M) onde nenhuma interação do usuário é necessária. O cliente se autentica diretamente com o servidor de autorização utilizando suas próprias credenciais (`client_id` e `client_secret`) e recebe um access token. Não há etapa de authorization code nem login do usuário.

Este tipo de grant é apropriado quando o cliente está agindo em seu próprio nome, em vez de em nome de um usuário.

Fluxo

1. O cliente faz POST de `client_id`, `client_secret`, `grant_type=client_credentials` e `scope` para o endpoint de token.
2. O servidor de autorização valida as credenciais do cliente.
3. O servidor retorna `access_token`, `token_type` e `expires_in`.

Observe que refresh tokens normalmente não são emitidos com este grant type, já que o cliente pode simplesmente solicitar um novo access token usando suas credenciais.

Configuração

Property	Descrição
<code>OAuth2Options.GrantType</code>	Defina como auth2ClientCredentials .
<code>OAuth2Options.ClientId</code>	O identificador do cliente emitido pelo servidor de autorização.
<code>OAuth2Options.ClientSecret</code>	O client secret emitido pelo servidor de autorização.
<code>AuthorizationServerOptions.TokenURL</code>	A URL do endpoint de token. Nenhuma AuthURL é necessária, pois não há etapa de autorização do usuário.
<code>AuthorizationServerOptions.Scope</code>	O escopo da requisição de acesso.

Exemplo

```
OAuth2->OAuth2Options->GrantType = 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 Utilizar

Utilize o grant Client Credentials para:

- Serviços de backend e daemons que rodam sem interação do usuário.

- Comunicação de API servidor-a-servidor.
- Jobs agendados, processos em batch e tarefas cron.
- Autenticação microsserviço-a-microsserviço.
- Qualquer cenário em que a aplicação age em seu próprio nome, não em nome de um usuário.

OAuth2 | Resource Owner Password Credentials Grant (RFC 6749)

Visão geral

A concessão Resource Owner Password Credentials (ROPC) permite que o cliente colete o nome de usuário e a senha do usuário diretamente e os troque por um access token. As credenciais do usuário são enviadas ao endpoint de token, e o servidor retorna um access token se forem válidas.

Este grant type só deve ser utilizado quando há um alto grau de confiança entre o resource owner e o cliente, como quando o cliente é uma aplicação first-party desenvolvida pela mesma organização que opera o servidor de autorização.

Aviso de Segurança

Este tipo de grant é considerado menos seguro do que outros fluxos OAuth2. Ele expõe as credenciais do usuário diretamente à aplicação cliente, o que viola o princípio de autorização delegada. A especificação OAuth 2.1 tornou obsoleto este tipo de grant. Utilize [Authorization Code com PKCE](#) sempre que possível.

Fluxo

1. O usuário fornece o username e a senha diretamente à aplicação cliente.
2. O cliente faz um POST de username, password, client_id, client_secret, grant_type=password e scope ao endpoint de token.
3. O servidor de autorização valida as credenciais.
4. O servidor retorna access_token, token_type, expires_in e, opcionalmente, um refresh_token.

Configuração

Property	Descrição
<code>OAuth2Options.GrantType</code>	Defina como auth2ResourceOwnerPassword .
<code>OAuth2Options.ClientId</code>	O identificador do cliente emitido pelo servidor de autorização.
<code>OAuth2Options.ClientSecret</code>	O client secret emitido pelo servidor de autorização.
<code>OAuth2Options.UserName</code>	O nome de usuário do resource owner.
<code>OAuth2Options.Password</code>	A senha do resource owner.
<code>AuthorizationServerOptions.TokenURL</code>	A URL do endpoint de token. Nenhuma AuthURL é necessária, pois as credenciais do usuário são fornecidas diretamente.
<code>AuthorizationServerOptions.Scope</code>	O escopo da requisição de acesso.

Exemplo

```
OAuth2->OAuth2Options->GrantType = 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();
```

Token Refresh

Se o servidor retornar um refresh token, você pode usá-lo para obter um novo token de acesso sem reinserir as credenciais do usuário:

Quando Utilizar

Utilize o grant Resource Owner Password Credentials apenas para:

- Aplicações legadas que não podem ser migradas para um fluxo mais seguro.
- Aplicações de primeira parte (first-party) onde você controla tanto o cliente quanto o servidor de autorização.
- Cenários de migração em que você está fazendo a transição da autenticação direta de username/password para OAuth2.

Para todos os outros casos, use [Authorization Code com PKCE](#) ou [Authorization Code](#).

OAuth2 | Device Authorization Grant (RFC 8628)

Visão geral

O grant Device Authorization (também conhecido como fluxo Device Code) foi projetado para dispositivos com restrições de entrada que não conseguem exibir um navegador facilmente ou aceitar entrada de teclado. Exemplos incluem smart TVs, dispositivos IoT, media players, ferramentas CLI e quiosques. O dispositivo exibe um código curto que o usuário insere em outro dispositivo (como um telefone ou computador) para concluir a autorização.

Fluxo

1. O cliente envia via POST `client_id` e `scope` para a **DeviceAuthorizationURL**.
2. O servidor retorna `device_code`, `user_code`, `verification_uri`, `expires_in` e `interval`.
3. O cliente dispara o evento **OnDeviceCode** com o `user code` e a `verification URI`. A aplicação os exibe ao usuário (por exemplo, "Vá para `https://provider.com/device` e insira o código: ABCD-1234").
4. O usuário navega até a URI de verificação em outro dispositivo e digita o `user code`.
5. O usuário autentica e concede permissão no dispositivo secundário.
6. Enquanto isso, o cliente faz polling automaticamente no `token endpoint` a cada `interval` segundos utilizando o `device code`.
7. Quando o usuário conclui a autorização, o próximo poll retorna o `access_token`.
8. Se o código do dispositivo expirar antes de o usuário autorizar, o evento **OnDeviceCodeExpired** é disparado.

Configuração

Property	Descrição
<code>OAuth2Options.GrantType</code>	Defina como auth2DeviceCode .
<code>OAuth2Options.ClientId</code>	O identificador do cliente emitido pelo servidor de autorização.
<code>OAuth2Options.ClientSecret</code>	Opcional. Alguns provedores o exigem, outros não.
<code>AuthorizationServerOptions.TokenURL</code>	A URL do endpoint de token usada para polling.
<code>AuthorizationServerOptions.DeviceAuthorizationURL</code>	O endpoint de autorização de dispositivo onde o cliente solicita um código de dispositivo.
<code>AuthorizationServerOptions.Scope</code>	O escopo da requisição de acesso.

Eventos

Event	Descrição

<code>OnDeviceCode</code>	Disparado quando o device code e o user code são recebidos do servidor. Exiba o user code e o verification URI ao usuário.
<code>OnDeviceCodeExpired</code>	Disparado quando o device code expira antes de o usuário concluir a autorização. A aplicação deve solicitar que o usuário reinicie o fluxo.
<code>OnAfterAccessToken</code>	Disparado quando o usuário conclui a autorização e o access token é recebido.
<code>OnErrorAccessToken</code>	Disparado quando ocorre um erro não recuperável durante o device flow.

Exemplo

Tratando o Evento Device Code

```
OAuth2->OAuth2Options->GrantType = 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->Start();
```

Comportamento de Polling

Após chamar `start`, o componente faz polling automático no endpoint de token no intervalo especificado pelo servidor (normalmente 5 segundos). Se o servidor responder com um erro `slow_down`, o componente aumenta o intervalo de polling em 5 segundos. O polling continua até que uma das seguintes situações ocorra:

- O usuário conclui a autorização e o token é retornado.
- O device code expira (`OnDeviceCodeExpired` é disparado).
- Ocorre um erro irreversível (`OnErrorAccessToken` dispara).

Quando Utilizar

Utilize o grant Device Authorization para:

- Smart TVs e dispositivos de streaming.
- Ferramentas de interface de linha de comando (CLI).
- Dispositivos IoT com capacidade de entrada limitada.
- Quiosques e sinalização digital.
- Qualquer dispositivo onde digitar uma URL completa ou credenciais seja impraticável.

OAuth2 | DPoP - Demonstrating Proof of Possession (RFC 9449)

Visão geral

O DPoP (Demonstrating Proof of Possession) é um mecanismo de segurança que vincula tokens de acesso a um cliente específico, exigindo a prova de posse de uma chave privada. Diferentemente dos bearer tokens, que podem ser usados por qualquer um que os obtenha, os tokens vinculados a DPoP são inúteis sem a chave privada correspondente. Isso fornece proteção contra roubo de token e ataques de replay.

O DPoP não é um grant type em si, mas um aprimoramento de segurança que pode ser utilizado com qualquer grant type OAuth2 (Authorization Code, Client Credentials, etc.).

Como funciona

1. O cliente gera um par de chaves assimétricas (ES256 ou RS256).
2. O cliente cria um **DPoP proof JWT** com a seguinte estrutura:
 - Header: typ: dpop+jwt, alg: ES256, jwk: {public key}
 - Payload: htm (método HTTP), htu (URI HTTP), iat (issued at), jti (identificador único)
3. O cliente envia a prova DPoP como um cabeçalho HTTP (DPoP: <proof>) com a requisição de token.
4. O servidor de autorização vincula o token emitido ao thumbprint JWK da chave pública do cliente.
5. O servidor retorna token_type: DPoP em vez de token_type: Bearer.
6. Para chamadas de API subsequentes, o cliente envia tanto o cabeçalho Authorization: DPoP <token> quanto um novo cabeçalho DPoP: <proof> específico para aquela requisição.

Key Generation

Você pode gerar um par de chaves ES256 diretamente no Delphi utilizando o método **GenerateDPoPKeyPair**. Isso utiliza OpenSSL internamente para criar o par de chaves e define automaticamente tanto PrivateKey quanto PublicKeyJWK:

```
OAuth2->DPoPOptions->Enabled = true;
OAuth2->DPoPOptions->Algorithm = dpopES256;
OAuth2->GenerateDPoPKeyPair(); // Generates PrivateKey + PublicKeyJWK automatically
OAuth2->Start();
```

PublicKeyToJWK — Converter Qualquer Chave ou Certificado em JWK

Se você já possui uma chave pública, chave privada ou certificado X.509 (formato PEM), pode convertê-lo para JWK utilizando a função de classe **PublicKeyToJWK**. Ela aceita qualquer uma dessas entradas e retorna o JWK JSON correto:

Input	Cabeçalho PEM
Chave pública EC ou RSA	-----BEGIN PUBLIC KEY-----
EC private key	-----BEGIN EC PRIVATE KEY-----
RSA private key	-----BEGIN RSA PRIVATE KEY-----
Certificado X.509	-----BEGIN CERTIFICATE-----

A partir de um arquivo de chave PEM

```
// From a public key file
OAuth2->DPoPOptions->PublicKeyJWK =
    OAuth2->PublicKeyToJWK(
        TFile::ReadAllText("public_key.pem"));

// From a private key file (extracts the public part)
OAuth2->DPoPOptions->PublicKeyJWK =
    OAuth2->PublicKeyToJWK(
        TFile::ReadAllText("private_key.pem"));
```

A partir de um certificado de smart card

Smart cards e HSMs protegem a chave privada — ela não pode ser lida ou exportada. No entanto, o **certificado** (que contém a chave pública) é sempre exportável. Exporte o certificado como PEM e passe-o para `PublicKeyToJWK`:

```
// Export the certificate from the smart card
UnicodeString 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;
```

Esta abordagem funciona com qualquer cartão ou HSM que possa exportar seu certificado: tokens PKCS#11, smart cards Windows CNG, chaves de segurança USB (YubiKey, Feitian) e HSMs em nuvem (Azure Key Vault, AWS CloudHSM).

Utilizando as ferramentas de linha de comando do OpenSSL

Você também pode gerar um par de chaves externamente:

Opções de DPoP

Property	Descrição
<code>DPoPOptions.Enabled</code>	Defina como True para habilitar a geração de prova DPoP.
<code>DPoPOptions.Algorithm</code>	O algoritmo de assinatura. Valores suportados: ES256 (recomendado), RS256 .
<code>DPoPOptions.PrivateKey</code>	A chave privada codificada em PEM usada para assinar o JWT de prova DPoP.
<code>DPoPOptions.PublicKeyJWK</code>	A chave pública no formato JWK (JSON Web Key), incluída no header da prova DPoP.

Métodos

Method	Descrição
<code>GenerateDPoPKeyPair</code>	Gera um par de chaves ES256 utilizando o OpenSSL e define automaticamente tanto <code>DPoPOptions.PrivateKey</code> (PEM) quanto <code>DPoPOptions.PublicKeyJWK</code> (JSON).

<code>GetDPoPProof(Method, URL, AccessToken)</code>	Gera um JWT de prova DPoP para o método HTTP, URL e access token fornecidos. Retorna a string do JWT assinado. A prova inclui o claim <code>ath</code> (hash do access token) para vincular a prova ao token.
<code>GetDPoPJWKThumbprint</code>	Retorna o thumbprint JWK (SHA-256) RFC 7638 da chave pública, utilizado pelo servidor para vincular o token.

Tratamento de Nonce

Alguns authorization servers exigem um nonce fornecido pelo servidor na prova DPoP. Quando o servidor responde com um cabeçalho `DPoP-Nonce` e um status 400 ou 401, o componente automaticamente:

1. Extrai o nonce do cabeçalho de resposta `DPoP-Nonce`.
2. Inclui o nonce na claim `nonce` da próxima prova DPoP JWT.
3. Repete a requisição com a prova atualizada.

Assinatura Personalizada com OnDPoPSign (HSM / Smart Card)

Em alguns ambientes, a chave privada é armazenada em um hardware security module (HSM) ou smart card (por exemplo, via PKCS#11 ou Windows CNG) e **não pode ser extraída**. O evento `OnDPoPSign` permite que você realize a assinatura externamente enquanto o componente lida com todo o resto (estrutura da prova, nonce, cabeçalhos). Quando `OnDPoPSign` está atribuído, o componente o chama antes de tentar a assinatura interna. Se você definir `Handled := True` e fornecer a `Signature`, o componente usa sua assinatura e ignora a assinatura interna. A propriedade `DPoPOptions.PrivateKey` não é necessária neste caso.

Assinatura do Evento

Parameter	Direction	Descrição
<code>SigningInput</code>	in	Os dados a assinar: <code>Base64URL(header) + "." + Base64URL(payload)</code> . Esta é a entrada de assinatura JWS padrão.
<code>Algorithm</code>	in	O nome do algoritmo: <code>"ES256"</code> ou <code>"RS256"</code> .
<code>Signature</code>	out	Defina isto com os bytes da assinatura codificada em Base64URL produzidos pelo seu dispositivo de assinatura.
<code>Handled</code>	out	Defina como <code>True</code> para usar sua assinatura personalizada. Se <code>False</code> , o componente recorre à assinatura interna usando <code>PrivateKey</code> .

Smart Card Example

O exemplo a seguir mostra como utilizar DPoP com um signing card. A chave privada nunca sai do cartão, apenas o hash é enviado ao cartão para assinatura.

```
// Configure DPoP - no PrivateKey needed when using OnDPoPSign
OAuth2->DPoPOptions->Enabled = true;
OAuth2->DPoPOptions->Algorithm = dpopES256;
OAuth2->DPoPOptions->PublicKeyJWK =
    "{\"kty\":\"EC\",\"crv\":\"P-256\",\"x\":\"...\",\"y\":\"...\"}";
OAuth2->OnDPoPSign = OnDPoPSignHandler;
OAuth2->Start();

void __fastcall TForm1::OnDPoPSignHandler(TObject *Sender,
    const UnicodeString SigningInput, const UnicodeString Algorithm,
    UnicodeString &Signature, bool &Handled)
```



```
{
    // Hash and sign with smart card
    TBytes Hash = GetHashSHA256(SigningInput);
    TBytes Sig = MySmartCardAPI->SignHash(Hash);
    Signature = EncodeBase64URL(Sig);
    Handled = true;
}
```

Essa abordagem funciona com qualquer backend de assinatura: tokens PKCS#11, Windows CNG (CryptoAPI Next Generation), Azure Key Vault, AWS CloudHSM ou qualquer hardware personalizado que possa produzir assinaturas ECDSA/RSA.

Exemplo

```
// Configure the grant type
OAuth2->OAuth2Options->GrantType = 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 = LoadPrivateKeyFromFile("dpop_private.pem");
OAuth2->DPoPOptions->PublicKeyJWK = "{\"kty\":\"EC\",\"crv\":\"P-256\",\"x\":\"...\", \"y\":\"...\"}";
OAuth2->Start();
// Generate DPoP proof for API calls
UnicodeString Proof = OAuth2->GetDPoPProof("GET", "https://api.provider.com/resource");
```

TsgcHTTP_OAuth2_Server

Este componente fornece a implementação do protocolo OAuth2 nos componentes do lado do servidor.

Introdução

Este componente fornece a implementação do protocolo OAuth2 nos componentes do lado do servidor. Os componentes de servidor têm uma propriedade chamada `Authorization.OAuth.OAuth2` onde você pode atribuir uma instância de `TsgcHTTP_OAuth2_Server`, de modo que, se a `Authentication` estiver habilitada e a propriedade `OAuth2` estiver anexada ao componente `OAuth2 Server`, as `WebSocket` e `HTTP Requests` exigirão um `Bearer Token` para serem processadas; caso contrário, a conexão será fechada automaticamente.

```
TsgcHTTP_OAuth2_Server *OAuth2 = new TsgcHTTP_OAuth2_Server(this);
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2 = OAuth2;
```

O servidor suporta os seguintes tipos de autorização:

- **auth2Code:** É utilizado para realizar autenticação e autorização na maioria dos tipos de aplicações, incluindo aplicações de página única, aplicações web e aplicações instaladas nativamente. O fluxo permite que as aplicações adquiram com segurança `access_tokens` que podem ser utilizados para acessar recursos protegidos, bem como `refresh tokens` para obter `access_tokens` adicionais, e `ID tokens` para o usuário autenticado.
- **auth2ClientCredentials:** Este tipo de concessão é comumente usado para interações servidor-a-servidor que devem rodar em segundo plano, sem interação imediata com um usuário. Esses tipos de aplicações são frequentemente chamados de `daemons` ou `contas de serviço`.
- **password** (Resource Owner Password Credentials): Permite que uma aplicação faça o login do usuário tratando diretamente suas credenciais. O cliente envia o nome de usuário e a senha do usuário ao endpoint de token.
- **urn:ietf:params:oauth:grant-type:device_code** (Device Code): Device Authorization Grant conforme a [RFC 8628](#). Permite que dispositivos com restrições de entrada (smart TVs, dispositivos IoT) obtenham autorização do usuário fazendo com que o usuário autorize em um dispositivo secundário.

O tipo de `Authorization` pode ser personalizado ao registrar o App; por padrão, todos os tipos de autorização são suportados.

Endpoints

Por padrão, o componente é configurado com os seguintes endpoints para tratar requisições de `Authorization` e `Token`

Authorization: `/sgc/oauth2/auth`

Token: `/sgc/oauth2/token`

Revocation: `/sgc/oauth2/revoke`

Introspection: `/sgc/oauth2/introspect`

Device Authorization: `/sgc/oauth2/device`

Device Verification: `/sgc/oauth2/device/verify`

Portanto, se o servidor estiver escutando na porta 443 e o domínio for `www.esegece.com`, os EndPoints serão:

Authorization: <https://www.esegece.com/sgc/oauth2/auth>

Token: <https://www.esegece.com/sgc/oauth2/token>

Revocation: <https://www.esegece.com/sgc/oauth2/revoke>

Introspection: <https://www.esegece.com/sgc/oauth2/introspect>

Device Authorization: <https://www.esegece.com/sgc/oauth2/device>

Device Verification: <https://www.esegece.com/sgc/oauth2/device/verify>

Os endpoints podem ser configurados na propriedade `OAuth2Options`.

Por padrão, o **PKCE** (é uma extensão do fluxo `Authorization Code` para evitar ataques de `CSRF` e de injeção de código de autorização) está habilitado.

Configuração

Antes de poder iniciar o processo de OAuth2, você deve registrar quais Apps estarão disponíveis, isso é feito utilizando a propriedade Apps do componente servidor OAuth2.

Registrar app

Use `Apps.AddApp` para adicionar um novo Application ao servidor OAuth2, você deve definir os seguintes parâmetros:

- **App Name:** é o nome da Aplicação. Exemplo: MyApp
- **RedirectURI:** é para onde as respostas serão redirecionadas. Exemplo: `http://127.0.0.1:8080`
- **ClientId:** é informação pública e é o ID do cliente.
- **ClientSecret:** deve ser mantido confidencial.

Opcionalmente, você pode definir os seguintes parâmetros:

- **ExpiresIn:** por padrão é 3600 segundos, portanto o token expirará em 1 hora; você pode definir um valor maior se precisar.
- **RefreshToken:** por padrão, os refresh tokens são suportados; se não, defina este parâmetro como false.
- **AllowedGrantTypes:** por padrão, todos os grant types são suportados (auth2Code e auth2ClientCredentials), mas o servidor pode ser configurado para permitir somente Code Authorization ou somente Client Credentials.

Excluir App

Use `Apps.RemoveApp` para excluir um App existente.

AddToken

Se o servidor tiver sido reiniciado enquanto havia alguns tokens emitidos, você pode recuperar esses tokens utilizando o método `AddToken` antes de iniciar o OAuth2 Server e após registrar os Apps

- **AppName:** o nome da aplicação.
- **Token:** token de acesso.
- **Expires:** quando o token expira.
- **RefreshToken:** refresh token.

RemoveToken

Remove um Token já emitido.

OAuth2Options

A propriedade `OAuth2Options` permite configurar os endpoints do servidor e recursos opcionais.

Revogação

Revogação de token conforme a [RFC 7009](#). Quando habilitado, os clientes podem revogar access ou refresh tokens emitidos anteriormente.

- **OAuth2Options.Revocation.Enabled:** defina como True para habilitar o endpoint de revogação.
- **OAuth2Options.Revocation.URL:** o caminho da URL do endpoint. Padrão: `/sgc/oauth2/revoke`

Introspection

Introspecção de token conforme a [RFC 7662](#). Quando habilitada, os servidores de recursos podem consultar o servidor de autorização para determinar o estado ativo e os metadados de um token.

- **OAuth2Options.Introspection.Enabled:** defina como True para habilitar o endpoint de introspecção.
- **OAuth2Options.Introspection.URL:** o caminho da URL do endpoint. Padrão: `/sgc/oauth2/introspect`

DeviceAuthorization

Device Authorization Grant conforme a [RFC 8628](#). Quando habilitado, dispositivos com entrada limitada podem solicitar autorização fazendo o usuário autorizar em um dispositivo secundário.

- **OAuth2Options.DeviceAuthorization.Enabled:** defina como True para habilitar o endpoint de device authorization.
- **OAuth2Options.DeviceAuthorization.URL:** o path da URL do endpoint para requisições de device code. Padrão: /sgc/oauth2/device
- **OAuth2Options.DeviceAuthorization.VerificationURL:** o caminho da URL do endpoint para a página de verificação do usuário. Padrão: /sgc/oauth2/device/verify
- **OAuth2Options.DeviceAuthorization.ExpiresIn:** o tempo de vida em segundos do device code. Padrão: 600 (10 minutos).
- **OAuth2Options.DeviceAuthorization.Interval:** o intervalo mínimo de polling em segundos que o cliente deve utilizar ao consultar o endpoint de token. Padrão: 5.

Referência

- Propriedades — 2 propriedades disponíveis no componente.
- Métodos — 11 métodos públicos.
- Eventos — 13 eventos que você pode tratar.

Guides

- **QuickStart**
 - [OAuth2 Server Example](#)
 - [OAuth2 Customize Sign-in HTML](#)
 - [OAuth2 Server Endpoints](#)
 - [OAuth2 Register Apps](#)
 - [OAuth2 Recover Access Tokens](#)
- **Authenticate**
 - [OAuth2 Server Authentication](#)
 - [OAuth2 None Authenticate some URLs](#)

OAuth2 | Exemplo de Servidor

Vamos fazer um exemplo simples de servidor OAuth2, usando um [TsgcWebSocketHTTPServer](#).

Primeiro, crie um novo TsgcWebSocketHTTPServer escutando na porta 443 e utilizando um certificado autoassinado no arquivo sgc.pem.

```
oServer = new TsgcWebSocketHTTPServer(this);
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;
```

Em seguida, crie uma nova instância de TsgcHTTP_OAuth2_Server e atribua-a ao servidor criado anteriormente. Registre uma nova Aplicação com os seguintes valores:

Name: MyApp
 RedirectURI: http://127.0.0.1:8080
 ClientId: client-id
 ClientSecret: client-secret

```
OAuth2 = new TsgcHTTP_OAuth2_Server.Create(this);
OAuth2->Apps->AddApp("MyApp", "http://127.0.0.1:8080", "client-id", "client-secret");
oServer->Authentication->Enabled = true;
oServer->Authentication->OAuth->OAuth2 = OAuth2;
```

Em seguida, manipule o evento OnOAuth2Authentication do componente de servidor OAuth2 e implemente seu próprio método para fazer login dos usuários. Vou usar o par "user/secret" para aceitar um login.

```
void OnOAuth2Authentication(TsgcWSConnection *Connection, TsgcHTTPOAuth2Request *OAuth2, string aUser,
    string aPassword, ref bool Authenticated)
{
    if ((aUser == "user") and (aPassword == "secret"))
    {
        Authenticated = true;
    }
}
```

Por fim, inicie o servidor e utilize um cliente OAuth2 para fazer login; por exemplo, você pode utilizar o [TsgcHTTP_OAuth2_Client](#) incluído na biblioteca sgcWebSockets.

 OAuth2

Configuration

Auth. URL

Token URL

Scope

Authorization Server Options

IP

Port

Redirect URL

Local Server Options

ClientId

Secret

Username

Password

OAuth2 Options

New Access Token

Access Token

Token Type

Expires In

Refresh Token

Scope

Refresh Token

Solicite um Novo Access Token, uma nova sessão de Navegador Web será exibida e o usuário deve Permitir a conexão e então fazer login.

OAuth2 Authorization

Sign in

Introduce your username and password.

GO BACK

Sign in

SIGN IN

Se o login for bem-sucedido, um novo Token será retornado ao cliente. A partir de então, todas as requisições devem incluir este token bearer nos Headers 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 | Personalizar HTML de Login

Quando um cliente OAuth2 faz uma requisição para obter um novo Access Token, uma página web é exibida em um navegador web para permitir esta conexão e fazer login com um usuário e senha.

A página HTML é incluída por padrão no componente Server, mas este código pode ser personalizado usando o evento **OnAuth2BeforeDispatchPage**.

```
void OnAuth2BeforeDispatchPage(TObject *Sender; TsgcHTTPOAuth2Request *OAuth2; ref string HTML)
{
    HTML = "your custom html";
}
```

Se você personalizar seu HTML com um código HTML completamente novo, ao menos você deve manter o form onde o Username e a senha são enviados:

```
<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>
```

O parâmetro id, que está oculto, deve manter o mesmo valor do formulário original para permitir que o servidor identifique a requisição.

OAuth2 | Server Endpoints

Por padrão, o Servidor OAuth2 usa os seguintes Endpoints:

Authorization: /sgc/oauth2/auth

Token: /sgc/oauth2/token

Revocation: /sgc/oauth2/revoke (POST) - Revoga tokens conforme [RFC 7009](#)

Introspection: /sgc/oauth2/introspect (POST) - Retorna metadados do token conforme a [RFC 7662](#)

Device Authorization: /sgc/oauth2/device (POST) - Emite device codes conforme a [RFC 8628](#)

Device Verification: /sgc/oauth2/device/verify (GET/POST) - Página de verificação do usuário

O que significa que, se o seu servidor escuta no IP 80.54.41.30 e na porta 8443, os endpoints OAuth2 completos serão:

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

Device Verification: https://80.54.41.30:8443/sgc/oauth2/device/verify

Estes endpoints podem ser modificados facilmente, basta acessar a propriedade OAuth2Options do componente e modificar as URLs de Authorization e Token.

Exemplo: se seus endpoints devem estar

Authorization: https://80.54.41.30:8443/authentication/auth

Token: https://80.54.41.30:8443/authentication/token

Defina a propriedade OAuth2Options com os seguintes valores:

OAuth2Options.Authorization.URL = /authentication/auth

OAuth2Options.Token.URL = /authentication/token

A mesma abordagem se aplica aos outros endpoints:

OAuth2Options.Revocation.URL = /authentication/revoke

OAuth2Options.Introspection.URL = /authentication/introspect

OAuth2Options.DeviceAuthorization.URL = /authentication/device

OAuth2Options.DeviceAuthorization.VerificationURL = /authentication/device/verify

OAuth2 | Register Apps

Antes de um novo OAuth2 ser requisitado por um cliente, o App deve ser registrado no servidor. O registro de um novo App requer as seguintes informações:

- **App Name:** é o nome da Aplicação. Exemplo: MyApp
- **RedirectURI:** é para onde as respostas serão redirecionadas. Exemplo: `http://127.0.0.1:8080`
- **ClientId:** é informação pública e é o ID do cliente.
- **ClientSecret:** deve ser mantido confidencial.

Opcionalmente, você pode definir os seguintes parâmetros:

- **ExpiresIn:** por padrão é 3600 segundos, portanto o token expirará em 1 hora; você pode definir um valor maior se precisar.
- **RefreshToken:** por padrão, os refresh tokens são suportados; se não, defina este parâmetro como false.

Se um novo cliente quiser se autenticar utilizando OAuth2, primeiro o App precisa estar registrado no servidor; você pode utilizar:

1. RegisterApp

Este método requer o App Name e o RedirectURI, e retornará um ClientId e um ClientSecret.

2. Apps.AddApp

Este método requer AppName, RedirectURI, ClientId e ClientSecret. Normalmente você pode utilizar este método quando um servidor já tem alguns Apps criados e você quer carregá-los antes de iniciá-lo.

Ambos os métodos fazem o mesmo, registram a aplicação no servidor, mas o primeiro é mais útil quando a aplicação é registrada pela primeira vez e o segundo método quando você deseja carregar todas as aplicações registradas antes de iniciar o servidor (porque estão salvas em um banco de dados, por exemplo).

OAuth2 | Recuperar Access Tokens

Se o OAuth2 Server for destruído (porque foi reiniciado) e houver Access Tokens válidos emitidos, eles são perdidos por padrão. Você pode recuperar esses Access Tokens usando o método **AddToken**. Este método armazena os tokens no OAuth2 Server.

Adicionar um Token requer as seguintes informações:

- **AppName:** o nome da aplicação.
- **Token:** token de acesso.
- **Expires:** quando o token expira.
- **RefreshToken:** refresh token.

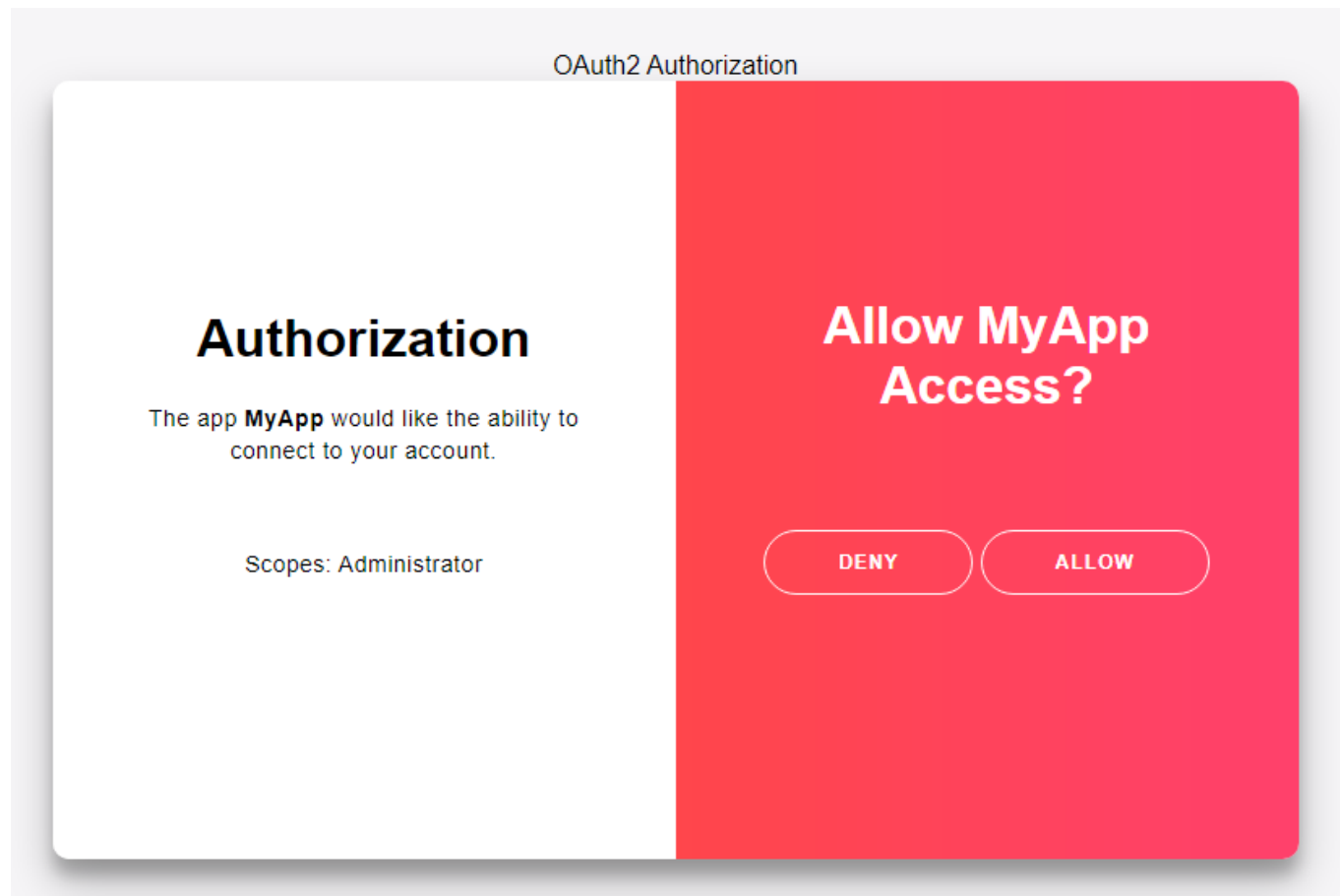
Você pode salvar os tokens emitidos tratando o evento **OAuth2AfterAccessToken**.

```
private void OnOAuth2AfterAccessToken(TObject *Sender, TsgcWSConnection *Connection, TsgcHTTPOAuth2Request *OAuth2Request,
    string aResponse)
{
    // ... store OAuth2 Token data
}

OAuth2 = new TsgcHTTP_OAuth2_Server.Create(this);
OAuth2->Apps->AddApp("MyApp", "http://127.0.0.1:8080", "client-id", "client-secret");
OAuth2->AddToken("MyApp", "12146ce12b0e4813987f2794f768905cefc39da6fbd54f6d9b38387489280608", EncodeDate(2022,1,1), "ef3e3dfa56ec44109c3d345b1541f08e539ce21432d9433099b48a3d08d34bc0");
oServer->Authentication->Enabled = true;
oServer->Authentication->OAuth->OAuth2 = OAuth2;
```

OAuth2 | Autenticação do Servidor

Quando um cliente OAuth2 solicita uma nova Autorização, o servidor exibe uma página web onde o usuário deve permitir a conexão e então fazer login. Esta página é fornecida pela biblioteca `sgcWebSockets` e é despachada automaticamente quando um cliente solicita uma Autorização.



Se o usuário Permitir o acesso, um formulário de login será exibido onde o usuário deve definir o Username e a Password.

Estes dados serão recebidos no evento `OnOAuth2Authentication`, portanto você deve validar se o usuário/senha está correto e, se estiver, então definir o parâmetro `Authenticated` como `true`.

```
void OnOAuth2Authentication(TsgcWSConnection *Connection, TsgcHTTPOAuth2Request *OAuth2, string aUser,
    string aPassword, ref bool Authenticated)
{
    if ((aUser == "user") and (aPassword == "secret"))
    {
        Authenticated = true;
    }
}
```

OAuth2 | None Authenticate URLs

Por padrão, quando o OAuth2 está habilitado no lado do servidor, todas as requisições HTTP exigem autenticação usando Bearer Tokens.

Se você quiser permitir que algumas URLs sejam acessadas sem a necessidade de usar um Bearer Token, você pode usar o evento **OnOAuth2BeforeRequest**

```
procedure OnOAuth2BeforeRequest(TObject *Sender; TsgwSConnection *aConnection; TStringList *aHeaders;
    ref bool Cancel)
{
    if (DecodeGETFullPath(aHeaders) == "/Public.html")
    {
        Cancel = true;
    }
}
```

OAuth2 Server | Authorization Code Grant

Visão geral

O grant Authorization Code é o fluxo OAuth2 mais comum para aplicações do lado do servidor. O [TsgcHTTP_OAuth2_Server](#) fornece dois endpoints para tratar esse fluxo: um endpoint de Authorization que exibe uma página de login ao usuário e um endpoint de Token que troca o código de autorização por um access token. Este fluxo é adequado para aplicações web, aplicações desktop e aplicações móveis em que o cliente pode interagir com um navegador.

Fluxo

1. O cliente redireciona o usuário para `/sgc/oauth2/auth?response_type=code&client_id=...&redirect_uri=...&scope=...&state=...`
2. O servidor dispara **OnOAuth2BeforeDispatchPage** para permitir a personalização da página HTML de login.
3. O usuário envia as credenciais. O servidor dispara **OnOAuth2Authentication** para validar o nome de usuário e a senha.
4. Se autenticado, o servidor redireciona o usuário para o `redirect_uri` com `?code=...&state=...` anexado.
5. O cliente faz POST para `/sgc/oauth2/token` com `grant_type=authorization_code&code=...&redirect_uri=...&client_id=...&client_secret=...`
6. O servidor valida o código de autorização e as credenciais do cliente, dispara **OnOAuth2AfterAccessToken** e retorna uma resposta JSON contendo o `access_token`, `token_type`, `expires_in` e, opcionalmente, um `refresh_token`.

Configuração

Property	Descrição
<code>OAuth2Options.Authorization.Enabled</code>	Defina como True para habilitar o endpoint de autorização. Padrão: True.
<code>OAuth2Options.Authorization.URL</code>	O caminho da URL do endpoint de autorização. Padrão: <code>/sgc/oauth2/auth</code>
<code>OAuth2Options.Token.Enabled</code>	Defina como True para habilitar o endpoint de token. Padrão: True.
<code>OAuth2Options.Token.URL</code>	O caminho da URL do endpoint de token. Padrão: <code>/sgc/oauth2/token</code>
<code>OAuth2Options.PKCE</code>	Quando True, o servidor requer PKCE (<code>code_challenge</code> e <code>code_verifier</code>) para requisições de authorization code. Padrão: True.

App Registration

Antes de os clientes poderem usar o fluxo Authorization Code, você deve registrar a aplicação no servidor. Utilize `Apps.AddApp` e garanta que `AllowedGrantTypes` inclua `auth2Code`.

```
OAuth2Server->Apps->AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2Code);
```

Eventos

Event	Descrição
<code>OnOAuth2BeforeDispatchPage</code>	Disparado antes de a página HTML de login ser enviada ao usuário. Permite a personalização do conteúdo HTML.
<code>OnOAuth2Authentication</code>	Disparado quando o usuário submete as credenciais. Valide o nome de usuário e a senha aqui e defina <code>Authenticated</code> como True ou False.
<code>OnOAuth2AfterAccessToken</code>	Disparado depois que o servidor emite com sucesso um token de acesso. Útil para registro.
<code>OnOAuth2AfterValidateAccessToken</code>	Disparado quando um cliente faz uma requisição com um token válido e não expirado. Útil para registro e auditoria.

Exemplo

O exemplo a seguir mostra uma configuração completa de servidor com o grant Authorization Code, incluindo o registro do app e os handlers de eventos.

```
// Create and configure OAuth2 server
TsgcHTTP_OAuth2_Server *OAuth2Server = new TsgcHTTP_OAuth2_Server(this);

// Register an application
OAuth2Server->Apps->AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2Code);

// Attach to HTTP server
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2 = OAuth2Server;

void __fastcall TForm1::OAuth2ServerOAuth2Authentication(TObject *Sender,
    const UnicodeString User, const UnicodeString Password, bool &Authenticated)
{
    Authenticated = (User == "admin") && (Password == "secret");
}

void __fastcall TForm1::OAuth2ServerOAuth2AfterAccessToken(TObject *Sender,
    const UnicodeString Token, const UnicodeString RefreshToken, int ExpiresIn)
{
    Log("Access token issued: " + Token);
}
```

PKCE Support

Quando `OAuth2Options.PKCE` é definido como True (o padrão), o servidor impõe o Proof Key for Code Exchange. O cliente deve incluir um parâmetro `code_challenge` na requisição de autorização e um `code_verifier` correspondente na requisição de token. O servidor valida que o `code_verifier` corresponde ao `code_challenge` original antes de emitir um token.

O PKCE é recomendado para todos os clientes, especialmente clientes públicos (apps nativos e SPAs) que não podem armazenar com segurança um client secret.

OAuth2 Server | Client Credentials Grant

Visão geral

O grant Client Credentials é usado para comunicação máquina a máquina (M2M) onde nenhuma interação do usuário é necessária. O cliente se autentica diretamente com o [TsgcHTTP_OAuth2_Server](#) usando suas próprias credenciais (`client_id` e `client_secret`) e recebe um access token. Não há etapa de authorization code, nenhum redirecionamento e nenhuma página de login do usuário.

Este grant type é apropriado quando a aplicação cliente está agindo em seu próprio nome, em vez de em nome de um usuário específico.

Fluxo

1. O cliente faz POST para o endpoint de token com
`grant_type=client_credentials&client_id=...&client_secret=...&scope=...`
2. O servidor valida as credenciais do cliente em relação à aplicação registrada.
3. O servidor dispara **OnOAuth2AfterAccessToken** e retorna uma resposta JSON com `access_token`, `token_type` e `expires_in`.

Observe que nenhum evento `OnOAuth2Authentication` é disparado para este tipo de concessão, já que não há usuário a ser autenticado. O servidor valida o `client_id` e o `client_secret` contra a aplicação registrada automaticamente.

Configuração

Registre a aplicação com `AllowedGrantTypes` que inclua `auth2ClientCredentials`. Apenas o endpoint de token é necessário para este fluxo.

Property	Descrição
<code>OAuth2Options.Token.Enabled</code>	Defina como True para habilitar o endpoint de token. Padrão: True.
<code>OAuth2Options.Token.URL</code>	O caminho da URL do endpoint de token. Padrão: <code>/sgc/oauth2/token</code>

App Registration

```
OAuth2Server->Apps->AddApp("MyServiceApp", "",
    "service-client-id", "service-client-secret", 3600, false,
    TsgcOAuth2GrantTypes() << auth2ClientCredentials);
```

A `RedirectURI` pode ser deixada vazia, já que nenhum redirecionamento está envolvido. `RefreshToken` normalmente é definido como False porque o cliente pode solicitar um novo token a qualquer momento utilizando suas credenciais.

Eventos

Event	Descrição
<code>OnOAuth2AfterAccessToken</code>	Disparado depois que o servidor emite com sucesso um token de acesso. Útil para registro.

OnOAuth2AfterValidateAccessToken

Disparado quando um cliente faz uma requisição com um token válido. Útil para auditoria.

Exemplo

```
TsgcHTTP_OAuth2_Server *OAuth2Server = new TsgcHTTP_OAuth2_Server(this);
OAuth2Server->Apps->AddApp("BackendService", "",
    "service-client-id", "service-client-secret", 7200, false,
    TsgcOAuth2GrantTypes() << auth2ClientCredentials);
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2 = OAuth2Server;
void __fastcall TForm1::OAuth2ServerOAuth2AfterAccessToken(TObject *Sender,
    const UnicodeString Token, const UnicodeString RefreshToken, int ExpiresIn)
{
    Log("Service token issued: " + Token);
}
```

Quando Utilizar

Utilize o grant Client Credentials no servidor para:

- Serviços de backend e daemons que se autenticam sem interação do usuário.
- Comunicação de API servidor-a-servidor.
- Autenticação microsserviço-a-microsserviço.
- Processos automatizados, jobs agendados e operações em lote.

OAuth2 Server | Resource Owner Password Credentials Grant

Visão geral

O grant Resource Owner Password Credentials (ROPC) permite que o cliente envie o nome de usuário e a senha do usuário diretamente ao endpoint de token. O [TsgcHTTP_OAuth2_Server](#) valida as credenciais através do evento **OnOAuth2Authentication** e emite um access token se a autenticação for bem-sucedida.

Este fluxo não envolve um redirecionamento de navegador nem uma página de sign-in. O cliente coleta as credenciais diretamente do usuário e as submete ao servidor.

Aviso de Segurança

Este tipo de grant está obsoleto no OAuth 2.1. Ele só deve ser usado quando outros fluxos (como Authorization Code com PKCE) não forem viáveis, por exemplo em aplicações legadas ou clientes first-party altamente confiáveis. O grant Authorization Code com PKCE é a alternativa recomendada.

Fluxo

1. O cliente faz POST no endpoint de token com
`grant_type=password&username=...&password=...&client_id=...&client_secret=...&scope=...`
2. O servidor dispara **OnOAuth2Authentication** para validar o nome de usuário e a senha.
3. Se autenticado, o servidor dispara **OnOAuth2AfterAccessToken** e retorna uma resposta JSON com `access_token`, `token_type`, `expires_in` e, opcionalmente, um `refresh_token`.
4. Se a autenticação falhar, o servidor retorna uma resposta de erro.

Configuração

Registre a aplicação com `AllowedGrantTypes` que inclua `auth2ResourceOwnerPassword`.

Property	Descrição
<code>OAuth2Options.Token.Enabled</code>	Defina como True para habilitar o endpoint de token. Padrão: True.
<code>OAuth2Options.Token.URL</code>	O caminho da URL do endpoint de token. Padrão: <code>/sgc/oauth2/token</code>

App Registration

```
OAuth2Server->Apps->AddApp("MyApp", "",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2ResourceOwnerPassword);
```

Eventos

Event	Descrição

<code>OnOAuth2Authentication</code>	Disparado quando o servidor recebe o nome de usuário e a senha. Valide as credenciais aqui e defina <code>Authenticated</code> como True ou False.
<code>OnOAuth2AfterAccessToken</code>	Disparado depois que o servidor emite com sucesso um token de acesso. Útil para registro.
<code>OnOAuth2AfterValidateAccessToken</code>	Disparado quando um cliente faz uma requisição com um token válido.

Exemplo

```
TsgcHTTP_OAuth2_Server *OAuth2Server = new TsgcHTTP_OAuth2_Server(this);
OAuth2Server->Apps->AddApp("MyApp", "",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2ResourceOwnerPassword);
Server->Authentication->Enabled = true;
Server->Authentication->OAuth2 = OAuth2Server;
void __fastcall TForm1::OAuth2ServerOAuth2Authentication(TObject *Sender,
    const UnicodeString User, const UnicodeString Password, bool &Authenticated)
{
    Authenticated = ValidateUser(User, Password);
}
void __fastcall TForm1::OAuth2ServerOAuth2AfterAccessToken(TObject *Sender,
    const UnicodeString Token, const UnicodeString RefreshToken, int ExpiresIn)
{
    Log("Token issued for user: " + Token);
}
```

Quando Utilizar

Utilize o grant Resource Owner Password Credentials apenas quando:

- O cliente é uma aplicação first-party altamente confiável.
- Outros tipos de grant (Authorization Code com PKCE) não são viáveis.
- Migração de aplicações legadas que já coletam credenciais do usuário diretamente.

Para novas aplicações, sempre prefira o grant [Authorization Code](#) com PKCE.

OAuth2 Server | Refresh Token Grant

Visão geral

O grant Refresh Token permite que um cliente troque um refresh token emitido anteriormente por um novo access token sem exigir interação do usuário. O [TsgcHTTP_OAuth2_Server](#) valida o refresh token e emite um novo access token (e, opcionalmente, um novo refresh token).

Isto é útil para sessões de longa duração onde o access token tem uma expiração curta, mas o cliente precisa de acesso contínuo a recursos protegidos.

Fluxo

1. O cliente faz um POST ao endpoint de token com
grant_type=refresh_token&refresh_token=...&client_id=...&client_secret=...&scope=...
2. O servidor valida o token de atualização contra seus tokens armazenados.
3. O servidor dispara **OnOAuth2AfterRefreshToken** e retorna uma resposta JSON com um novo access_token, token_type, expires_in e, opcionalmente, um novo refresh_token.

Configuração

Os refresh tokens são habilitados por aplicação ao registrar o app. Defina o parâmetro RefreshToken como True em Apps.AddApp.

Property	Descrição
<code>OAuth2Options.Token.Enabled</code>	Defina como True para habilitar o endpoint de token. Padrão: True.
<code>OAuth2Options.Token.URL</code>	O caminho da URL do endpoint de token. Padrão: /sgc/oauth2/token
<code>RefreshToken</code> (AddApp parâmetro)	Defina como True para emitir tokens de atualização para esta aplicação. Padrão: True.

App Registration

Ao registrar o aplicativo, defina o parâmetro RefreshToken como True para que o servidor emita tokens de atualização junto com os tokens de acesso.

```
// RefreshToken = true (6th parameter)
OAuth2Server->Apps->AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2Code);
```

Eventos

Event	Descrição
<code>OnOAuth2AfterRefreshToken</code>	Disparado depois que o servidor emite com sucesso um novo token de acesso a partir de um refresh token. Útil para registrar a rotação de tokens.
<code>OnOAuth2AfterAccessToken</code>	Disparado após o novo token de acesso ser emitido.

Exemplo

```
TsgcHTTP_OAuth2_Server *OAuth2Server = new TsgcHTTP_OAuth2_Server(this);
OAuth2Server->Apps->AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2Code);
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2 = OAuth2Server;
void __fastcall TForm1::OAuth2ServerOAuth2AfterRefreshToken(TObject *Sender,
    const UnicodeString OldToken, const UnicodeString NewToken,
    const UnicodeString NewRefreshToken, int ExpiresIn)
{
    Log("Token refreshed. New token: " + NewToken);
}
```

Rotação de Token

Quando um token de atualização é utilizado, o servidor pode emitir um novo token de atualização junto com o novo token de acesso. O token de atualização antigo é invalidado. Isto é conhecido como rotação de token de atualização e ajuda a evitar ataques de repetição de token.

Recuperando tokens após reinicialização

Se o servidor for reiniciado enquanto os tokens ainda forem válidos, você pode recuperá-los usando `Apps.AddToken` antes de iniciar o servidor. Veja [OAuth2 Recover Access Tokens](#) para detalhes.

OAuth2 Server | Device Authorization Grant (RFC 8628)

Visão geral

O Device Authorization Grant é projetado para dispositivos com entrada limitada (smart TVs, dispositivos IoT, ferramentas de CLI) que não conseguem exibir facilmente um navegador ou aceitar entrada de teclado. O [TsgcHTTP_OAuth2_Server](#) fornece um endpoint de autorização de dispositivo e uma página de verificação para que os usuários possam autorizar o dispositivo a partir de um dispositivo secundário, como um telefone ou computador.

Fluxo

1. O dispositivo envia via POST para `/sgc/oauth2/device` com `client_id=...&scope=...`
2. O servidor dispara **OnOAuth2DeviceAuthorization** e retorna uma resposta JSON com `device_code`, `user_code`, `verification_uri`, `expires_in` e `interval`.
3. O dispositivo exibe o `user_code` e a `verification_uri` ao usuário (por exemplo, "Acesse <https://example.com/sgc/oauth2/device/verify> e digite o código: ABCD-1234").
4. O usuário navega até a URL de verificação em um dispositivo secundário e digita o `user_code`.
5. O servidor dispara **OnOAuth2DeviceCodeVerification** para validar o `user_code` e autorizar o dispositivo.
6. Enquanto isso, o dispositivo consulta o endpoint de token com `grant_type=urn:ietf:params:oauth:grant-type:device_code&device_code=...&client_id=...`
7. O servidor retorna `authorization_pending` até que o usuário autorize. Uma vez autorizado, ele retorna o token de acesso.

Configuração

Property	Descrição
<code>OAuth2Options.DeviceAuthorization.Enabled</code>	Defina como True para habilitar o endpoint de autorização de dispositivo. Padrão: False.
<code>OAuth2Options.DeviceAuthorization.URL</code>	O caminho da URL do endpoint de autorização de dispositivo. Padrão: <code>/sgc/oauth2/device</code>
<code>OAuth2Options.DeviceAuthorization.VerificationURL</code>	O caminho da URL da página de verificação onde o usuário insere o código. Padrão: <code>/sgc/oauth2/device/verify</code>
<code>OAuth2Options.DeviceAuthorization.ExpiresIn</code>	O lifetime em segundos do device code. Padrão: 600 (10 minutos).
<code>OAuth2Options.DeviceAuthorization.Interval</code>	O intervalo mínimo de polling, em segundos, que o dispositivo deve usar. Padrão: 5.

Eventos

Event	Descrição
<code>OnOAuth2DeviceAuthorization</code>	Disparado quando um dispositivo solicita um device code. Permite personalizar a resposta ou registrar a requisição.
<code>OnOAuth2DeviceCodeVerification</code>	Disparado quando um usuário envia um código de verificação na página de verificação de dispositivo. Valide o <code>user_code</code> e autorize ou negue o dispositivo.
<code>OnOAuth2AfterAccessToken</code>	Disparado depois que o servidor emite o access token ao dispositivo.

Exemplo

```
TsgcHTTP_OAuth2_Server *OAuth2Server = new TsgcHTTP_OAuth2_Server(this);
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,
    TsgcOAuth2GrantTypes() << auth2Code);
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2 = OAuth2Server;
void __fastcall TForm1::OAuth2ServerOAuth2DeviceCodeVerification(TObject *Sender,
    const UnicodeString UserCode, bool &Authorized)
{
    Authorized = true;
}
void __fastcall TForm1::OAuth2ServerOAuth2AfterAccessToken(TObject *Sender,
    const UnicodeString Token, const UnicodeString RefreshToken, int ExpiresIn)
{
    Log("Device authorized. Token: " + Token);
}
```

Comportamento de Polling

Enquanto o usuário ainda não autorizou o dispositivo, o dispositivo faz polling no endpoint de token no intervalo configurado. O servidor responde com:

- `authorization_pending` -- o usuário ainda não concluiu a autorização.
- `slow_down` -- o dispositivo está fazendo polling com frequência excessiva. O dispositivo deve aumentar o intervalo em 5 segundos.
- `expired_token` -- o código do dispositivo expirou. O dispositivo deve reiniciar o fluxo.
- `access_denied` -- o usuário negou a autorização.

Depois que o usuário autoriza o dispositivo, o próximo poll retorna o access token.

OAuth2 Server | Token Revocation (RFC 7009)

Visão geral

A revogação de token permite que os clientes notifiquem o [TsgcHTTP_OAuth2_Server](#) de que um token de acesso ou de atualização previamente emitido não é mais necessário. O servidor invalida o token para que ele não possa mais ser utilizado para acessar recursos protegidos.

Isso é útil quando um usuário faz logout, quando uma aplicação é desinstalada ou quando um token pode ter sido comprometido.

Endpoint

POST para `/sgc/oauth2/revoke` com os seguintes parâmetros:

Parameter	Descrição
<code>token</code>	O token a revogar (obrigatório).
<code>token_type_hint</code>	Hint opcional sobre o tipo de token: <code>access_token</code> ou <code>refresh_token</code> .

Behavior

De acordo com a [RFC 7009](#), o servidor sempre retorna HTTP 200 OK independentemente de o token ter sido encontrado ou revogado com sucesso. Isso evita o vazamento da existência de tokens, um cliente não consegue determinar se um token era válido observando o status da resposta.

Quando um token de atualização é revogado, o token de acesso associado também pode ser invalidado (dependendo da implementação). Quando um token de acesso é revogado, o token de atualização associado permanece válido a menos que seja explicitamente revogado.

Configuração

Property	Descrição
<code>OAuth2Options.Revocation.Enabled</code>	Defina como True para habilitar o endpoint de revogação. Padrão: False.
<code>OAuth2Options.Revocation.URL</code>	O caminho da URL do endpoint de revogação. Padrão: <code>/sgc/oauth2/revoke</code>

Eventos

Event	Descrição
<code>OnOAuth2AfterRevokeToken</code>	Disparado após uma tentativa de revogação de token. Fornece o valor do token, o <code>token_type_hint</code> e um parâmetro <code>Revoked</code> indicando se o

token foi invalidado com sucesso. Útil para registrar a atividade de revogação.

Exemplo

```
TsgcHTTP_OAuth2_Server *OAuth2Server = new TsgcHTTP_OAuth2_Server(this);

OAuth2Server->OAuth2Options->Revocation->Enabled = true;

OAuth2Server->Apps->AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2Code);

Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2 = OAuth2Server;

void __fastcall TForm1::OAuth2ServerOAuth2AfterRevokeToken(TObject *Sender,
    const UnicodeString Token, const UnicodeString TokenTypeHint, bool &Revoked)
{
    Log("Token revoked: " + Token + " (type: " + TokenTypeHint + ")");
    Revoked = true;
}
```

Exemplo de Requisição do Cliente

Um cliente revoga um token enviando uma requisição POST ao endpoint de revogação:

```
// 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)

Visão geral

A introspecção de token permite que os resource servers consultem o [TsgcHTTP_OAuth2_Server](#) para determinar o estado ativo e os metadados de um token de acesso ou refresh token. Isso é útil quando um resource server precisa validar um token sem ter que decodificá-lo localmente.

Endpoint

Envie via POST para `/sgc/oauth2/introspect` com os seguintes parâmetros:

Parameter	Descrição
<code>token</code>	O token a inspecionar (introspect) (obrigatório).
<code>token_type_hint</code>	Hint opcional sobre o tipo de token: <code>access_token</code> ou <code>refresh_token</code> .

Resposta

O endpoint de introspecção retorna um objeto JSON com os seguintes campos:

Field	Descrição
<code>active</code>	Boolean. True se o token estiver atualmente ativo (válido e não expirado ou revogado).
<code>scope</code>	O escopo associado ao token.
<code>client_id</code>	O identificador de cliente da aplicação que solicitou o token.
<code>token_type</code>	O tipo do token (por exemplo, Bearer).
<code>exp</code>	O tempo de expiração do token (timestamp Unix).
<code>username</code>	O nome de usuário associado ao token (se aplicável).

Se o token for inválido, expirado ou revogado, a resposta contém apenas `{"active": false}`.

Configuração

Property	Descrição
<code>OAuth2Options.Introspection.Enabled</code>	Defina como True para habilitar o endpoint de introspecção. Padrão: False.

`OAuth2Options.Introspection.URL`

O caminho da URL do endpoint de introspection. Padrão: /sgc/oauth2/introspect

Eventos

Event	Descrição
<code>OnOAuth2AfterIntrospectToken</code>	Disparado após uma requisição de introspection de token. Fornece o valor do token e um parâmetro <code>IsActive</code> que pode ser modificado para sobrescrever o resultado de validação padrão. Útil para lógica de validação personalizada e auditoria.

Exemplo

```
TsgcHTTP_OAuth2_Server *OAuth2Server = new TsgcHTTP_OAuth2_Server(this);
OAuth2Server->OAuth2Options->Introspection->Enabled = true;
OAuth2Server->Apps->AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2Code);
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2 = OAuth2Server;

void __fastcall TForm1::OAuth2ServerOAuth2AfterIntrospectToken(TObject *Sender,
    const UnicodeString Token, bool &IsActive)
{
    Log("Token introspected: " + Token + " Active: " + BoolToStr(IsActive));
}
```

Exemplo de Requisição ao Resource Server

Um resource server valida um token enviando uma requisição POST ao endpoint de introspecção:

```
// Resource server: introspect a token
UnicodeString 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 | DPoP Validation (RFC 9449)

Visão geral

DPoP (Demonstrating Proof of Possession) conforme a [RFC 9449](#) permite que o [TsgcHTTP_OAuth2_Server](#) exija tokens vinculados ao remetente. Quando o DPoP está habilitado, o servidor valida JWTs de prova DPoP, vincula tokens ao thumbprint JWK do cliente e retorna `token_type: DPoP` em vez de `Bearer`.

Isso garante que os tokens de acesso só possam ser utilizados pelo cliente que originalmente os obteve, prevenindo roubo e ataques de repetição de token.

Como funciona

1. O cliente inclui um cabeçalho `DPoP` com uma prova JWT assinada nas requisições de token.
2. O servidor valida o JWT de prova:
 - `typ` deve ser `dpop+jwt`
 - O Algoritmo (`alg`) deve ser assimétrico (por exemplo, ES256, RS256)
 - `htm` (método HTTP) e `htu` (URI HTTP) devem corresponder à requisição
 - `iat` (issued at) deve ser recente (verificação de frescor)
 - A assinatura é verificada contra a JWK incorporada no cabeçalho de prova
3. O servidor extrai o thumbprint JWK e o vincula ao token emitido.
4. A resposta do token inclui `token_type: DPoP` em vez de `Bearer`.
5. Em requisições de recurso subsequentes, o servidor valida a prova DPoP, incluindo a `claim ath` (hash do token de acesso) para garantir que a prova esteja vinculada ao token correto.

Configuração

Property	Descrição
<code>OAuth2Options.DPoP</code>	Boolean. Defina como <code>True</code> para habilitar a validação de tokens vinculados a DPoP. Padrão: <code>False</code> .

Verificação de Assinatura

O servidor reconstrói a chave pública EC ou RSA a partir do JWK incluído no cabeçalho de prova DPoP e verifica a assinatura do JWT. Apenas algoritmos assimétricos são aceitos (ES256, ES384, ES512, RS256, RS384, RS512, PS256, PS384, PS512). Algoritmos simétricos (HS256, etc.) são rejeitados.

Suporte a Nonce

O servidor pode emitir um cabeçalho `DPoP-Nonce` nas respostas de token. Quando presente, o cliente deve incluir o valor do nonce na `claim nonce` dos JWTs de prova DPoP subsequentes. Isso fornece uma camada adicional de proteção contra replay.

Eventos

Event	Descrição

OnOAuth2ValidateDPoP

Disparado depois que a validação padrão de DPoP é concluída. Permite lógica de validação personalizada, como verificar claims adicionais ou impor uma política. Defina **Valid** como False para rejeitar a prova.

Exemplo

```
TsgcHTTP_OAuth2_Server *OAuth2Server = new TsgcHTTP_OAuth2_Server(this);
OAuth2Server->OAuth2Options->DPoP = true;
OAuth2Server->Apps->AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2Code);
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2 = OAuth2Server;
void __fastcall TForm1::OAuth2ServerOAuth2ValidateDPoP(TObject *Sender,
    const UnicodeString DPoPProof, const UnicodeString AccessToken,
    const UnicodeString Method, const UnicodeString URL, bool &Valid)
{
    Log("DPoP proof validated for: " + Method + " " + URL);
    Valid = true;
}
```

Invalid Proof Rejection

Quando a validação da prova DPoP falha, o servidor retorna HTTP 400 com uma resposta de erro:

```
{"error": "invalid_dpop_proof", "error_description": "..."}

```

As razões comuns para rejeição incluem:

- Cabeçalho DPoP ausente ou malformatado.
- Assinatura JWT inválida.
- Algoritmo simétrico utilizado (somente algoritmos assimétricos são permitidos).
- Claims `htm` ou `htu` incompatíveis.
- Claim `iat` expirada ou no futuro.
- `ath` (hash do access token) incompatível em requisições de recurso.
- `nonce` inválido ou ausente quando o servidor o exige.

TsgcHTTP_OAuth2_Server_Provider

Este componente permite que você integre Provedores OAuth2 Externos (como Azure AD, Google, Facebook).

Introdução

Este componente permite que você integre Provedores OAuth2 Externos (como Azure AD, Google, Facebook...) no seu componente servidor (como um servidor HTTP), para que um usuário possa fazer login usando as credenciais do Azure AD e, se a autenticação for bem-sucedida, o servidor HTTP pode fornecer acesso a recursos protegidos.

Os componentes de servidor possuem uma propriedade chamada `Authorization.OAuth.OAuth2Provider`, onde você pode atribuir uma instância de `TsgcHTTP_OAuth2_Server_Provider`, então se a `Authentication` estiver habilitada e a propriedade `OAuth2Provider` estiver vinculada ao `OAuth2 Provider Server Component`, os `WebSocket` e `HTTP Requests` exigem um `Cookie / Bearer Token` para serem processados; caso contrário, a conexão será fechada automaticamente.

```
TsgcHTTP_OAuth2_Server_Provider *OAuth2Provider = new TsgcHTTP_OAuth2_Server_Provider(this);
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2Provider = OAuth2Provider;
```

Registrar Provedor OAuth2

Antes de o servidor ser iniciado, você deve configurar os OAuth2 Providers que o servidor utilizará para autenticar. Utilize o método **RegisterProvider** para configurar os OAuth2 Providers, este método tem os seguintes parâmetros:

- **Name:** é o nome do provedor, pode ser qualquer nome, é apenas para identificar o provedor posteriormente.
- **ClientId:** é o client Id público, este valor é fornecido pelo Provedor OAuth2.
- **ClientSecret:** é o client secret privado (deve ser mantido confidencial); este valor é fornecido pelo Provedor OAuth2.
- **AuthorizeURL:** é a URL para onde o cliente OAuth2 redirecionará para fazer login (a conexão usa um navegador web).
- **TokenURL:** é a URL que o servidor utilizará para validar o token fornecido após uma autorização bem-sucedida (a conexão é de servidor para servidor).
- **Scope:** é o valor do(s) escopo(s).
- **URL:** é a URL do Servidor HTTP que será utilizada para redirecionar à URL de Autorização.
- **CallbackURL:** é a URL configurada no Provedor OAuth2 que processará a resposta enviada pelo servidor OAuth2 após uma Autorização bem-sucedida.

Exemplo: para configurar o Azure AD, é necessário um tenant-id que é adicionado às URLs OAuth2, ClientId, ClientSecret, Scope e uma 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'
);
```

Para excluir um Provider existente, use o método **UnRegisterProvider**.

Referência

- Propriedades — 2 propriedades disponíveis no componente.
- Métodos — 15 métodos públicos.
- Eventos — 4 eventos que você pode tratar.

Guides

- **QuickStart**
 - [OAuth2 Provider Azure AD](#)
- **Authenticate**
 - [OAuth2 Provider Private Endpoints](#)
 - [OAuth2 Provider Authentication](#)
 - [OAuth2 Provider Requests](#)

OAuth2 Provider | Azure AD

O Azure AD utiliza as seguintes URLs de Autorização OAuth2

Authorization: `https://login.microsoftonline.com/<tenant-id>/oauth2/v2.0/authorize`

Token: `https://login.microsoftonline.com/<tenant-id>/oauth2/v2.0/token`

O **<tenant-id>** deve ser substituído pelos seus próprios valores.

Ao criar a configuração OAuth2, você deve configurar uma url de callback do servidor. Esta url será usada pela Azure para enviar uma resposta ao seu servidor após uma autorização bem-sucedida.

Exemplo: encontre abaixo um exemplo simples de como registrar o provedor Azure AD.

Valores fornecidos pelo 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`

Como Registrar o 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

Toda vez que o Servidor recebe uma Requisição HTTP, o evento **OnOAuth2IsPrivateEndpoint** é chamado para perguntar se o Endpoint é privado ou não. Por padrão, não é privado.

```
void OnOAuth2IsPrivateEndpoint(TObject *Sender, const string aEndpoint, ref bool IsPrivate)
{
    if (aEndpoint == "/private")
    {
        IsPrivate = True;
    }
}
```

OAuth2 Provider | Authentication

O Componente Servidor de Provedor OAuth2 permite que você autentique usando um provedor OAuth2 externo (como Azure AD ou Google) para acessar os recursos protegidos do seu servidor. **Exemplo:** você pode configurar seu servidor HTTP para permitir que os usuários façam login usando credenciais Azure; se o login for bem-sucedido, esses usuários terão permissão para acessar os recursos protegidos do seu servidor.

O processo de autenticação é feito no lado do servidor e os tokens OAuth2 não são compartilhados com os clientes. Isso significa que quando o usuário faz login usando o Azure, por exemplo, se a autenticação for bem-sucedida, o Azure retorna um Access Token que permite enviar requisições ao servidor Azure para obter algumas informações (dependendo do scope) sobre o perfil do usuário, emails... Este Access Token **NÃO É COMPARTILHADO** com o cliente (por exemplo, um navegador web); em vez de retornar o Access Token ao cliente, o servidor cria um ID aleatório que é vinculado internamente ao Access Token, de modo que toda vez que o Cliente (Navegador Web) quiser fazer uma chamada ao servidor OAuth2, usa o ID público e o servidor usa este ID para obter o Access Token OAuth2 para fazer o proxy das requisições HTTP.

Veja abaixo um exemplo de como a Autenticação OAuth2 funciona. O exemplo utilizará a configuração do Azure AD descrita no link a seguir [OAuth2 Provider Azure AD](#).

Inicie o Servidor

O servidor começa a escutar em localhost e na porta 443. O Servidor HTTP sgcWebSockets é vinculado ao Componente Servidor Provedor OAuth2 e a propriedade Authentication está habilitada.

Antes de o servidor ser iniciado, o Provedor OAuth2 da Azure é registrado utilizando a seguinte chamada de método.

```
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'
);
```

Logins de usuário

O usuário abre um novo navegador web e vai para o endpoint '/login'.

O servidor detecta que o endpoint '/login' é usado para fazer login usando o provedor Azure, então redireciona para

<https://login.microsoftonline.com/a0ca2055-5dd1-467f-bf13-291f6fd715c6/oauth2/v2.0/authorize>

E o Fluxo de autenticação OAuth2 Inicia.

OAuth2 Authentication

O usuário é redirecionado ao Endpoint de Autenticação do Servidor OAuth2; agora ele deve fazer login usando as credenciais e aceitar os termos da Aplicação OAuth2.

Se a autorização for bem-sucedida, o Azure AD envia um Code para a url

<https://localhost/callback>

Validar o OAuth2 Code

Agora, o servidor recebeu um código do Azure e fará uma conexão interna ao Azure (de servidor a servidor) para validar se esse token está correto (e evitar que alguém tente invadir o servidor).

O servidor se conecta a

`https://login.microsoftonline.com/a0ca2055-5dd1-467f-bf13-291f6fd715c6/oauth2/v2.0/token`

Ao passar alguns parâmetros como o código recebido e o `clientsecret`, se a validação for bem-sucedida, o Azure retorna o Access Token que pode ser utilizado para acessar os Azure Protected Resources, como ler o perfil, e-mail...

Access Token Bem-Sucedido

Quando o servidor recebe um `AccessToken` bem-sucedido, o evento **OnOAuth2ProviderTokenValid** é chamado, portanto aqui você pode configurar como o `AccessToken` é armazenado (se for), acessando a classe de parâmetro `TsgcHTTPOAuth2ProviderToken`

AccessToken: é o Token OAuth2 retornado pelo Azure

ID: é o identificador público armazenado como um cookie.

Neste evento você pode configurar o que fazer após uma autenticação bem-sucedida, exemplo: se você quiser redirecionar o usuário para a url privada, utilize o seguinte

`Response.Redirect.URL := 'https://localhost/private';`

Enviar Requisições para a Azure

Agora, você pode enviar requisições ao servidor Azure usando o Public ID armazenado como um cookie. Exemplo: se você quiser ler os dados do perfil, utilize o método a seguir.

`Get('ID', 'https://graph.microsoft.com/v1.0/me');`

Onde ID é o identificador de ID público.

OAuth2 Provider | Requests

Uma vez que a Autenticação tenha sido bem-sucedida, você pode enviar requisições ao Servidor Protegido por OAuth2 usando o Public ID Token armazenado como um cookie.

O Componente Servidor Provedor OAuth2 tem vários métodos para enviar Requisições HTTP: GET, POST, DELETE...

Você pode passar o Token como parâmetro ou passar a classe RequestInfo se estiver utilizando os componentes Indy Server.

```
void OnCommandGet(TidContext *AContext, TidHttpRequestInfo *ARequestInfo, TidHttpResponseInfo *AResponseInfo)
{
    if (ARequestInfo->Document == "/private"
    {
        // return OAuth2 profile data
        AResponseInfo->ContentText = OAuth2Provider->Get(ARequestInfo, "https://graph.microsoft.com/v1.0/me");
        AResponseInfo->ContentType = "application/json";
        AResponseInfo->ResponseNo = 200;
    }
    else
    {
        AResponseInfo->ResponseNo = 404;
    }
}
```

HTTP | JWT

O JWT (JSON Web Token) normalmente consiste em header + payload + signature.

Header

Contém as informações de metadados sobre o JWT.

- **alg**: o algoritmo usado para assinar o token.
- **typ**: o tipo do token, sempre JWT.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Você pode encontrar mais cabeçalhos, mas os anteriores estarão sempre presentes.

Payload

O payload contém as claims do JWT. Os cabeçalhos padrão são os seguintes:

- **iss**: o emissor, a entidade que gera e emite o JWT.
- **sub**: o subject, a entidade identificada por este token.
- **aud**: a audience, o público-alvo deste JWT.
- **exp**: a expiração, o timestamp em formato UNIX após o qual o token não deve ser aceito.
- **iat**: issued at, especifica a data em que o token foi emitido.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

Assinatura

A assinatura é criada utilizando o Header Codificado, o Payload Codificado, um Secret e um Algoritmo Criptográfico.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MzkwMj0
```

```
.dIyfq.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

Algoritmos suportados

Os algoritmos a seguir são suportados tanto pelos componentes JWT de Cliente quanto de Servidor.

- HS256
- HS384
- HS512
- RS256
- RS384
- RS512
- ES256
- ES384
- ES512

As bibliotecas **OpenSSL** são necessárias para assinar e verificar o JWT.

Componentes

- **TsgcHTTP_JWT_Client**: Cliente JWT que permite codificar e assinar JWT e enviá-lo como um Authorization Header nos protocolos **HTTP** e **WebSocket**.
- **TsgcHTTP_JWT_Server**: servidor JWT que permite decodificar e validar JWT recebidos como um Authorization Header nos protocolos **HTTP** e **WebSocket**.

** Os Componentes JWT exigem pelo menos a versão 10.6.0.5169 do Indy ou a Edição Enterprise do sgcWebSockets.*

TsgcHTTP_JWT_Client

Configure quais bibliotecas openSSL você usará ao usar o cliente JWT.

Introdução

O componente TsgcHTTP_JWT_Client permite que você codifique e assine JWT Tokens, anexado a um [WebSocket Client](#) ou [HTTP/2 client](#), o token será enviado automaticamente como um Authorization Bearer Token Header.

Configuração

Você pode configurar os valores do JWT nas propriedades **JWTOptions**; há 2 propriedades principais: **Header** e **Payload**, basta definir os valores de cada propriedade obrigatória.

Se a Assinatura for criptografada usando uma Chave Privada (algoritmos RS e ES), defina o valor na propriedade **PrivateKey** do Algorithm.

Se a Assinatura for criptografada usando um Secret (algoritmos HS), defina o valor na propriedade **Secret** do Algorithm.

OpenSSL Options

Configure quais bibliotecas openSSL você usará ao usar o cliente JWT.

OpenSSL_Options: configuração das bibliotecas openSSL.

APIVersion: permite definir qual API OpenSSL será usada.

oslAPI_1_0: usa a API 1.0 do OpenSSL, é a mais recente suportada pelo Indy

oslAPI_1_1: usa a API 1.1 do OpenSSL, requer nossa biblioteca Indy personalizada e permite que você use bibliotecas OpenSSL 1.1.1 (com suporte a TLS 1.3).

oslAPI_3_0: usa a API 3.0 do OpenSSL, requer nossa biblioteca Indy personalizada e permite que você use bibliotecas OpenSSL 3.0.0 (com suporte a TLS 1.3).

LibPath: aqui você pode configurar onde estão localizadas as bibliotecas openSSL

oslpNone: este é o padrão, as bibliotecas openSSL devem estar na mesma pasta onde está o binário ou em um caminho conhecido.

oslpDefaultFolder: define automaticamente o caminho openSSL onde as bibliotecas devem estar localizadas para todas as personalidades do IDE.

oslpCustomFolder: se esta for a opção selecionada, defina o caminho completo na propriedade LibPathCustom.

LibPathCustom: quando LibPath = oslpCustomFolder, defina aqui o caminho completo onde estão localizadas as bibliotecas openSSL.

UnixSymLinks: habilita ou desabilita o carregamento de SymLinks em sistemas Unix (por padrão está habilitado, exceto no OSX64):

oslsSymLinksDefault: por padrão estão habilitados, exceto sob OSX64 (após o MacOS Monterey falha ao tentar carregar a biblioteca sem versão.).

oslsSymLinksLoadFirst: Carrega os SymLinks e faz isso antes de tentar carregar as bibliotecas de versão.

oslsSymLinksLoad: Carrega os SymLinks após tentar carregar as bibliotecas de versão.

oslsSymLinksDontLoad: não carrega os SymLinks.

Custom Headers

As propriedades Header e Payload contêm os cabeçalhos mais comuns usados para gerar um JWT, mas você pode adicionar mais cabeçalhos chamando o método **AddKeyValue** e passando a Key e o Value como parâmetros.

Exemplo: se você quiser adicionar um novo registro no Cabeçalho JWT com o seu nome, utilize o método a seguir


```
Header->AddKeyValue("name", "John Smith");
```

Após configurar as propriedades, para gerar o JWT, basta chamar o método **Sign** e ele retornará o valor do JWT.

WebSocket Client e JWT

[TsgcWebSocketClient](#) permite o uso de JWT ao conectar-se a servidores WebSocket; basta criar um novo cliente JWT e atribuí-lo à propriedade **Authentication.Token.JWT**.

```
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;
```

Clientes HTTP e JWT

[TsgcHTTP2Client](#) e [TsgcHTTP1Client](#) permitem o uso de JWT ao conectar-se a servidores HTTP/2; basta criar um novo JWT client e atribuí-lo à propriedade **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");
```

Expiração

O Authorization Token pode ser **recriado a cada vez** que você envia uma requisição HTTP usando um cliente HTTP ou pode ser **reutilizado várias vezes até expirar**.

Exemplo: chamar o APNs da Apple utilizando Tokens requer que o token seja reutilizado por pelo menos 20 minutos e no máximo 1 hora. Utilize a propriedade **RefreshTokenAfter** para definir os segundos quando o token expirará, por exemplo após 30 minutos.

```
RefreshTokenAfter = 60 * 40.
```

Criar Assinatura JWT

Você pode **criar JWT Signatures manualmente** para utilizar em aplicações que não fazem uso do protocolo WebSocket ou HTTP, ou se você estiver utilizando componentes de aplicações de terceiros e você só precisar do JWT Token.

Para obter a JWT Signature, basta criar uma nova instância do JWT Client e preencher as propriedades manualmente. Quando todas as propriedades estiverem definidas, chame o método **Sign** e ele retornará o JWT Token.

```
TsgcHTTP_JWT_Client oJWT = new TsgcHTTP_JWT_Client(this);
// ... 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
ShowMessage(oJWT->Sign());
```

Referência

- Propriedades — 2 propriedades disponíveis no componente.
- Métodos — 2 métodos públicos.
- Eventos — 0 eventos que você pode tratar.

TsgcHTTP_JWT_Server

Componente *TsgcHTTP_JWT_Server*.

Introdução

O componente *TsgcHTTP_JWT_Server* permite que você **decodifique** e **valide** tokens JWT recebidos no **Web-Socket Handshake** ao utilizar o protocolo WebSocket ou como **HTTP Header** ao utilizar o protocolo HTTP.

Configuração

Você pode configurar as seguintes propriedades na propriedade **JWTOptions** do componente:

Se a assinatura for validada utilizando uma Chave Pública (algoritmos RS e ES), defina o valor na propriedade **PublicKey** do **Algorithm**.

Se a Signature for validada utilizando um Secret (algoritmos HS), defina o valor na propriedade **Secret** do **Algorithm**.

Para validar tokens JWT, basta **anexar uma instância de *TsgcHTTP_JWT_Server*** à propriedade **Authentication.JWT.JWT** do Servidor WebSocket/HTTP.

```
TsgcWebSocketHTTPServer oServer = new TsgcWebSocketHTTPServer();
oServer->Port = 80;
TsgcHTTP_JWT_Server oJWT = new TsgcHTTP_JWT_Server();
oJWT->JWTOptions->Algorithms->RS->PublicKey->Text = "public key here";
oServer->Authorization->Enabled = true;
oServer->Authorization->JWT->JWT = oJWT;
oServer->Active = true;
```

Checks permite que você habilite algumas verificações no Payload do JWT; por padrão, verifica se as datas de emissão são válidas.

Referência

- Propriedades — 2 propriedades disponíveis no componente.
- Métodos — 3 métodos públicos.
- Eventos — 7 eventos que você pode tratar.

WebAuthn

WebAuthn (Web Authentication) é um padrão web desenvolvido pelo World Wide Web Consortium (W3C) e pela FIDO Alliance para habilitar uma autenticação **segura e sem senha na web**. Faz parte do framework FIDO2 mais amplo e visa **reduzir a dependência de senhas tradicionais**, que muitas vezes são vulneráveis a phishing, credential stuffing e outros ataques.

Em sua essência, **o WebAuthn permite que os usuários se autentiquem utilizando criptografia de chave pública. Em vez de um nome de usuário e senha, os usuários registram um par único de chaves pública-privada em uma aplicação web** (a Relying Party). A chave privada é armazenada com segurança em um authenticator — como uma chave de segurança de hardware, um smartphone ou um dispositivo biométrico integrado — enquanto a chave pública é armazenada no servidor.

Durante a autenticação, o servidor emite um challenge que deve ser assinado pela chave privada do usuário. O challenge assinado é retornado e verificado usando a chave pública armazenada, garantindo tanto a integridade quanto a origem da resposta. Essa abordagem impede que as credenciais sejam interceptadas ou reutilizadas.

O WebAuthn suporta uma variedade de autenticadores e dispositivos, tornando-o flexível tanto para desenvolvedores quanto para usuários. Ele também habilita autenticação multifator (MFA) quando combinado com outros fatores como PINs ou biometria, melhorando significativamente a segurança sem sacrificar a usabilidade.

Server

- **TsgcWSAPIServer_WebAuthn**: O componente fornece uma solução simples, mas poderosa, para implementar o servidor WebAuthn Relying Party, habilitando autenticação sem senha em sua aplicação web. O servidor passou no **FIDO Conformance Test** completo utilizando a ferramenta Conformance Self-Validation Testing.

Client

- **Javascript Client**: atua como a ponte entre o usuário, o navegador e o autenticador durante o registro e a autenticação

TsgcWSAPIServer_WebAuthn

Apenas o OpenSSL 3.

Introdução

O componente **TsgcWSAPIServer_WebAuthn** fornece uma solução simples, porém poderosa, para implementar o servidor WebAuthn Relying Party, habilitando autenticação sem senha em sua aplicação web. Uma aplicação WebAuthn consiste em um servidor WebAuthn que trata do registro e da autenticação no lado do servidor, e uma aplicação do lado do cliente que geralmente é uma aplicação javascript.

O WebAuthn requer o uso de conexões seguras (SSL/TLS), portanto as bibliotecas [OpenSSL](#) devem ser implantadas e configuradas com o servidor.

Apenas a OpenSSL 3.0.0+ API é suportada, portanto versões anteriores do OpenSSL podem não funcionar.

Configuração

O **TsgcWSAPIServer_WebAuthn** deve ser anexado a um servidor HTTP, [TsgcWebSocketHTTP_Server](#) ou [TsgcWebSocketServer_HTTPAPI](#) utilizando a propriedade `Server`. Você pode configurar os endpoints do servidor que tratarão as opções de registro e autenticação, e as opções do WebAuthn, como algoritmos suportados, origens e muito mais.

Endpoints Options

Aqui você pode configurar os endpoints do servidor que tratarão as requisições HTTP/JavaScript para utilizar o WebAuthn como autenticador. O componente já vem configurado com endpoints padrão, mas você pode alterar todos eles para atender às suas necessidades.

- **AuthenticationOptions:** por padrão é `/sgcWebAuthn/Authentication/Options`
- **AuthenticationVerify:** por padrão é `/sgcWebAuthn/Authentication/Verify`
- **RegistrationOptions:** por padrão é `/sgcWebAuthn/Registration/Options`
- **RegistrationVerify:** por padrão é `/sgcWebAuthn/Registration/Verify`
- **Webauthn:** inclui a biblioteca javascript usada por padrão. Você pode desabilitar esta propriedade e usar sua própria biblioteca webauthn.
- **Test:** por padrão está desabilitado; utilize apenas para testar a funcionalidade WebAuthn.

Exemplo: se o seu servidor está escutando no domínio `www.test.com`, a requisição às opções de autenticação por padrão será `http://www.test.com/sgcWebAuthn/Authentication/Options`

WebAuthn Options

Nesta propriedade, você pode configurar as principais opções do Componente Servidor WebAuthn.

- **RelyingParty:** uma propriedade **obrigatória** onde o **nome DNS do servidor** deve ser definido. Exemplo: se o servidor está em execução no domínio `www.test.com`, defina esta propriedade como `"www.test.com"`.

O WebAuthn utiliza origens para impor restrições da política de mesma origem, que são essenciais para prevenir ataques de phishing e cross-site. Durante os processos de registro e autenticação WebAuthn, a origem é estritamente validada pelo navegador e pelo authenticator.

- **Origins:** Se as requisições puderem vir de diferentes origens, use a propriedade `Origin` para definir as origens adicionais. Exemplo: se as requisições puderem vir de `login.test.com` e `www.test.co.uk`, configure a propriedade `Origins` com os valores: `https://login.test.com` e `https://www.test.co.uk`
- **TopOrigins:** Normalmente, o WebAuthn depende da origem do frame chamador (ou seja, aquele que invoca `navigator.credentials.create()` ou `navigator.credentials.get()`). No entanto, páginas web podem ser incorporadas em iframes, que podem vir de uma origem diferente da página de nível superior. Isso abre potencial para abusos ou ataques no estilo clickjacking. Para mitigar isso, a especificação WebAuthn Level 2 introduz `TopOrigin`, onde você pode definir os `TopOrigins`.

No WebAuthn, `crossOrigin` é um parâmetro booleano que indica se a operação WebAuthn está sendo realizada a partir de um contexto cross-origin, como um iframe incorporado de uma origem diferente do contexto de navegação de nível superior.

Este parâmetro foi introduzido para ajudar navegadores e authenticators a tratar requisições de autenticação com segurança em ambientes embarcados—um cenário comum em aplicações web modernas.

- **AllowCrossOrigins:** se `true`, indica que requisições feitas a partir de um iframe cross-origin (por exemplo, um iframe em `https://auth.example.com` incorporado em uma página em `https://app.example.org`) são permitidas. Por padrão, está desabilitado.

O WebAuthn suporta uma variedade de **algoritmos criptográficos** para geração e verificação de credenciais de chave pública. Esses algoritmos são utilizados durante o registro de credenciais (com `navigator.credentials.create()`) e a autenticação (com `navigator.credentials.get()`), e garantem a assinatura e validação seguras dos desafios utilizando pares de chaves assimétricas. O servidor é configurado por padrão com o **ES256 e RS256**, que são os algoritmos mais comuns. Você pode alterar a qualquer momento quais algoritmos são suportados pela propriedade **Algorithms**. Os seguintes algoritmos são suportados:

- ES256
- ES384
- ES512
- RS256
- RS384
- RS512
- PS256
- PS384
- PS512
- RS1
- EdDSA

No WebAuthn, **attestation é um mecanismo opcional** que permite ao authenticator (por exemplo, dispositivo ou chave de segurança) fornecer informações sobre seu fabricante, modelo e características de segurança durante a criação da credencial. Essas informações ajudam o Relying Party (RP) a decidir se deve confiar no authenticator.

Diferentes formatos de atestação definem como esses dados são estruturados e verificados. Três formatos comumente utilizados são `android-key`, `packed`, e outros como `fido-u2f`, `apple` ou `none`. Por padrão, todos os formatos de atestação estão habilitados. Você pode encontrar abaixo a lista de formatos de atestação suportados:

- **NoneAttestation:** neste caso, nenhum dado de attestation é retornado. Prioriza a privacidade do usuário ao evitar a exposição de identificadores do dispositivo. Comum em aplicações que não se importam com a procedência do dispositivo.
- **PackedAttestation:** é um formato flexível e compacto usado por muitos autenticadores. O autenticador retorna um certificado de atestação e uma assinatura. Pode ser: **Atestação completa:** Assinada com uma chave e certificado fornecidos pelo fabricante ou **Autoatestação:** Assinada usando a chave privada da credencial. O mais amplamente utilizado em diferentes plataformas (por exemplo, YubiKey, Windows Hello).
- **TPMAttestation:** Utilizado por dispositivos com um Trusted Platform Module (TPM). A atestação é assinada utilizando chaves do TPM e inclui uma cadeia de certificados. Utilizado por desktops/laptops corporativos com chips TPM (por exemplo, máquinas Windows).
- **AndroidKeyAttestation:** Utilizado por dispositivos Android com o Android Keystore. A chave é gerada em hardware, e a attestation inclui informações assinadas por uma cadeia de certificados emitida pelo fabricante do dispositivo. Utilizado por telefones Android com keystores apoiados por hardware (TEE ou StrongBox).

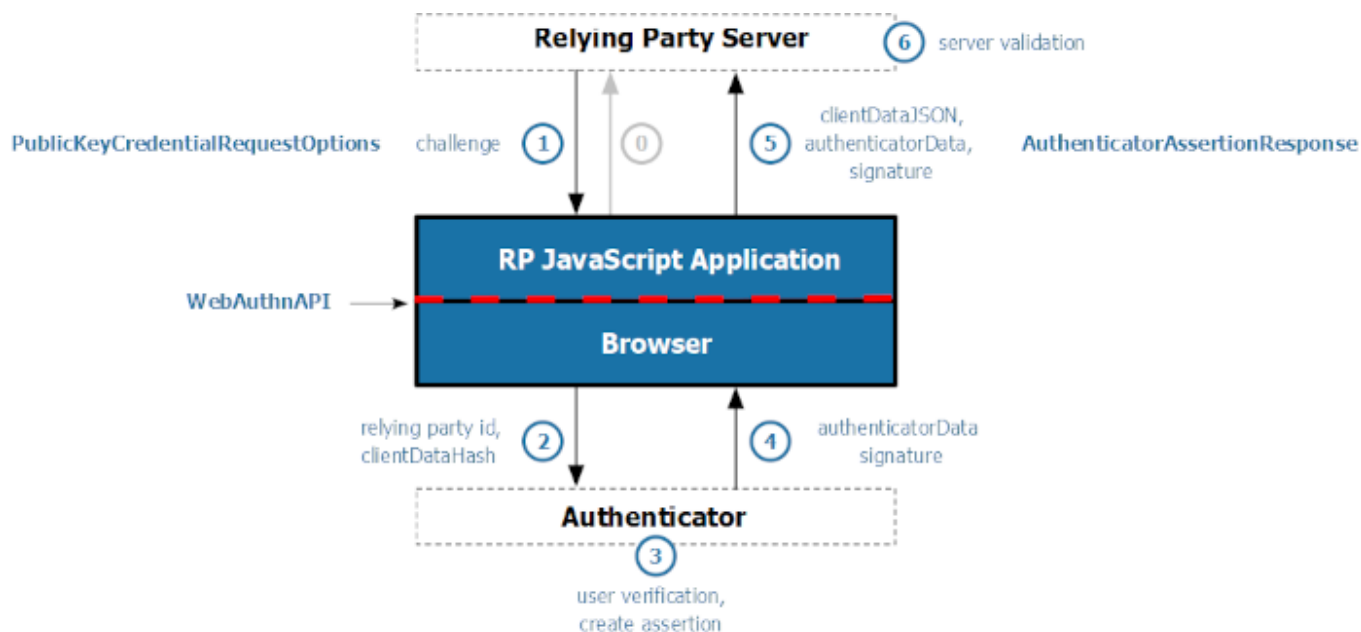
- **AppleAttestation:** Usado por autenticadores de plataforma da Apple, como Touch ID e Face ID. A attestation é gerada pelas APIs internas da Apple e inclui um formato de certificado especial. Usado no Safari com biometria da Apple.
- **FidoU2FAttestation:** Formato de atestação legado utilizado por autenticadores FIDO U2F. Retorna um certificado e uma assinatura compatíveis com U2F. Utilizado por chaves de segurança mais antigas (por exemplo, as primeiras YubiKeys) que suportam FIDO U2F.

Na API WebAuthn, **AllowCredentials** é um **campo opcional** utilizado durante o processo de autenticação (via `navigator.credentials.get()`). Ele **especifica uma lista de IDs de credenciais que estão autorizadas a autenticar o usuário** para uma Relying Party (RP) específica. Esse mecanismo permite que a RP controle quais credenciais são consideradas válidas para uma tentativa de login. A propriedade `credentials` possui os seguintes campos:

- **AllowCredentials:** se habilitado (false por padrão), especifica uma lista de IDs de credenciais que têm permissão para autenticar o usuário
- **ExcludeCredentials:** dado um nome de usuário, mostra todas as credenciais existentes já armazenadas no componente do servidor.
- **Limit:** o número máximo de credenciais que serão enviadas quando `ExcludeCredentials` for true.

WebAuthn Protocol

- **WebAuthn Registration:** O servidor gera um desafio e o envia ao cliente, que utiliza um authenticator (por exemplo, uma chave de segurança ou um dispositivo biométrico) para criar um par de chaves. A chave pública é enviada de volta e armazenada pelo servidor para autenticação futura. Encontre abaixo mais informações sobre os eventos do fluxo de registro:
 - [WebAuthn Registration Request](#)
 - [WebAuthn Registration Response](#)
 - [WebAuthn Registration Result](#)
- **WebAuthn Authentication:** O servidor envia um desafio ao cliente, que o assina utilizando a chave privada previamente registrada armazenada no authenticator. A resposta assinada é verificada pelo servidor utilizando a chave pública armazenada para confirmar a identidade do usuário. Encontre abaixo mais informações sobre os eventos do fluxo de autenticação:
 - [WebAuthn Authentication Request](#)
 - [WebAuthn Authentication Response](#)
 - [WebAuthn Authentication Result](#)
- **MDS:** O FIDO Alliance Metadata Service (MDS) é um repositório centralizado de Metadata Statements utilizado pelas relying parties para validar a attestation do autenticador e provar a autenticidade do modelo do dispositivo.



- **Authorization:** O cliente pode solicitar um bearer token do servidor durante o fluxo de autenticação. Este token pode ser utilizado posteriormente para abrir uma nova conexão WebSocket ou HTTP sem a necessidade de fazer login utilizando passkeys.
 - [WebAuthn Authorization WebSocket](#)
 - [WebAuthn Authorization HTTP](#)

Referência

- Propriedades — 4 propriedades disponíveis no componente.
- Métodos — 8 métodos públicos.
- Eventos — 15 eventos que você pode tratar.

WebAuthn | Registration

O **Registro WebAuthn** envolve o **cliente** (navegador), o **autenticador** (dispositivo de segurança) e a **relying party** (RP; servidor delphi/cbuilder TsgcWSAPIServer_WebAuthn).

Opções de Registro

A criação de uma nova credencial de usuário normalmente requer que o cliente inicie o fluxo de registro usando uma nova Requisição HTTP ao **Registration Options Endpoint** configurado no servidor TsgcWSAPIServer_WebAuthn. Por padrão, o endpoint é /sgcWebAuthn/Registration/Options, portanto, se seu servidor estiver escutando no domínio https://www.test.com, o cliente deve fazer uma nova requisição à url https://www.test.com/sgcWebAuthn/Registration/Options.

O cliente envia uma nova requisição passando como payload o seguinte json (usando test como nome de usuário)

```
{"username": "test", "algorithms": []}
```

O servidor lê a requisição e retorna uma resposta com um novo desafio.

```
{
  "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
  }
}
```

A resposta retorna os seguintes dados:

- **challenge:** Dados binários aleatórios, codificados em base64.
- **rp:** Informações do Relying Party (sua aplicação/site web).
- **user:** Identificador do usuário e nome de exibição.

- **pubKeyCredParams:** Algoritmos permitidos (por exemplo, ES256, RS256).
- **authenticatorSelection:** Preferências para o tipo de autenticador (platform ou cross-platform, opções de verificação do usuário).
- **timeout:** Valor de timeout sugerido.
- **attestation:** Preferência de transmissão de atestação ("none", "direct", "indirect").

Registration Verify

Agora o cliente tem a resposta do servidor, lê a resposta e o autenticador retorna os dados criptográficos ao navegador web do cliente. Agora o cliente envia uma nova Requisição HTTP com os seguintes dados ao Registration Verify Endpoint configurado no servidor TsgcWSAPIServer_WebAuthn, por padrão é /sgcWebAuthn/Registration/Verify

, portanto, se seu servidor estiver escutando no domínio <https://www.test.com>, o cliente deve fazer uma nova requisição à url <https://www.test.com/sgcWebAuthn/Registration/Verify>.

O authenticator responde de volta ao JavaScript no navegador com:

- **id:** Credential ID (ID único para o par de chaves gerado pelo authenticator).
- **rawId:** Forma binária do Credential ID.
- **type:** Tipo de credencial ("public-key").
- **response:** Contém attestationObject e clientDataJSON.

Encontre abaixo um exemplo:

```
{
  "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"
}
```

O servidor lê a requisição JSON do cliente e decodifica, verifica e armazena a chave pública e o credential ID.

Decodifique attestationObject e clientDataJSON:

- Extraia a chave pública de attestationObject.
- Valide se o desafio em clientDataJSON corresponde ao original enviado pelo servidor.
- Garanta a validação adequada da origem (myapp.example.com).

Store Credential:

- Se a verificação for bem-sucedida, o evento **OnWebAuthnRegistrationSuccessful** é chamado para que você possa armazenar com segurança o ID da credencial e a chave pública extraída no seu banco de dados para autenticações futuras.
- Se houver algum erro, o evento **OnWebAuthnRegistrationError** é chamado e você pode obter os detalhes do erro.

Fluxo de Registro

Obtenha mais informações sobre o processo de Fluxo de Registro usando os links a seguir:

- [Requisição de Registro WebAuthn](#)
- [Resposta de Registro WebAuthn](#)
- [WebAuthn Registration Result](#)

WebAuthn Registration | Request

Esta requisição de opções de registro é essencial para inicializar o registro WebAuthn com segurança. Ela solicita ao servidor para:

- Gera um desafio,
- Procure ou provisione o user ID,
- Retorna um objeto `PublicKeyCredentialCreationOptions` válido (com RP, user, algoritmos, etc.).

O navegador deve obter as opções de criação de credencial do servidor para iniciar um registro seguro de credencial WebAuthn. Isso normalmente é feito através de uma requisição HTTPS POST do navegador para um endpoint do servidor (por exemplo, `/sgcWebAuthn/Registration/Options`), que prepara parâmetros como o challenge, RP ID e informações do usuário.

O corpo da requisição contém dados que identificam o usuário e possivelmente fornecem preferências de configuração. Frequentemente, ele se parece com isto:

```
{
  "username": "alice@example.com",
  "displayName": "Alice Smith",
  "authenticatorSelection": {
    "authenticatorAttachment": "platform", // optional: platform or cross-platform
    "userVerification": "preferred" // optional: required | preferred | discouraged
  },
  "attestation": "none", // or "direct", "indirect"
  "residentKey": "discouraged" // optional
}
```

O Componente Servidor WebAuthn escuta no endpoint configurado na propriedade **EndpointOptions.RegistrationOptions** a requisição inicial do navegador para obter um `PublicKeyCredentialCreationOptions`. Quando o servidor recebe uma nova Requisição HTTP, o evento **OnWebAuthnRegistrationOptionsRequest** é chamado e você pode acessar a requisição enviada pelo cliente e cancelar a requisição definindo o parâmetro `Accept` como `False`.

```
void __fastcall TForm1::OnWebAuthnRegistrationOptionsRequest(TObject *Sender, TsgcWebAuthn_RegistrationOptions_Re
{
    if (aRequest->Username == "anonymous")
        Accept = false;
}
```

WebAuthn Registration | Response

O servidor responde à requisição de opções de registro do cliente (por exemplo, POST /sgcWebAuthn/Registration/Options) com um payload JSON semelhante ao seguinte (após a codificação base64url dos campos binários):

```
{
  "publicKey": {
    "rp": {
      "name": "esegece software",
      "id": "esegece.com"
    },
    "user": {
      "id": "c3ViamVjdC1pZA", // base64url-encoded ArrayBuffer (user handle)
      "name": "webauthn@esegece.com",
      "displayName": "Delphi Developer"
    },
    "challenge": "Xz8x2K6nY3gZ...", // base64url-encoded challenge
    "pubKeyCredParams": [
      { "type": "public-key", "alg": -7 }, // ES256
      { "type": "public-key", "alg": -257 } // RS256
    ],
    "timeout": 60000,
    "attestation": "none",
    "authenticatorSelection": {
      "authenticatorAttachment": "platform",
      "userVerification": "preferred",
      "residentKey": "discouraged"
    }
  }
}
```

Veja abaixo uma descrição dos campos:

- **rp:** Informações do Relying Party:
 - **name:** Nome de exibição amigável (por exemplo, "Example Inc.").
 - **id:** ID do Relying Party (deve corresponder ao origin do site ou ser um sufixo registrável).
- **user:** Informações que identificam a conta do usuário:
 - **id:** Array de bytes único e estável (por exemplo, UUID do usuário).
 - **name:** Identificador único de usuário (nome de usuário ou e-mail).
 - **displayName:** Nome de exibição voltado ao usuário.
- **challenge:** Um nonce criptograficamente aleatório (16–64 bytes), codificado em base64url.
- **pubKeyCredParams:** Algoritmos que o RP está disposto a aceitar:
 - **Comuns:** -7 (ES256), -257 (RS256)
- **timeout:** Timeout opcional para a operação de criação (em ms).
- **attestation:** Um de "none", "indirect", "direct".
- **authenticatorSelection:**
 - **authenticatorAttachment:** "platform" (integrado) ou "cross-platform" (por exemplo, YubiKey).
 - **userVerification:** "required", "preferred" ou "discouraged".
 - **residentKey:** "required", "preferred" ou "discouraged".

Antes de a resposta ser enviada ao cliente, o evento **OnWebAuthnRegistrationOptionsResponse** é chamado, permitindo que você personalize a resposta.

```
void __fastcall OnWebAuthnRegistrationOptionsResponse(TObject *Sender,
  TsgcWebAuthn_RegistrationOptions_Request* aRequest,
  TsgcWebAuthn_RegistrationOptions_Response* aResponse)
{
  if (aRequest->Username == "esegece.com") {
```

```
aResponse->ExcludeCredentials->AddCredentialRecordFromJSON("json1.txt");  
aResponse->ExcludeCredentials->AddCredentialRecordFromJSON("json2.txt");  
}  
}
```

Registro WebAuthn | Resultado

O objetivo é verificar a autenticidade e a integridade dos dados retornados pelo cliente, garantir que a credencial esteja vinculada ao usuário esperado e registrar com segurança uma credencial de chave pública para autenticação futura.

O servidor deve validar a resposta do cliente seguindo estes passos:

- Analise a resposta do cliente: O cliente envia ao servidor uma resposta como esta.

```
{
  "id": "base64url-encoded credential ID",
  "rawId": "base64url-encoded ID bytes",
  "response": {
    "clientDataJSON": "base64url",
    "attestationObject": "base64url"
  },
  "type": "public-key"
}
```

- O servidor valida o clientDataJSON e o attestationObject
- Verifica o Attestation Statement (se definido)
- Se tudo estiver válido, ele armazena a credencial e pode confiar na chave pública para logins futuros.

Registro Bem-sucedido

Se a resposta enviada pelo cliente for válida, o evento **OnWebAuthnRegistrationSuccessful** é chamado e o Credential Record pode ser armazenado com segurança em um banco de dados para validações de login futuras.

```
void __fastcall OnWebAuthnRegistrationSuccessful(TObject *Sender,
  TsgcWebAuthn_Registration* aRegistration,
  TsgcWebAuthn_CredentialRecord* aCredentialRecord,
  bool &Accept)
{
  // Store in a DB
  DB->Credentials->Append();
  DB->Credentials->FieldByName("Credentials")->AsString = aCredentialRecord->AsJSON;
  DB->Credentials->Post();
}
```

Registration Error

Se houver algum erro ao validar a resposta do cliente, o evento **OnWebAuthnRegistrationError** é chamado e você pode acessar o motivo do erro no parâmetro aError.

```
void __fastcall OnWebAuthnRegistrationError(TObject *Sender,
  TsgcWebAuthn_RegistrationVerify_Request* aRequest,
  TsgcWebAuthn_Registration* aRegistration,
  const String aError)
{
  Log("#webauthn_registration_error: " + aError);
}
```

WebAuthn | Autenticação

A **Autenticação WebAuthn** permite que os usuários façam login utilizando credenciais de chave pública previamente registradas. Ela envolve validar um desafio (challenge) assinado utilizando a chave pública armazenada do usuário no registro.

Authentication Options

A autenticação requer que o cliente inicie o fluxo de autenticação utilizando uma nova HTTP Request ao **Authentication Options Endpoint** configurado no servidor TsgcWSAPIServer_WebAuthn. Por padrão, o endpoint é /sgcWebAuthn/Authentication/Options, então, se o seu servidor estiver escutando no domínio https://www.test.com, o cliente deve fazer uma nova requisição à url https://www.test.com/sgcWebAuthn/Authentication/Options.

O cliente envia a assertion (resposta de autenticação) ao Servidor via POST

```
{"username": "test", "user_verification": "preferred"}
```

O servidor gera 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": {}
}
```

Authentication Verify

A autenticação requer que o cliente inicie o fluxo de autenticação utilizando uma nova HTTP Request ao **Authentication Verify Endpoint** configurado no servidor TsgcWSAPIServer_WebAuthn. Por padrão, o endpoint é /sgcWebAuthn/Authentication/Verify, então, se o seu servidor estiver escutando no domínio https://www.test.com, o cliente deve fazer uma nova requisição à url https://www.test.com/sgcWebAuthn/Authentication/Verify.

O navegador solicita ao usuário que utilize seu autenticador (por exemplo, impressão digital, YubiKey). O autenticador assina o desafio com a chave privada vinculada ao credential ID. A credencial retornada inclui:

- **rawId**: ID da credencial (base64url).
- **response.authenticatorData**: Metadados do autenticador (binário).
- **response.clientDataJSON**: Contém o desafio (challenge) original, a origin e o type.
- **response.signature**: Assinatura sobre authenticatorData + hash(clientDataJSON).

Encontre abaixo um exemplo json da requisição do cliente:


```
{
  "id": "yeA4BVRlrAfLG-KzqsL_rlI4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
  "rawId": "yeA4BVRlrAfLG-KzqsL_rlI4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
  "response": {
    "authenticatorData": "SZYN5YgOjGh0NBcPZHZgW4_krrmihjLHmVzzuoMdl2MFAAAABw",
    "clientDataJSON": "eyJ0eXBldjoid2ViYXV0aG4uZ.....",
    "signature": "MEQCIARqvvys8....",
    "userHandle": "36b9d6a84204487382fee62e7e67a80d"
  },
  "type": "public-key",
  "clientExtensionResults": {},
  "authenticatorAttachment": "cross-platform"
}
```

O servidor lê a requisição do cliente, valida que a credencial está armazenada e verifica a assinatura. Se a assinatura for válida, o evento **OnWebAuthnAuthenticationSuccessful** é chamado.

Authentication Flow

Obtenha mais informações sobre o processo de Authentication Flow utilizando os links a seguir:

- [Requisição de Autenticação WebAuthn](#)
- [Resposta de Autenticação WebAuthn](#)
- [Resultado da autenticação WebAuthn](#)

WebAuthn Authentication | Request

Quando um usuário tenta fazer login, o navegador envia uma requisição ao servidor solicitando as opções de autenticação (também chamadas de "assertion options").

```
{  
  "username": "alice@example.com"  
}
```

Esta requisição permite que o servidor:

- Gera um desafio (um valor seguro e aleatório).
- Consulta quais credenciais (IDs de chave pública) são válidas para este usuário.
- Responde com parâmetros criptográficos que o navegador utilizará em `navigator.credentials.get()`.

Quando o Componente Servidor WebAuthn recebe esta requisição, o que normalmente é feito através de uma requisição HTTPS POST do navegador para um endpoint do servidor (por exemplo, `/sgcWebAuthn/Authentication/Options`), o evento **OnWebAuthnAuthenticationOptionsRequest** é chamado, para que você possa adicionar as credenciais associadas ao nome de usuário (se houver).

```
void __fastcall TForm1::OnWebAuthnAuthenticationOptionsRequest(  
  TObject *Sender,  
  TsgcWebAuthn_AuthenticationOptions_Request *aRequest,  
  TsgcWebAuthn_CredentialRecords &CredentialRecords,  
  bool &Accept)  
{  
  if (UserExistsInDB(aRequest->Username)) {  
    while (!EOF()) {  
      CredentialRecords.AddCredentialRecordFromJSON(RecordFromDB());  
      Next();  
    }  
  }  
}
```

WebAuthn Authentication | Response

Assim que o navegador (cliente) envia uma requisição ao servidor com o nome de usuário (ou um identificador similar), o servidor responde com as opções de autenticação que o cliente utilizará para iniciar o processo de autenticação através da API `navigator.credentials.get()` do navegador.

Esta resposta fornece os parâmetros e restrições necessários ao navegador para gerar uma assertion de autenticação WebAuthn usando o authenticator do usuário (por exemplo, chave de segurança, dispositivo biométrico).

Exemplo de Resposta JSON:

```
{
  "challenge": "Z3lVbWV5YXBpbmdvZG90IQ",
  "timeout": 60000,
  "rpId": "example.com",
  "allowCredentials": [
    {
      "type": "public-key",
      "id": "dXNlckNyZWZJZA",
      "transports": ["usb", "nfc", "ble"]
    }
  ],
  "userVerification": "preferred"
}
```

Veja abaixo uma descrição dos campos:

- **challenge:** Um desafio criptográfico (gerado aleatoriamente) para prevenir ataques de replay. Deve ser único por requisição.
- **timeout:** Opcional. Quanto tempo o navegador deve esperar (em ms) antes de cancelar a operação.
- **rpId:** Relying Party Identifier — geralmente o nome do seu domínio.
- **allowCredentials:** Uma lista de credenciais aceitáveis (IDs de credencial) que o usuário registrou anteriormente.
- **userVerification:** Informa ao navegador como verificar o usuário: "required", "preferred" ou "discouraged".
- **extensions (opcional):** Recursos ou dados adicionais solicitados ao authenticator.

Antes de a resposta ser enviada ao cliente, o evento **OnWebAuthnAuthenticationOptionsResponse** é chamado, permitindo que você personalize a resposta.

Autenticação WebAuthn | Resultado

O objetivo é validar a assertion assinada fornecida pelo navegador, que prova que o usuário possui a chave privada originalmente registrada. É isso que faz o login seguro do usuário. Depois que o usuário interage com seu autenticador (por exemplo, impressão digital, chave de segurança), o navegador envia uma requisição POST de volta ao servidor com o resultado da autenticação. Veja abaixo um exemplo json:

```
{
  "id": "credential-id",
  "rawId": "base64url-encoded-credential-id",
  "type": "public-key",
  "response": {
    "clientDataJSON": "base64url",
    "authenticatorData": "base64url",
    "signature": "base64url",
    "userHandle": "optional"
  }
}
```

Quando o servidor recebe esta requisição no endpoint configurado (por exemplo, /sgcWebAuthn/Authentication/Verify), ele deve validar os seguintes passos:

- Recuperar Credencial do BD: Utilize o credential-id para localizar a chave pública registrada do usuário e os metadados (por exemplo, o contador de assinaturas).
- Decodificar & Fazer Parse do clientDataJSON
- Validar Authenticator Data
- Verify Signature

Se todas as validações estiverem corretas, a autenticação é bem-sucedida e o evento **OnWebAuthnAuthenticationSuccessful** é chamado.

Se alguma verificação falhar, o evento **OnWebAuthnAuthenticationError** é chamado com o motivo do erro.

WebAuthn | MDS

O **Metadata Service (MDS)** é um **serviço centralizado fornecido pela FIDO Alliance** que agrega e publica **Metadata Statements** sobre autenticadores certificados por meio dos programas de certificação FIDO. Essas declarações contêm informações detalhadas de segurança, conformidade e operacionais sobre os autenticadores.

- O endpoint do serviço é frequentemente referido como **MDS3** (a terceira versão do protocolo Metadata Service).
- As Relying Parties (por exemplo, sites ou aplicações que implementam WebAuthn) **recuperam declarações de metadados** do MDS para tomar decisões de confiança informadas sobre os autenticadores.

O MDS adiciona uma camada crucial de validação de confiança e segurança para os relying parties que utilizam WebAuthn:

- **Validação do Authenticator:** Habilita a verificação da conformidade do authenticator com os padrões FIDO e ajuda a validar o AAGUID apresentado em uma attestation WebAuthn.
- **Deteção de Comprometimento e Revogação:** Fornece informações atualizadas sobre authenticators comprometidos ou revogados, permitindo que as relying parties bloqueiem dispositivos inseguros.
- **Garantia de Segurança:** Ajuda a impor políticas de segurança, como permitir apenas autenticadores que atendam a um certo nível de certificação FIDO ou força de verificação de usuário.
- **Interoperabilidade:** Garante comportamento e expectativas de segurança consistentes entre diferentes navegadores, plataformas e dispositivos que usam diferentes authenticators.

Configuração

Você pode configurar o uso do MDS utilizando a propriedade `WebAuthnOptions.MDS`; encontre abaixo as principais propriedades:

- **Enabled:** se `true` (o valor padrão), as requisições `webauthn` serão validadas contra o arquivo MDS configurado.
- **MDS_FileName:** o caminho onde o arquivo MDS está armazenado. Pode ser baixado da seguinte URL: <https://mds3.fidoalliance.org/>
- **RootCert_FileName:** o caminho onde o Root Certificate está armazenado. Deve ser definido para validar a cadeia de certificados. Pode ser baixado de <https://valid.r3.roots.globalsign.com/>
- **Leaf_CertificateCRL:** se `true` (por padrão é `false`), o certificado folha do blob será validado contra a CRL (Certificate Revocation List).
- **CRL_FileName:** o caminho onde o arquivo CRL está armazenado. Se não estiver definido e `Leaf_CertificateCRL` estiver habilitado, ele tentará baixar o CRL automaticamente.

WebAuthn Authorization

Se você quiser que o servidor envie um bearer token após uma autenticação bem-sucedida que possa ser usado para abrir uma nova conexão WebSocket ou HTTP, passe o parâmetro `token = true`. Exemplo:

```
{
  "username": "alice@example.com", "token": true
}
```

Após uma autenticação bem-sucedida, o servidor enviará uma resposta como esta:

```
{
  "verified": "ok",
  "authentication": {
    "token": "C760C1C39E3D4E829693A13F18F5CFDE537B516336FC48F7BAB0276176F9E6DE"
  }
}
```

O evento **OnWebAuthnUnauthorized** é chamado quando uma requisição não é autorizada e a conexão será desconectada. Aqui você pode configurar quais endpoints requerem autenticação WebAuthn e quais não.

WebAuthn Authorization | HTTP

Uma vez que você tenha um novo token, basta enviar um cabeçalho de autorização com este bearer token. Você pode usar a propriedade CustomHeaders do TsgcHTTP1Client para configurar o Bearer Token. Exemplo:

```
String GetHTTPRequest(const String aURL, const String aToken)
{
    // Create the HTTP client dynamically
    std::unique_ptr<TsgcHTTP1Client> oHTTP(new TsgcHTTP1Client(nullptr));
    // Add Authorization header
    oHTTP->Request->CustomHeaders->AddValue("Authorization", "Bearer " + aToken);
    // Perform GET request and return result
    return oHTTP->Get(aURL);
}
```

WebAuthn Authorization | WebSocket

Uma vez que você tenha um novo token, basta enviar um cabeçalho de autorização com este bearer token. Você pode usar o evento OnHandShake para adicionar o Bearer Token à requisição de conexão. Exemplo:

```
void __fastcall OnClientHandshake(TsgcWSConnection *Connection, TStringList *Headers)
{
    Headers->Add("Authorization: Bearer C760C1C39E3D4E829693A13F18F5CFDE537B516336FC48F7BAB0276176F9E6DE");
}
```


Webauthn | Javascript Client

WebAuthn (Web Authentication API) é um padrão W3C que permite autenticação segura sem senha usando **criptografia de chave pública**. Em vez de senhas, os usuários registram-se e autenticam-se usando **autenticadores baseados em hardware** (como leitores de impressão digital, Face ID, YubiKeys, etc.) ou autenticadores de plataforma (integrados, como o Touch ID).

Veja abaixo como tratar o Registro e a Autenticação utilizando um cliente Javascript.

WebAuthn Registration

Como funciona o WebAuthn Registration

- **O usuário inicia o registro:**
 - O usuário fornece um nome de usuário e clica em **Registrar**.
- **O navegador solicita as Options ao servidor:**
 - O frontend faz uma requisição POST para obter as opções de registro.
- **O Navegador Cria as Credenciais:**
 - Utilizando a API `navigator.credentials.create()` por meio de `SimpleWebAuthnBrowser.startRegistration()`, uma credencial é criada.
- **O Servidor Verifica o Registro:**
 - O navegador envia a credencial de volta ao servidor.
 - O servidor verifica o registro e armazena a chave pública para esse usuário.

O componente [TsgcWSAPIServer_WebAuthn](#) possui um arquivo html para testar o protocolo WebAuthn. Este arquivo HTML contém uma interface mínima e JavaScript para interagir com WebAuthn.

Tutorial de sgcWebAuthn.html

Visão geral da estrutura:

- **Username Input:** Captura o identificador do usuário.
- **Botões:**
 - Register – inicia o registro WebAuthn.
 - Authenticate – inicia o login (tratado de forma similar).
- **Debug Console:** Exibe informações de depuração em tempo real (JSON do WebAuthn).

1. UI HTML para Entrada

```
<input type="text" id="username" name="username" autocomplete="username webauthn" />
<button id="btnRegBegin"><strong>Register</strong></button>
```

2. JavaScript: Handler de clique do botão

```
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. Resposta do servidor (endpoint simulado em HTML)

```
fetch('/sgcWebAuthn/Registration/Options', ...)
fetch('/sgcWebAuthn/Registration/Verify', ...)
```

4. Finalizando o Registro

```
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!`;
}
```

WebAuthn Authentication

Como Funciona a Autenticação WebAuthn

- **O usuário inicia a autenticação:**
 - O usuário fornece um nome de usuário e clica em **Registrar**.
- **O navegador solicita as Options ao servidor:**
 - O frontend faz uma requisição POST para obter as opções de autenticação.
- **O Navegador Cria as Credenciais:**
 - Utilizando a API `navigator.credentials.get()` via `SimpleWebAuthnBrowser.startAuthentication()`.
- **O servidor verifica a autenticação:**
 - O navegador envia a credencial de volta ao servidor.
 - O servidor verifica o resultado assinado.

1. Configuração da Interface 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. Obter as Authentication Options

Antes de chamar `startAuthentication`, você envia o nome de usuário ao servidor

```
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'
  }),
});
```

O servidor responde com um objeto JSON que inclui:

```
{
  "challenge": "base64url-encoded-random-string",
  "allowCredentials": [
    {
      "id": "base64url-credential-id",
      "type": "public-key"
    }
  ],
  "userVerification": "preferred",
  "rpId": "yourdomain.com"
}
```

Isto é chamado de `PublicKeyCredentialRequestOptions`.

3. Receba a Resposta do Authenticator

O `asseResp` se parece com isto (simplificado):

```
{
  "id": "credentialId",
  "rawId": "base64url-encoded-id",
  "response": {
    "authenticatorData": "...",
    "clientDataJSON": "...",
    "signature": "...",
    "userHandle": "..."
  },
  "type": "public-key",
  "clientExtensionResults": {}
}
```

Esta resposta prova que o usuário:

- Possui a chave privada armazenada no autenticador
- Assinou o desafio do servidor
- Estava fisicamente presente (se exigido)

4. Envie a Signed Authentication Response ao servidor

Depois que o usuário interage com seu autenticador (via `startAuthentication()`), você obtém um objeto de resposta em 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. Obtenha a Resposta do Servidor

O servidor responderá com um resultado como este:

```
{ "verified": true }
```

Ou se algo deu errado:

```
{ "verified": false, "error": "Invalid signature" }
```

E você o trata no código do seu 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

O que é o Amazon SQS?

Amazon Simple Queue Service (SQS) é um serviço de enfileiramento de mensagens totalmente gerenciado que permite desacoplar e escalar microsserviços, sistemas distribuídos e aplicações serverless. O SQS elimina a complexidade e a sobrecarga associadas ao gerenciamento e à operação de middleware orientado a mensagens, e capacita os desenvolvedores a focarem no trabalho que faz a diferença. Usando o SQS, você pode enviar, armazenar e receber mensagens entre componentes de software em qualquer volume, sem perder mensagens ou exigir que outros serviços estejam disponíveis.

Benefícios

Eliminar a sobrecarga administrativa

Com o SQS, não há custo inicial, nem necessidade de adquirir, instalar e configurar software de mensagens, e nem a construção e manutenção demorada de infraestrutura de suporte.

Entregar mensagens de forma confiável

O SQS permite que você desacople os componentes da aplicação para que eles sejam executados e falhem de forma independente, aumentando a tolerância geral a falhas do sistema.

Mantenha os dados sensíveis seguros

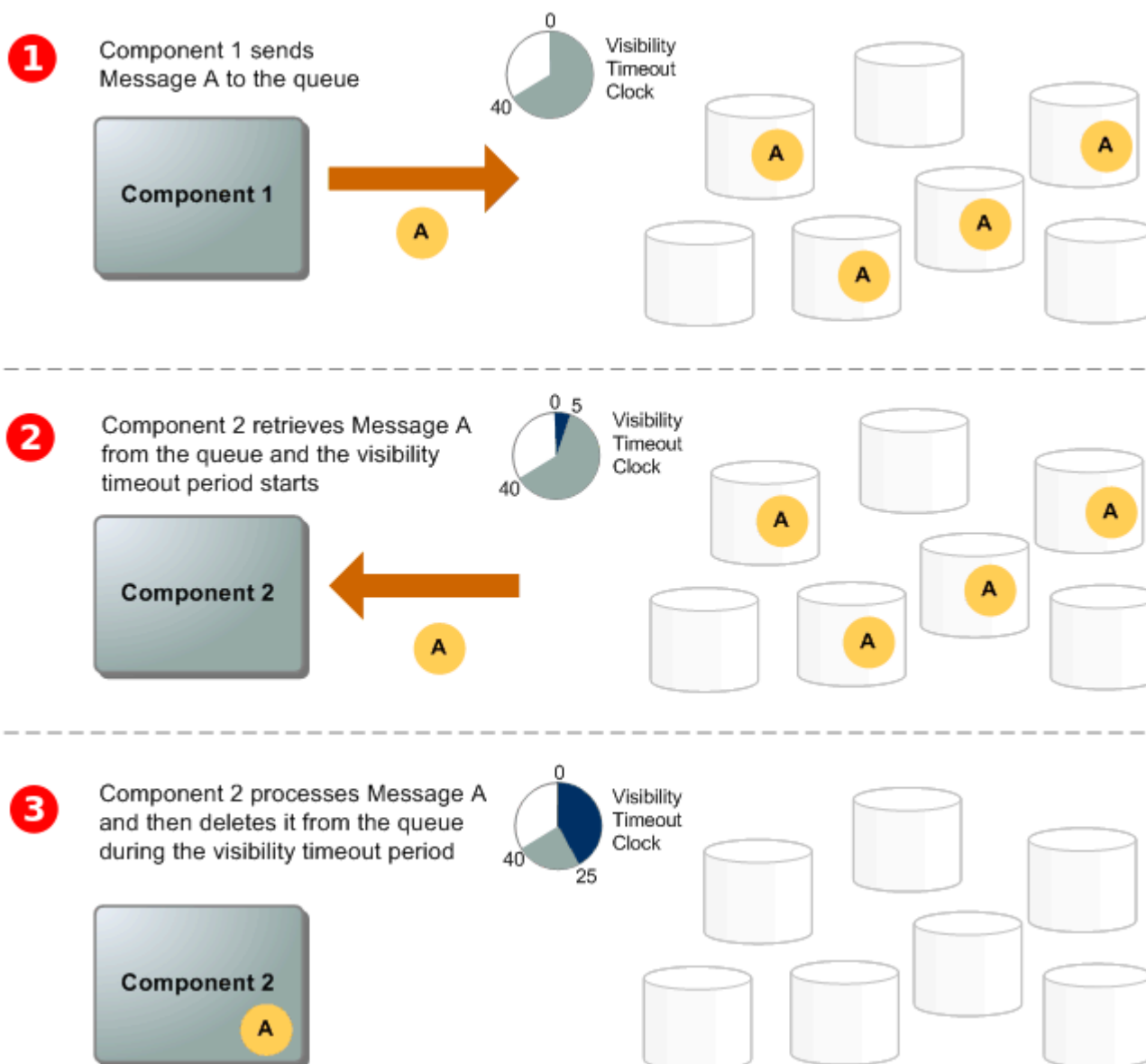
Você pode usar o Amazon SQS para trocar dados sensíveis entre aplicações usando criptografia do lado do servidor (SSE) para criptografar o corpo de cada mensagem.

Escale de forma elástica e econômica

O SQS escala elasticamente com sua aplicação, então você não precisa se preocupar com planejamento de capacidade e pré-provisionamento.

WorkFlow

O cenário a seguir descreve o ciclo de vida de uma mensagem Amazon SQS em uma fila, da criação à exclusão.



Primeiros Passos com Amazon SQS

Antes de começar, conclua as etapas em Setting Up Amazon SQS.

Passo 1: Crie uma fila

1. Faça login no console do Amazon SQS.
2. Escolha Create New Queue.
3. Na página Create New Queue, certifique-se de estar na região correta e então digite o Queue Name.
4. Standard é selecionado por padrão. Escolha FIFO.
5. Para criar sua fila com os parâmetros padrão, escolha Quick-Create Queue.

Sua nova queue é criada e selecionada na lista de queues.

Passo 2: Envie uma mensagem

Após criar sua fila, você pode enviar uma mensagem para ela. O exemplo a seguir mostra o envio de uma mensagem para uma fila existente.

1. Na lista de filas, selecione a fila que você criou.
2. Em Queue Actions, selecione Send a Message.
3. Sua mensagem é enviada e a caixa de diálogo Send a Message to QueueName é exibida, mostrando os atributos da mensagem enviada.

Etapa 3: Receba e Exclua Sua Mensagem

Depois de enviar uma mensagem para uma queue, você pode consumi-la (recuperá-la da queue). Quando você solicita uma mensagem de uma queue, não pode especificar qual mensagem obter. Em vez disso, você especifica o número máximo de mensagens (até 10) que deseja obter.

Passo 4: Exclua Sua Fila

Se você não usa uma queue Amazon SQS (e não prevê usá-la em um futuro próximo), é uma boa prática excluí-la do Amazon SQS.

SQS Client

```
// TsgcHTTPAWS_SQS_Client is the component used for connect to Amazon SQS.
// Client connects using HTTPs protocol and authenticates using Access Key provided by Amazon.

// Before you attempt to connect to SQS service, you must set some data in AWSOptions property.

// Region: your endpoint region, example: us-east-1.
// AccessKey: access key provided by Amazon.
// SecretKey: secret key provided by Amazon.

// The following methods are supported by SQS client:

// AddPermission
// Adds a permission to a queue for a specific principal. This allows sharing access to the queue.

// ChangeMessageVisibility
// Changes the visibility timeout of a specified message in a queue to a new value. The default visibility
// The minimum is 0 seconds. The maximum is 12 hours.

// ChangeMessageVisibilityBatch
// Changes the visibility timeout of multiple messages. This is a batch version of ChangeMessageVisibility.
// The result of the action on each message is reported individually in the response. You can send up to 10

// CreateQueue
// Creates a new standard or FIFO queue. You can pass one or more attributes in the request.

vURL = SQS->CreateQueue("sqs_queue");
if (vURL != "")
{
    DoLog("#CreateQueue: " + vURL);
}

// DeleteMessage
// Deletes the specified message from the specified queue. To select the message to delete, use the Receipt

if (SQS->DeleteMessage("sqs_queue", "...receipt handle goes here...") == true)
{
    DoLog("#DeleteMessage: ok");
}
else
{
    DoLog("#DeleteMessage: error");
}

// DeleteMessageBatch
// Deletes up to ten messages from the specified queue. This is a batch version of DeleteMessage. The result

// DeleteQueue
// Deletes the queue specified by the queue name, regardless of the queue's contents.

if (SQS->DeleteQueue(txtQueueName->Text) == true)
{
    DoLog("#Delete Queue: ok");
}
else
```

```

    {
        DoLog("#Delete Queue: error");
    }

// GetQueueAttributes
// Gets attributes for the specified queue.

TsgcSQSAttributes *oAttributes = new TsgcSQSAttributes();
try
{
    if (SQS->GetQueueAttributes("sqs_queue", oAttributes) == true)
    {
        for (int i = 0; i < oAttributes->Count; i++)
        {
            DoLog("#Attribute: " + static_cast<TsgcSQSAttribute*>(oAttributes->Item[i])
                ->AttributeName + " " + static_cast<TsgcSQSAttribute*>(oAttributes->Item[i])
                ->AttributeValue);
        }
    }
    else
    {
        DoLog("#GetQueueAttributes: error");
    }
}
finally
{
    oAttributes->Free();
}

// GetQueueUrl
// Returns the URL of an existing Amazon SQS queue.

// ListDeadLetterSourceQueues
// Returns a list of your queues that have the RedrivePolicy queue attribute configured with a dead-letter

// ListQueueTags
// List all cost allocation tags added to the specified Amazon SQS queue.

// PurgeQueue
// Deletes the messages in a queue specified by the QueueName parameter.

if (SQS->PurgeQueue("sqs_queue") == true)
{
    DoLog("#PurgeQueue: ok");
}
else
{
    DoLog("#PurgeQueue: error");
}

// ReceiveMessage
// Retrieves one or more messages (up to 10), from the specified queue.

TsgcSQSReceiveMessageResponses *oResponses = new TsgcSQSReceiveMessageResponses();
try
{
    if (SQS->ReceiveMessage("sqs_test", oResponses) == true)
    {
        for (int i = 0; i < oResponses->Count; i++)
        {
            DoLog("#ReceiveMessage: " + static_cast<TsgcSQSReceiveMessageResponse*>(oResponses->Item[i])->Body)
                FReceiptHandle = static_cast<TsgcSQSReceiveMessageResponse*>(oResponses->Item[i])->ReceiptHandle;
        }
    }
}
finally
{
    oResponses->Free();
}

// RemovePermission
// Revokes any permissions in the queue policy that matches the specified Label parameter.

// SendMessage
// Delivers a message to the specified queue.

if (SQS->SendMessage("sqs_queue", "My First Message") == true)
{
    DoLog("#SendMessage: ok");
}
else
{
    DoLog("#SendMessage: error");
}

// SendMessageBatch

```



```

// Delivers up to ten messages to the specified queue. This is a batch version of SendMessage.

// SetQueueAttributes
// Sets the value of one or more queue attributes. When you change a queue's attributes, the change can tak
// for most of the attributes to propagate throughout the Amazon SQS system.

TsgcSQSAttributes *oAttributes = new TsgcSQSAttributes();
try
{
    oAttributes->AddSQSAttribute(sqsatVisibilityTimeout, "45");
    if (SQS->SetQueueAttributes("sqs_queue", oAttributes) == true)
    {
        DoLog("#SetQueueAttributes: ok");
    }
    else
    {
        DoLog("#SetQueueAttributes: error");
    }
}
finally
{
    oAttributes->Free();
}

// TagQueue
// Add cost allocation tags to the specified Amazon SQS queue.

// UntagQueue
// Remove cost allocation tags from the specified Amazon SQS queue.

```

Eventos

OnSQSBeforeRequest

Este evento é chamado antes de o componente sqs fazer uma requisição HTTP. Você pode acessar o parâmetro URL e, se o parâmetro Handled for definido como True, significa que o componente não fará uma requisição HTTP.

OnSQSError

Se houver qualquer erro quando o componente faz uma requisição, este evento será chamado com o Error Code e a Error Description.

OnSQSResponse

Este evento é chamado após uma requisição HTTP com resposta bruta do servidor.

Google Cloud | Google OAuth2 Keys

Para usar os componentes Google Cloud do sgcWebSockets e Autenticar usando OAuth2, primeiro você deve obter a OAuth2 Key do Google Cloud.

Veja abaixo os passos para obter as chaves Google OAuth2 e como configurá-las na nossa aplicação de exemplo PubSub.

Primeiro **faça login** na sua **Google Cloud Account** e utilize um projeto existente ou crie um novo.

Depois disso, vá ao menu **Credentials** e pressione o botão **CREATE CREDENTIALS**, selecione a opção **OAuth Client ID**.

Selecione o tipo da sua aplicação e defina um nome descritivo.

Se bem-sucedido, você obterá seu Client ID e 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-ehcskpfsp4180r1bdfoe6mc32e3ncmn0.apps.gc



Your Client Secret

pvogD9reE0t9i1L6eR1jE60Z



OK

Não compartilhe seus dados OAuth2 com ninguém!

Agora copie para o exemplo PubSub do sgcWebSockets, e adicione o Project Id (NÃO o nome do projeto)

Select a project



NEW PROJECT

RECENT

ALL

Name		ID
✓ 	PubSub 	pubsub-270909
	GmailApiTest 	gmailapitest-265010
	My Project 	melodic-voice-265009

CANCEL

OPEN

É assim que deve ser configurado no exemplo PubSub do sgcWebSocket.


Google Pub/Sub API Client

Google API

Client Id

843483347040-ehcspkfsp4180rlbdfoe6mc32e3ncmn0.app

Client Secret

pvogD9reEOt9iLL6eR1jE60Z

Publisher

Project-id

pubsub-270909

Topic

topic-1

Create Topic

Delete Topic

Get Topic

List Topics

Publish

Message sent from sgcWebSockets Pub/Sub API.

Subscriber

Project-id

Subscription

subscription-1

Topic

topic-1

Create Subs.

Delete Subs.

Get Subs.

List Subs.

Pull

Acknowledge

Em seguida, você pode tentar criar um novo tópico, por exemplo. Na primeira vez, você deve autorizar a conexão OAuth2, então um novo navegador web será exibido para solicitar autorização de acesso à sua conta com as credenciais OAuth2 fornecidas pela 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](#)

Permita a conexão e, se for bem-sucedida, você pode começar a trabalhar com esta API.

Google Pub/Sub API Client

Google API

Client Id

>4180rlbdf6mc32e3ncmn0.apps.googleusercontent.com

Client Secret

pvogD9reEOt9iLL6eR1jE60Z

Publisher

Project-id

pubsub-270909

Topic

topic-1

Create Topic

Delete Topic

Get Topic

List Topics

Publish

Message sent from sgcWebSockets Pub/Sub API.

Subscriber

Project-id

Subscription

subscription-1

Topic

topic-1

Create Subs.

Delete Subs.

Get Subs.

List Subs.

Pull

Acknowledge

#CreateTopic: { "name": "projects/pubsub-270909/topics/topic-1" }

#Publish: { "messageIds": ["1780785665748120"] }

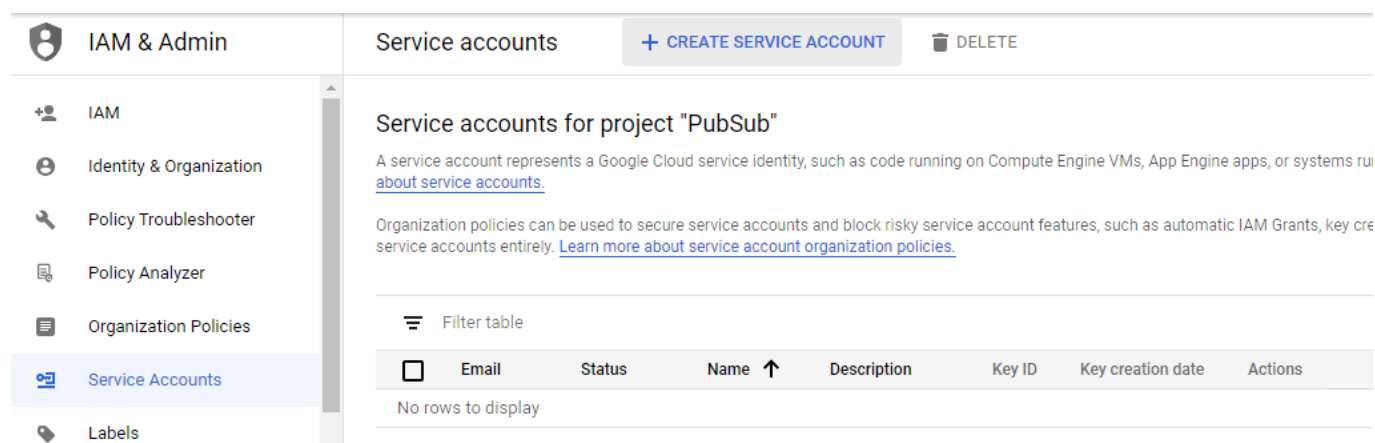
#Publish: { "messageIds": ["1780799518708503"] }

Google Cloud | Google Service Accounts

Para utilizar os componentes Google Cloud do sgcWebSockets e autenticar utilizando Service Accounts, primeiro você deve obter o Private Key Certificate do Google Cloud.

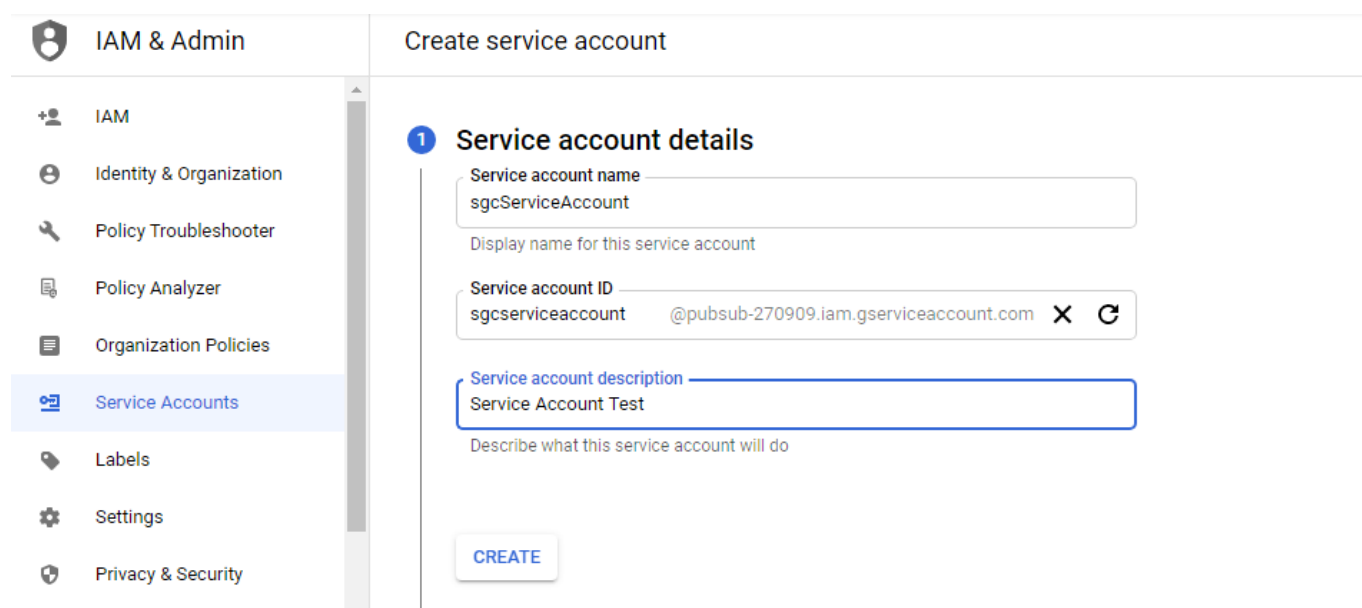
Encontre abaixo os passos para obter o Google Private Key Certificate e como configurá-lo em nossa aplicação de exemplo PubSub.

Primeiro **faça login** na sua **Google Cloud Account** e utilize um projeto existente ou crie um novo.



The screenshot shows the Google Cloud IAM & Admin console. On the left, the 'Service Accounts' menu item is selected. The main panel displays 'Service accounts for project "PubSub"'. It includes a '+ CREATE SERVICE ACCOUNT' button and a 'DELETE' button. Below this, there is explanatory text about service accounts and a table with columns: Email, Status, Name, Description, Key ID, Key creation date, and Actions. The table currently shows 'No rows to display'.

Selecione **CREATE SERVICE ACCOUNT** e uma nova página será exibida, onde você deve definir o nome e a descrição da conta de serviço



The screenshot shows the 'Create service account' page in the Google Cloud IAM & Admin console. The left sidebar shows the 'Service Accounts' menu item selected. The main panel is titled 'Create service account' and contains a '1 Service account details' section. This section has three input fields: 'Service account name' (filled with 'sgcServiceAccount'), 'Service account ID' (filled with 'sgcserviceaccount' and '@pubsub-270909.iam.gserviceaccount.com'), and 'Service account description' (filled with 'Service Account Test'). A 'CREATE' button is located at the bottom of the form.

Em seguida, selecione pelo menos um papel. Aqui selecionamos PubSub Admin para permitir que o cliente publique e assine tópicos, mas você pode selecionar outro papel com menos privilégios.

IAM & Admin

- IAM
- Identity & Organization
- Policy Troubleshooter
- Policy Analyzer
- Organization Policies
- Service Accounts**
- Labels
- Settings
- Privacy & Security
- Identity-Aware Proxy

Create service account

- Service account details
- Grant this service account access to project (optional)**

Grant this service account access to PubSub so that it has permission to complete specific actions on the resources in your project. [Learn more](#)

Role

Pub/Sub Admin

Condition
[Add condition](#)

Full access to topics, subscriptions, and snapshots.

[+ ADD ANOTHER ROLE](#)

CONTINUE

Pressione CONTINUE e, por fim, você pode conceder acesso a outros usuários

IAM & Admin

- IAM
- Identity & Organization
- Policy Troubleshooter
- Policy Analyzer
- Organization Policies
- Service Accounts**
- Labels
- Settings
- Privacy & Security
- Identity-Aware Proxy
- Roles

Create service account

- Service account details
- Grant this service account access to project (optional)
- Grant users access to this service account (optional)**

Grant access to users or groups that need to perform actions as this service account. [Learn more](#)

Service account users role

Grant users the permissions to deploy jobs and VMs with this service account

Service account admins role

Grant users the permission to administer this service account

DONE CANCEL

Pressione DONE quando terminar, e um novo registro será exibido.

IAM & Admin
IAM
Identity & Organization
Policy Troubleshooter
Policy Analyzer
Organization Policies
Service Accounts
Labels

Service accounts
+ CREATE SERVICE ACCOUNT
DELETE

Service accounts for project "PubSub"

A service account represents a Google Cloud service identity, such as code running on Compute Engine VMs, App Engine apps, or system [about service accounts](#).

Organization policies can be used to secure service accounts and block risky service account features, such as automatic IAM Grants, ke service accounts entirely. [Learn more about service account organization policies](#).

Filter table

☐	Email	Status	Name ↑	Description	Key ID
☐	sgcserviceaccount@pubsub-270909.iam.gserviceaccount.com	✓	sgcServiceAccount	Service Account Test	No keys

O próximo passo é criar uma nova chave, portanto selecione a opção Create Key na coluna de ações. Selecione JSON para baixar a configuração em formato JSON, e uma nova chave será criada.

Service accounts for project "PubSub"

A service account represents a Google Cloud service identity, such as code running on Compute Engine VMs, App Engine apps, or systems running outside Google. [Learn more about service accounts](#).

Organization policies can be used to secure service accounts and block risky service account features, such as automatic IAM Grants, key creation/upload, or the creation of service accounts entirely. [Learn more about service account organization policies](#).

Filter table							
☐	Email	Status	Name ↑	Description	Key ID	Key creation date	Actions
☐	sgcserviceaccount@pubsub-270909.iam.gserviceaccount.com	✓	sgcServiceAccount	Service Account Test	425a876f68b8b2a66f2fcc5b2dee8e918b0eb9ab	Dec 20, 2020	⋮

Por fim, você só precisa preencher os dados fornecidos pelo Google no cliente PubSub do sgcWebSockets. Você pode usar **LoadSettingsFromFile** para carregar o arquivo JSON de configuração.

Google Pub/Sub API Client

OAuth2

Service Account

Client Email:

sgcserviceaccount@pubsub-270909.iam.gserviceaccount.com

Private Key Id:

425a876f68b8b2a66f2fcc5b2dee8e918b0eb9ab

Private Key:

-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBCwggSjAgEAAoIBA
QC3xz0L2Wkr9Hvj
HOMMf5Qum78Ml1c8nVcF8HlQVc8amSLV

Load JSON
Settings

Publisher

Project-id

pubsub-270909

Topic

topic-1

Create Topic

Delete Topic

Get Topic

List Topics

Publish

Message sent from sgcWebSockets Pub/Sub API.

Subscriber

Project-id

pubsub-270909

Subscription

subscription-1

Topic

topic-1

Create Subs.

Delete Subs.

Get Subs.

List Subs.

Pull

Acknowledge

Google Cloud | Pub/Sub

O que é o Google Cloud Pub/Sub?

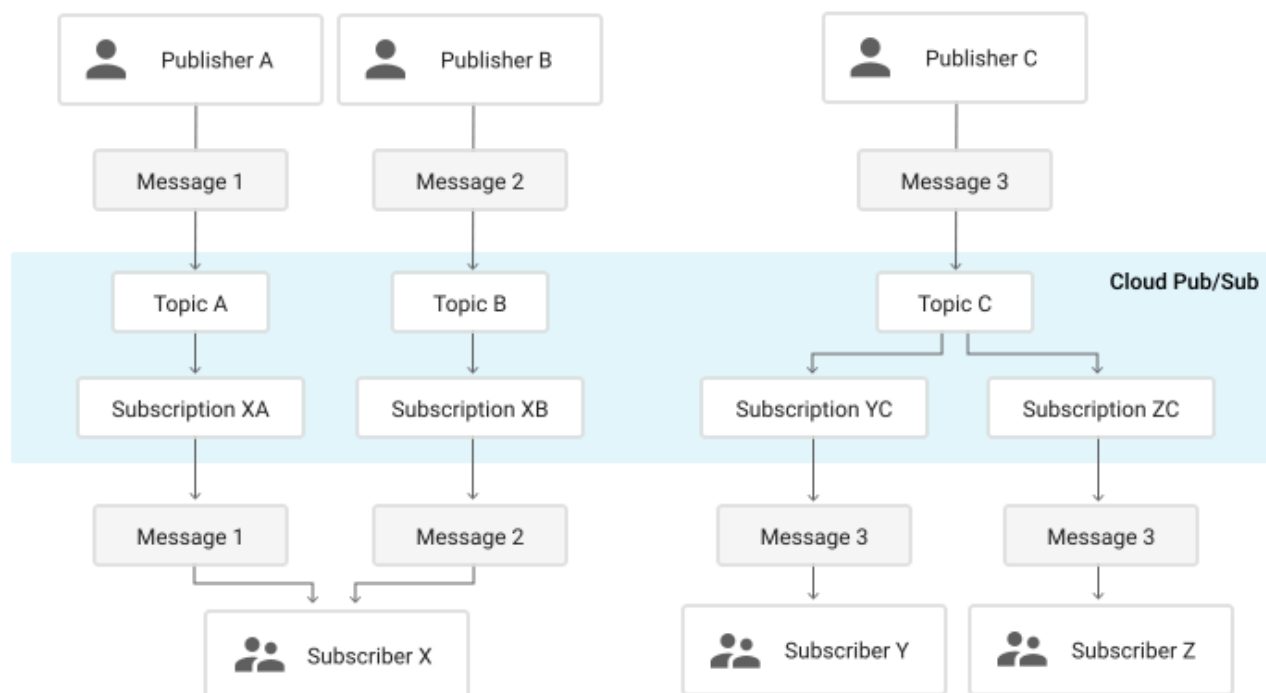
O Pub/Sub traz a flexibilidade e a confiabilidade do middleware corporativo orientado a mensagens para a nuvem. Ao mesmo tempo, o Pub/Sub é um sistema escalável e durável de ingestão e entrega de eventos que serve como base para pipelines modernos de stream analytics. Ao fornecer mensageria assíncrona muitos-para-muitos que desacopla remetentes e receptores, ele permite comunicação segura e de alta disponibilidade entre aplicações escritas de forma independente. O Pub/Sub entrega mensageria durável e de baixa latência que ajuda os desenvolvedores a integrar rapidamente sistemas hospedados na Google Cloud Platform e externamente.

Features

Entrega at-least-once A replicação síncrona de mensagens entre zonas e o rastreamento de recibo por mensagem garantem a entrega at-least-once em qualquer escala.	Open APIs abertas e bibliotecas de cliente em sete linguagens suportam implantações cross-cloud e híbridas.	Processamento exactly-once O Cloud Dataflow suporta processamento confiável, expressivo e exactly-once de streams do Cloud Pub/Sub.
Global por padrão Publique de qualquer lugar do mundo e consuma de qualquer lugar, com latência consistente. Nenhuma replicação necessária.	Sem provisionamento, tudo automático O Cloud Pub/Sub não tem shards nem partições. Basta definir sua cota, publicar e consumir.	Conformidade e segurança O Cloud Pub/Sub é um serviço em conformidade com a HIPAA, oferecendo controles de acesso granulares e criptografia de ponta a ponta.
Integrado Aproveite as integrações com múltiplos serviços, como eventos de atualização do Cloud Storage e do Gmail e Cloud Functions para computação serverless orientada a eventos.	Seek and replay Retroceda seu backlog para qualquer ponto no tempo ou um snapshot, dando a capacidade de reprocessar as mensagens. Avance rapidamente para descartar dados desatualizados.	

Relações publisher-subscriber

Uma aplicação publicadora cria e envia mensagens para um tópico. As aplicações assinantes criam uma assinatura em um tópico para receber mensagens dele. A comunicação pode ser de um-para-muitos (fan-out), muitos-para-um (fan-in) e muitos-para-muitos.



Casos de uso comuns

- **Balanceamento de cargas de trabalho em clusters de rede.** Por exemplo, uma grande fila de tarefas pode ser distribuída eficientemente entre vários workers, como instâncias do Google Compute Engine.
- **Implementando fluxos de trabalho assíncronos.** Por exemplo, uma aplicação de processamento de pedidos pode registrar um pedido em um tópico, a partir do qual ele pode ser processado por um ou mais workers.
- **Distribuição de notificações de eventos.** Por exemplo, um serviço que aceita cadastros de usuários pode enviar notificações sempre que um novo usuário se registra, e os serviços downstream podem se inscrever para receber notificações do evento.
- **Atualização de caches distribuídos.** Por exemplo, uma aplicação pode publicar eventos de invalidação para atualizar os IDs de objetos que foram alterados.
- **Registro em múltiplos sistemas.** Por exemplo, uma instância do Google Compute Engine pode gravar logs no sistema de monitoramento, em um banco de dados para consulta posterior, e assim por diante.
- **Streaming de dados de vários processos ou dispositivos.** Por exemplo, um sensor residencial pode transmitir dados para servidores backend hospedados na nuvem.
- **Melhoria de confiabilidade.** Por exemplo, um serviço Compute Engine de zona única pode operar em zonas adicionais fazendo subscribe a um tópico comum, para se recuperar de falhas em uma zona ou região.

Autorização

O cliente Google Pub/Sub pode fazer login nos servidores do Google usando os seguintes métodos:

- **gcaOAuth2:** protocolo OAuth2
- **gcaJWT:** tokens JWT.

OAuth2

O login é feito usando um navegador web onde o usuário faz login com sua própria conta e autoriza as requisições PubSub.

- **GoogleCloudOptions.OAuth2.ClientId:** é o ClientID fornecido pela Google para autenticar por meio do protocolo OAuth2.
- **GoogleCloudOptions.OAuth2.ClientSecret:** é a string Client Secret fornecida pela Google para autenticar através do protocolo OAuth2.
- **GoogleCloudOptions.OAuth2.Scope:** é o escopo do OAuth2; geralmente não há necessidade de modificar o valor padrão, a menos que você precise obter mais acesso do que o padrão.
- **GoogleCloudOptions.OAuth2.LocalIP:** o protocolo OAuth2 requer um servidor local escutando a resposta do servidor de autenticação. Este é o IP ou nome DNS. Por padrão, é 127.0.0.1.
- **GoogleCloudOptions.OAuth2.LocalPort:** Porta de escuta do servidor local.
- **GoogleCloudOptions.OAuth2.RedirectURL:** se você precisar definir uma URL de redirecionamento diferente de LocalPort + LocalIP, pode defini-la nesta propriedade (exemplo: `http://127.0.0.1:8080/oauth2`).

Service Accounts

O login é feito assinando as requisições usando uma chave privada fornecida pelo Google. Este método é recomendado para serviços automatizados ou aplicações sem interação do usuário.

- **GoogleCloudOptions.JWT.ClientEmail:** o nome do e-mail do cliente fornecido ao criar a nova conta de serviço. Nó "client_email" no arquivo de configuração JSON.
- **GoogleCloudOptions.JWT.PrivateKeyId:** o Private Key ID fornecido pela Google. Nó "private_key_id" no arquivo de configuração JSON.
- **GoogleCloudOptions.JWT.PrivateKey:** o certificado de chave privada fornecido pelo Google. Nó "private_key" no arquivo de configuração JSON.

TLS Options

TLSOptions centraliza a configuração do canal seguro utilizado para comunicação com o Google Cloud Pub/Sub.

- **TLSOptions.IOHandler:** escolha a implementação TLS que se adequa à sua implantação (OpenSSL, SChannel...).
- **TLSOptions.Version:** restringe a negociação a uma determinada versão TLS quando sua infraestrutura exige.
- **TLSOptions.VerifyCertificate:** controla a validação de certificados, habilitando políticas de segurança mais rígidas.
- **TLSOptions.OpenSSL_Options.LibPath:** defina o diretório que contém os binários do OpenSSL distribuídos com sua aplicação.
- **TLSOptions.SChannel_Options.UseLegacyCredentials:** ativa o tratamento legado de credenciais do Windows quando o SChannel o exige.

Exemplo:

```
TsgcHTTPGoogleCloud_PubSub_Client *oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client(NULL);
oPubSub->TLSOptions->IOHandler = ioHandlerOpenSSL;
oPubSub->TLSOptions->Version = tls1_3;
oPubSub->TLSOptions->VerifyCertificate = true;
oPubSub->TLSOptions->OpenSSL_Options->LibPath = oslpDefaultFolder;
```

Quando uma nova conta de serviço é criada, você pode baixar um arquivo JSON com todas as configurações. Este arquivo pode ser processado pelo componente PubSub; basta chamar o método **LoadSettingsFromFile** e passar o nome do arquivo JSON como argumento.

Most common uses

- **Configuration**
 - [Google OAuth2 Keys](#)
 - [Service Accounts](#)

Google Pub/Sub Client

OAuth2

Para trabalhar com a API Google Pub/Sub, o componente Pub/Sub do `sgcWebSockets` utiliza OAuth2 como autenticação padrão, portanto primeiro você deve definir seu **ClientId** e **ClientSecret** da sua conta 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...";
```

Service Accounts

As Service Accounts exigem a construção de um JWT e seu envio como token de autorização.

```
TsgcHTTPGoogleCloud_PubSub_Client *oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client(this);
oPubSub->GoogleCloudOptions->Authorization = gcaJWT;
oPubSub->GoogleCloudOptions->JWT->ClientEmail = "...google email...";
oPubSub->GoogleCloudOptions->JWT->PrivateKeyId = "...private key id...";
oPubSub->GoogleCloudOptions->JWT->PrivateKey->Lines->Text = "...private key certificate...";
```

Isso é necessário para obter uma Authorization Token Key do Google que será utilizada para todas as chamadas da Rest API.

Todos os métodos retornam uma resposta, que pode ser bem-sucedida ou retornar um erro.

Projects.Snapshots

Method	Parâmetros	Descrição	Exemplo
CreateSnapshot	project, snapshot, subscription	Cria um snapshot a partir da assinatura solicitada. Os snapshots são utilizados em operações <code>subscriptions.seek</code> , que permitem gerenciar confirmações de mensagens em massa. Ou seja, você pode definir o estado de confirmação das mensagens em uma assinatura existente para o estado capturado por um snapshot.	<code>CreateSnapshot('pubsub-270909', 'snapshot-1', 'subscription-1')</code>
DeleteSnapshot	project, snapshot	Remove um snapshot existente	<code>DeleteSnapshot('pubsub-270909', 'snapshot-1')</code>

ListSnapshots	project	Lista os snapshots existentes	ListSnapshots('pubsub-270909')

Projects.Subscriptions

Method	Parâmetros	Descrição	Exemplo
Acknowledge-Subscription			
Create-Subscription	project, subscription, topic	Cria uma assinatura para um determinado tópico. Se a assinatura já existir, retorna ALREADY_EXISTS. Se o tópico correspondente não existir, retorna NOT_FOUND.	CreateSubscription('pubsub-270909', 'subscription-1', 'topic-1')
Delete-Subscription	project, subscription	Exclui uma assinatura existente. Todas as mensagens retidas na assinatura são imediatamente descartadas.	DeleteSubscription('pubsub-270909', 'subscription-1')
Get-Subscription	project, subscription	Obtém os detalhes de configuração de uma subscription.	GetSubscription('pubsub-270909', 'subscription-1')
List-Subscriptions	project	Lista as inscrições correspondentes.	ListSubscriptions('pubsub-270909', 'subscription-1')
ModifyAck-Deadline-Subscription	project, subscription, AckIds	Modifica o ack deadline de uma mensagem específica. Este método é útil para indicar que mais tempo é necessário para processar uma mensagem pelo subscriber, ou para tornar a mensagem disponível para reentrega se o	

		processamento foi interrompido. Note que isto não modifica o <code>ackDeadlineSeconds</code> no nível da assinatura utilizado para mensagens subsequentes.	
Modify-Push-Config-Subscription	project, subscription	Modifica a Push-Config de uma assinatura especificada. Isso pode ser usado para mudar uma assinatura push para pull (indicada por uma PushConfig vazia) ou vice-versa, ou para alterar a URL do endpoint e outros atributos de uma assinatura push. As mensagens continuarão se acumulando para entrega ao longo da chamada, independentemente de mudanças na PushConfig.	
Pull	project, subscription	Extraí mensagens do servidor. O servidor pode retornar UNAVAILABLE se houver muitas requisições de pull simultâneas pendentes para a assinatura informada.	<code>pull('pubsub-270909', 'subscription-1')</code>
Seek	project, subscription, timeUTC, snapshot	Reposiciona uma assinatura existente para um ponto no tempo ou para um snapshot informado, o que for fornecido na requisição. Snapshots são utilizados em operações <code>subscriptions.seek</code> , que permitem gerenciar confirmações de mensa-	

gens em massa. Ou seja, você pode definir o estado de confirmação das mensagens em uma assinatura existente para o estado capturado por um snapshot. Observe que tanto a assinatura quanto o snapshot devem estar no mesmo tópico.

Projects.Topics

Method	Parâmetros	Descrição	Exemplo
Create-Topic	project, topic	Cria o tópico fornecido com o nome fornecido	<code>CreateTopic('pubsub-270909', 'topic-1')</code>
Delete-Topic	project, topic	Exclui o tópico com o nome fornecido. Retorna NOT_FOUND se o tópico não existir. Após um tópico ser excluído, um novo tópico pode ser criado com o mesmo nome; este é um tópico totalmente novo, sem nenhuma das configurações ou assinaturas antigas.	<code>DeleteTopic('pubsub-270909', 'topic-1')</code>
GetTopic	project, topic	Obtém a configuração de um tópico.	<code>GetTopic('pubsub-270909', 'topic-1')</code>
ListTopics	project	Lista os tópicos correspondentes.	<code>ListTopics('pubsub-270909')</code>
Publish	project, topic, message	Adiciona uma ou mais mensagens ao tópico. Retorna NOT_FOUND se o tópico não existir.	<code>Publish('pubsub-270909', 'topic-1', 'My First PubSub Message.')</code>

Projects.Topics.Subscriptions

Method	Parâmetros	Descrição	Exemplo
ListTopicSubscriptions	project, topic	Lista os nomes das assinaturas deste tópico.	ListTopicSubscriptions('pubsub-270909', 'topic-1')

Métodos mais comuns

Encontre abaixo os métodos mais comuns utilizados com a Google Cloud Pub/Sub API.

Como criar um novo Topic

Cria um novo tópico para o projeto com id: pubsub-270909 e nome de tópico 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");
```

Resposta do Servidor

```
{
  "name": "projects/pubsub-270909/topics/topic-1"
}
```

Publicar uma mensagem

Publica uma nova mensagem no tópico recém-criado.

```
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 sgWebSockets.");
```

Resposta do Servidor

```
{
  "messageIds": [
    "1050732082561505"
  ]
}
```

Publish a Message with Attributes

```
TsgcHTTPGoogleCloud_PubSub_Client *oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client();
oPubSub->GoogleCloudOptions->OAuth2->ClientId = "... your google client id...";
```

```
oPubSub->GoogleCloudOptions->OAuth2->ClientSecret = "... your google client secret...";
oAttributes = new TStringList();
try
{
    oAttributes->CommaText = "origin=gcloud-sample,username=gcp";
    oPubSub->Publish("pubsub-270909", "topic-1", "My First Message from sgcWebSockets.", oAttributes, "username"));
}
__finally
{
    oAttributes->Free();
}
```

Como criar uma nova assinatura

Cria uma nova assinatura para o projeto com id: pubsub-270909, com o nome de assinatura subscription-1 e o tópico 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");
```

Resposta do Servidor

```
{
  "name": "projects/pubsub-270909/subscriptions/subscription-1",
  "topic": "projects/pubsub-270909/topics/topic-1",
  "pushConfig": {},
  "ackDeadlineSeconds": 10,
  "messageRetentionDuration": "604800s",
  "expirationPolicy": {
    "ttl": "2678400s"
  }
}
```

Como Ler mensagens de uma Subscription

Lê mensagens da inscrição anterior criada.

```
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");
```

Resposta do Servidor

```
{
  "receivedMessages": [
    {
      "ackId": "PjA-RVNEUAYWLF1GSFE3GQhoUQ5PXiM_NSAoRREFC08CKF15MEorQVh0Dj4N",
      "message": {
        "data": "TXkgRmlyc3QgTWVzc2FnZSBmcm9tIHNnY1dlY1NvY2tldHMu",
        "messageId": "1050732082561505",
        "publishTime": "2020-03-14T15:25:31.505Z"
      }
    }
  ]
}
```

A mensagem é recebida codificada em Base64, portanto você deve decodificar primeiro para ler o conteúdo.

```
sgcBase_Helpers->DecodeBase64("TXkgRmlyc3QgTWVzc2FnZSBmcm9tIHNnY1dlY1NvY2tldHMu=");
```

Google Cloud | Calendar

A Google Calendar API permite que você integre seu app ao Google Calendar, criando novas formas de engajar seus usuários. A Calendar API permite que você exiba, crie e modifique eventos de calendário, bem como trabalhe com muitos outros objetos relacionados a calendários, como calendários ou controles de acesso.

API Resources

O Google Calendar utiliza os seguintes recursos:

- **Event:** Um evento em um calendário contendo informações como o título, horários de início e término e participantes. Os eventos podem ser eventos únicos ou eventos recorrentes. Um evento é representado por um recurso Event. A coleção Events de um determinado calendário contém todos os recursos de evento desse calendário.
- **Calendar:** Um calendário é uma coleção de eventos. Cada calendário possui metadados associados, como a descrição do calendário ou o fuso horário padrão do calendário. Os metadados de um único calendário são representados por um recurso Calendar. A coleção Calendars contém recursos Calendar para todos os calendários existentes.
- **CalendarList:** Uma lista de todos os calendários na lista de calendários de um usuário na UI do Calendar. Os metadados de um único calendário que aparece na lista de calendários são representados por um recurso CalendarListEntry. Esses metadados incluem propriedades específicas do usuário do calendário, como sua cor ou notificações de novos eventos. A coleção CalendarList contém todos os recursos CalendarListEntry para um determinado usuário. Para uma explicação mais detalhada da diferença entre as coleções Calendars e CalendarList, consulte Calendar e Calendar List.
- **Setting:** Uma preferência de usuário da UI do Calendar, como o fuso horário do usuário. Uma única preferência de usuário é representada por um Setting Resource. A coleção Settings contém todos os Setting resources de um determinado usuário.
- **ACL:** Uma regra de controle de acesso que concede a um usuário (ou a um grupo de usuários) um nível especificado de acesso a um calendário. Uma única regra de controle de acesso é representada por um recurso ACL. A coleção ACL de um determinado calendário contém todos os recursos ACL que concedem acesso a esse calendário.
- **Color:** Uma cor apresentada na interface do Calendar. O recurso Colors representa o conjunto de todas as cores disponíveis na interface do Calendar, em dois grupos: cores disponíveis para eventos e cores disponíveis para calendários.
- **Free/busy:** Um horário em que um calendário tem eventos agendados é considerado "busy"; um horário em que um calendário não tem eventos é considerado "free". O recurso Freebusy permite consultar o conjunto de horários ocupados de um determinado calendário ou conjunto de calendários.

Principais recursos

- API de Cliente Google Calendar V3 com Recursos Completos.
- Todos os Métodos suportados pela API podem ser chamados utilizando a client API.
- Requisições do cliente utilizando o protocolo HTTP/2 (*apenas na edição Enterprise).
- Tratamento automático de respostas parciais utilizando PageNextToken.
- Acesso fácil às propriedades de dados de Calendar e Event.
- Métodos de autenticação:
 - OAuth2: requer interação do usuário.
 - Service Accounts (requer Domain-Wide Delegation): para serviços windows, daemons...

Configuração

O cliente do componente Google Calendar tem as seguintes propriedades:

OAuth2

- **GoogleCloudOptions.OAuth2.ClientId:** é o ClientID fornecido pela Google para autenticar por meio do protocolo OAuth2.

- **GoogleCloudOptions.OAuth2.ClientSecret:** é a string Client Secret fornecida pela Google para autenticar através do protocolo OAuth2.
- **GoogleCloudOptions.OAuth2.Scope:** é o escopo do OAuth2; geralmente não há necessidade de modificar o valor padrão, a menos que você precise obter mais acesso do que o padrão.
- **GoogleCloudOptions.OAuth2.LocalIP:** o protocolo OAuth2 requer um servidor local escutando a resposta do servidor de autenticação. Este é o IP ou nome DNS. Por padrão, é 127.0.0.1.
- **GoogleCloudOptions.OAuth2.LocalPort:** Porta de escuta do servidor local.
- **GoogleCloudOptions.OAuth2.RedirectURL:** se você precisar definir uma URL de redirecionamento diferente de LocalPort + LocalIP, pode defini-la nesta propriedade (exemplo: <http://127.0.0.1:8080/oauth2>).

Você pode modificar os Scopes da sua client API usando a propriedade Scopes, basta selecionar quais escopos são suportados pelo seu cliente.

JWT

O login é feito assinando as requisições utilizando uma chave privada fornecida pelo google, este método é recomendado para serviços automatizados ou aplicações sem interação do usuário. Requer configurar a Service Account com [Domain-Wide Delegation](#).

- **GoogleCloudOptions.JWT.ClientEmail:** o nome do e-mail do cliente fornecido ao criar a nova conta de serviço. Nó "client_email" no arquivo de configuração JSON.
- **GoogleCloudOptions.JWT.PrivateKeyId:** o Private Key ID fornecido pela Google. Nó "private_key_id" no arquivo de configuração JSON.
- **GoogleCloudOptions.JWT.PrivateKey:** o certificado de chave privada fornecido pelo Google. Nó "private_key" no arquivo de configuração JSON.
- **GoogleCloudOptions.JWT.Subject:** é a conta de email do workspace vinculada à conta de serviço utilizando Domain-Wide Delegation.

TLS Options

Utilize a propriedade **TLSOptions** para personalizar a conexão segura estabelecida com os servidores Google.

- **TLSOptions.IOHandler:** seleciona a pilha TLS (OpenSSL, SChannel...).
- **TLSOptions.Version:** força uma versão específica do protocolo TLS (por exemplo, `tls1_2` ou `tls1_3`).
- **TLSOptions.VerifyCertificate:** habilita ou desabilita a validação do certificado do servidor.
- **TLSOptions.OpenSSL_Options.LibPath:** aponta para a pasta que contém as bibliotecas OpenSSL implantadas com a aplicação.
- **TLSOptions.SChannel_Options.UseLegacyCredentials:** habilita o fluxo de credenciais legado exigido por algumas versões do Windows ao utilizar o SChannel.

Os trechos a seguir mostram como configurar as opções de TLS:

```
TsgcHTTPGoogleCloud_Calendar_Client *oCalendar = new TsgcHTTPGoogleCloud_Calendar_Client(NULL);
oCalendar->TLSOptions->IOHandler = ioHandlerOpenSSL;
oCalendar->TLSOptions->Version = tls1_3;
oCalendar->TLSOptions->VerifyCertificate = true;
oCalendar->TLSOptions->OpenSSL_Options->LibPath = opensslDefaultFolder;
```

Most common uses

- **Configuração**
 - [Google Calendar Refresh Token](#)
 - [Google Calendar Service Account](#)
- **Synchronization**
 - [Google Calendar Sync Calendars](#)
 - [Google Calendar Sync Events](#)

Sincronizar Calendários

O componente `TsgcHTTPGoogleCloud_Calendar_Client` permite que você sincronize os calendários utilizando chamadas diretas à API do Google ou utilizando nossos métodos `Calendars` fáceis para sincronizar os calendários.

Method	Parâmetros	Descrição
NewCalendar	aSummary: o título do calendário.	Cria um novo Calendar
DeleteCalendar	aid: identificador do calendário.	Exclui um Calendar existente.
UpdateCalendar	aResource: objeto com os dados do calendário.	Atualiza um Calendário existente.
LoadCalendars		Carrega todos os calendários, e a propriedade <code>Calendars</code> é preenchida com esses dados.
LoadCalendarsChanged	aSyncToken: último token utilizado para atualizar seu calendário.	Carrega todas as alterações nos seus calendários a partir do Token definido.

O Cliente `Calendar` tem uma propriedade chamada **Calendars**, onde você pode acessar os dados do calendário após chamar qualquer um dos métodos anteriores. Esta propriedade é sincronizada automaticamente.

Sincronizar eventos

O componente `TsgcHTTPGoogleCloud_Calendar_Client` permite que você sincronize os eventos utilizando chamadas diretas da Google API ou utilizando nossos métodos `Event` fáceis para sincronizar os Eventos.

Method	Parâmetros	Descrição
NewEvent	aCalendarId: id da agenda. aResource: objeto com os dados do evento.	Cria um novo Event.
DeleteEvent	aCalendarId: id do calendário. aid: identificador do evento.	Exclui um Event existente.
UpdateEvent	aCalendarId: id do calendário. aResource: objeto com os dados do evento.	Atualiza um Event existente.
LoadEvents	aCalendarId: id do calendário.	Carrega todos os eventos do calendário.
LoadEventsChanged	aCalendarId: id do calendário. aSyncToken: último	Carrega todos os eventos do calendário a partir do Token definido.

	mo token utiliza- do para atualizar seu calendário.	

Você pode acessar os dados de eventos utilizando a propriedade **Calendars**. Selecione qualquer um dos calendários existentes da lista e acesse sua propriedade **Events**.

Google Calendar API Calls

Method	Descrição
ACL_Delete	Exclui uma regra de controle de acesso.
ACL_Get	Retorna uma regra de controle de acesso.
ACL_Insert	Cria uma regra de controle de acesso.
ACL_List	Retorna as regras na lista de controle de acesso do calendário.
ACL_Patch	Atualiza uma regra de controle de acesso. Este método suporta semântica de patch.
ACL_Update	Atualiza uma regra de controle de acesso.
ACL_Watch	Observa (watch) alterações em recursos ACL.

Method	Descrição
CalendarList_Delete	Remove um calendário da lista de calendários do usuário.
CalendarList_Get	Retorna um calendário da lista de calendários do usuário.
CalendarList_Insert	Insere um calendário existente na lista de calendários do usuário.
CalendarList_List	Retorna os calendários na lista de calendários do usuário.
CalendarList_Patch	Atualiza um calendário existente na lista de calendários do usuário. Este método suporta a semântica patch.
CalendarList_Update	Atualiza um calendário existente na lista

	de calendários do usuário.
CalendarList_Watch	Observa mudanças nos recursos CalendarList.

Method	Descrição
Calendar_Clear	Limpa uma agenda primária. Esta operação exclui todos os eventos associados à agenda primária de uma conta.
Calendar_Delete	Exclui um calendário secundário. Utilize calendars.clear para limpar todos os eventos em calendários primários.
Calendar_Get	Retorna os metadados de um calendário.
Calendar_Insert	Cria um calendário secundário.
Calendar_Patch	Atualiza os metadados de uma agenda. Este método suporta semântica de patch.
Calendar_Update	Atualiza os metadados de um calendário.

Method	Descrição
Channel_Stop	Parar de observar recursos através deste canal.

Method	Descrição
Color_Get	Retorna as definições de cores para calendários e eventos.

Method	Descrição
Event_Delete	Exclui um evento.
Event_Get	Retorna um evento.
Event_Import	Importa um evento. Esta operação é usada para adicionar uma cópia privada de um evento existente a um calendário.

Event_Insert	Cria um evento.
Event_Instances	Retorna instâncias do evento recorrente especificado.
Event_List	Retorna eventos no calendário especificado.
Event_Move	Movimenta um evento para outro calendário, ou seja, altera o organizador de um evento.
Event_Patch	Atualiza um evento. Este método suporta a semântica de patch. Os valores de campo que você especifica substituem os valores existentes. Os campos que você não especifica na requisição permanecem inalterados. Os campos de array, se especificados, sobrescrevem os arrays existentes; isso descarta quaisquer elementos de array anteriores.
Event_QuickAdd	Cria um evento baseado em uma string de texto simples.
Event_Update	Atualiza um evento.
Event_Watch	Monitora alterações em recursos de Events.

Method	Descrição
Freebusy_Query	Retorna informações de free/busy para um conjunto de calendários.

Method	Descrição
Settings_Get	Retorna uma única configuração de usuário.
Settings_List	Retorna todas as configurações de usuário do usuário autenticado.
Settings_Watch	Monitora alterações em recursos Settings.

Trocar Conta

Se você quiser alternar de uma conta para outra, primeiro chame o método **Clear**, para apagar a sessão de calendário atual e configurar a nova sessão.

Google Calendar | Carregar Calendários

O processo para obter todos os calendários da sua conta é muito fácil. Basta seguir os próximos passos:

1. Chame o método **LoadCalendars**.
2. Se o método retornar True, então você pode acessar a propriedade **Calendars** e iterar sobre a lista para obter acesso a todos os Calendars.

```
TsgcHTTPGoogleCloud_Calendar_Client oGoogleCalendar = new TsgcHTTPGoogleCloud_Calendar_Client(NULL);  
// ... 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++)  
    {  
        vCalendarTitle = oGoogleCalendar->Calendars->Calendar[i]->Summary;  
    }  
}  
else  
{  
    throw Exception("Error Calendar Sync");  
}
```

Google Calendar | Sync Events

O processo para obter todos os eventos de um calendário é muito fácil. Basta seguir os próximos passos:

1. Chame o método **LoadEvents** e passe o **CalendarId** como parâmetro.
2. Se o método retornar True, então você pode acessar a propriedade **Calendars.Events** e iterar pela lista para obter acesso a todos os Events do calendário.

```
TsgcHTTPGoogleCloud_Calendar_Client oGoogleCalendar = new TsgcHTTPGoogleCloud_Calendar_Client(NULL);
// ... 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
oCalendar = TsgcGoogleCalendarItem(oGoogleCalendar->Calendars->Calendar[0]);
if (oGoogleCalendar->LoadEvents(oCalendar->ID))
{
    // ... get events data
    for (int i = 0; i < oCalendar->Events->Count; i++)
    {
        vEventTitle = oCalendar->Events[i]->Summary;
    }
}
else
{
    throw Exception("Error Event Sync");
}
```

Google Calendar | RefreshToken

A API do Google Calendar usa OAuth2 para autenticar nos servidores do Google. O `sgcWebSockets` possui um componente que trata todo o processo de autenticação, mas se sua aplicação fechar e você tentar conectar novamente, você tem duas opções:

1. Autentique-se novamente utilizando suas Google APIs
2. Use o Refresh Token (se ainda válido), para que você evite o processo de autenticação.

Usando RefreshToken

Na primeira vez que você autenticar, use o evento `OnAuthToken` para salvar o **RefreshToken**, se ele existir. Você pode salvá-lo em um arquivo INI, por exemplo:

```
void OnGoogleCalendarAuthToken(TObject *Sender, string TokenType, string Token, string Data)
{
    TsgcJSON *oJSON = new TsgcJSON();
    try
    {
        oJSON->Read(Data);
        if (oJSON->Node["refresh_token"] != NULL)
        {
            TINIFile *oINI = new TINIFile(ChangeFileExt(Application->ExeName, ".ini"));
            try
            {
                oINI->WriteString("OAUTH2", "Token", oJSON->Node["refresh_token"]->Value);
            }
            finally
            {
                oINI->Free();
            }
        }
    }
    finally
    {
        oJSON->Free();
    }
}
```

Então, quando você iniciar sua aplicação novamente, se houver um RefreshToken, chame o método `RefreshToken` e passe o token como argumento (você deve primeiro definir as chaves da Google Calendar API). Se for bem-sucedido, você fará login nos servidores Google sem ter que se autenticar novamente.

```
GoogleCalendar->RefreshToken("your refresh token here");
```

Google Calendar | Service Account

O cliente Google Calendar pode funcionar como um serviço sem interação do usuário, então isso é útil quando você deseja executar um serviço do Windows, um daemon, etc.

O Google Cloud requer a criação de uma **Service Account** (em vez de credenciais OAuth2) para executar este tipo de projeto, e a API do Google Calendar requer que a conta de serviço utilize a **Domain-Wide Delegation** para obter as credenciais necessárias ao acesso às agendas.

Você pode ler mais sobre como criar [Google Service Accounts](#).

Uma vez que a conta Google Cloud tenha sido configurada com uma service account e vinculada a um e-mail de conta de workspace utilizando Domain-Wide Delegation, você pode configurar o cliente Google Calendar para trabalhar com ela seguindo os próximos passos:

- Defina a propriedade **GoogleCloudOptions.Authentication** com o valor **gcaJWT**.
- Importe o arquivo JSON gerado em sua Google Cloud Account usando o método **LoadSettingsFromFile**. Este arquivo contém a chave privada para criptografar o JWT e algumas outras propriedades exigidas pelo cliente.
- Após importar o arquivo JSON, as seguintes propriedades são preenchidas automaticamente:
 - **ClientEmail**: é o nome da conta de serviço
 - **PrivateKeyId**: é o id do arquivo de chave privada
 - **PrivateKey**: é o arquivo de chave privada
- Por fim, defina na propriedade **GoogleCloudOptions.JWT.Subject** a conta de e-mail do Workspace vinculada à conta de serviço.

Após configurar o cliente, você pode começar a enviar requisições à Google Calendar API sem interação do usuário.

Google Cloud FCM

Firebase Cloud Messaging (FCM) é uma solução de mensagens multiplataforma que permite que você envie mensagens de forma confiável e sem custo.

Utilizando FCM, você pode notificar um app cliente de que novos e-mails ou outros dados estão disponíveis para sincronização. Você pode enviar mensagens de notificação para promover o reengajamento e a retenção dos usuários. Para casos de uso como mensagens instantâneas, uma mensagem pode transferir um payload de até 4096 bytes para um app cliente.

O componente suporta a HTTP v1 API.

Autorização

O cliente Google FCM pode fazer login nos servidores do Google usando os seguintes métodos:

- **gcaOAuth2:** protocolo OAuth2
- **gcaJWT:** tokens JWT.

OAuth2

O login é feito utilizando um navegador web, onde o usuário faz login com sua própria conta e autoriza as requisições do FCM.

- **GoogleCloudOptions.OAuth2.ClientId:** é o ClientID fornecido pela Google para autenticar por meio do protocolo OAuth2.
- **GoogleCloudOptions.OAuth2.ClientSecret:** é a string Client Secret fornecida pela Google para autenticar através do protocolo OAuth2.
- **GoogleCloudOptions.OAuth2.Scope:** é o escopo do OAuth2; geralmente não há necessidade de modificar o valor padrão, a menos que você precise obter mais acesso do que o padrão.
- **GoogleCloudOptions.OAuth2.LocalIP:** o protocolo OAuth2 requer um servidor local escutando a resposta do servidor de autenticação. Este é o IP ou nome DNS. Por padrão, é 127.0.0.1.
- **GoogleCloudOptions.OAuth2.LocalPort:** Porta de escuta do servidor local.
- **GoogleCloudOptions.OAuth2.RedirectURL:** se você precisar definir uma URL de redirecionamento diferente de LocalPort + LocalIP, pode defini-la nesta propriedade (exemplo: `http://127.0.0.1:8080/oauth2`).

Service Accounts

O login é feito assinando as requisições usando uma chave privada fornecida pelo Google. Este método é recomendado para serviços automatizados ou aplicações sem interação do usuário.

- **GoogleCloudOptions.JWT.ClientEmail:** o nome do e-mail do cliente fornecido ao criar a nova conta de serviço. Nó "client_email" no arquivo de configuração JSON.
- **GoogleCloudOptions.JWT.Subject:** o nome do e-mail do cliente fornecido ao criar a nova conta de serviço. Nó "client_email" no arquivo de configuração JSON.
- **GoogleCloudOptions.JWT.PrivateKeyId:** o Private Key ID fornecido pela Google. Nó "private_key_id" no arquivo de configuração JSON.
- **GoogleCloudOptions.JWT.PrivateKey:** o certificado de chave privada fornecido pelo Google. Nó "private_key" no arquivo de configuração JSON.

TLS Options

As conexões do Firebase Cloud Messaging podem ser ajustadas por meio da propriedade **TLSOptions**.

- **TLSOptions.IOHandler:** escolha a pilha TLS que será utilizada (OpenSSL, SChannel...).
- **TLSOptions.Version:** solicita uma versão específica do protocolo TLS exigida pelo seu projeto.
- **TLSOptions.VerifyCertificate:** impõe a validação do certificado do servidor ao conectar ao Google.
- **TLSOptions.OpenSSL_Options.LibPath:** define onde as bibliotecas de runtime do OpenSSL estão implantadas.
- **TLSOptions.SChannel_Options.UseLegacyCredentials:** habilite quando o SChannel do Windows exigir negociação de credenciais legada.

Exemplo de configuração:

```
TsgcHTTPGoogleCloud_FCM_Client *oFCM = new TsgcHTTPGoogleCloud_FCM_Client(NULL);
oFCM->TLSOptions->IOHandler = iohOpenSSL;
oFCM->TLSOptions->Version = tls1_3;
oFCM->TLSOptions->VerifyCertificate = true;
oFCM->TLSOptions->OpenSSL_Options->LibPath = oslpDefaultFolder;
```

Quando uma nova conta de serviço é criada, você pode baixar um arquivo JSON com todas as configurações. Este arquivo pode ser processado pelo componente FCM, basta chamar o método **LoadSettingsFromFile** e passar o nome do arquivo JSON como argumento.

Most common uses

- **Configuration**
 - [Google OAuth2 Keys](#)
 - [Service Accounts](#)

Google FCM Client

OAuth2

Para trabalhar com a API Google FCM, o componente FCM do sgcWebSockets usa OAuth2 como método de autenticação padrão, portanto primeiro você deve definir seu **ClientId** e **ClientSecret** a partir da sua conta 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...";
```

Service Accounts

As Service Accounts exigem a construção de um JWT e seu envio como token de autorização.

```
TsgcHTTPGoogleCloud_FCM_Client *oFCM = new TsgcHTTPGoogleCloud_FCM_Client(this);
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->Lines->Text = "...private key certificate...";
```

Isso é necessário para obter uma Authorization Token Key do Google que será utilizada para todas as chamadas da Rest API.

Todos os métodos retornam uma resposta, que pode ser bem-sucedida ou retornar um erro.

Enviar mensagem

Use o método **SendMessage** para enviar uma mensagem de notificação. A função tem 2 argumentos:

1. Project: é o id do projeto da sua conta do google cloud.
2. Payload: é a mensagem de texto json. Exemplo:

```
{
  "message": {
    "topic": "news",
    "notification": {
      "title": "Breaking News",
      "body": "New news story available."
    },
    "data": {
      "story_id": "story_12345"
    }
  }
}
```

TsgcWebView2

TsgcWebView2 é um componente VCL visual que encapsula o Microsoft Edge WebView2 (Chromium). Solte-o em um formulário para incorporar um navegador web moderno e completo na sua aplicação Delphi. Ele suporta Delphi 7 até Delphi 13.

Requisitos

- Microsoft Edge WebView2 Runtime (incluído com o Windows 10/11 ou disponível para download na Microsoft)
- WebView2Loader.dll (incluído, coloque ao lado do seu executável)
- Apenas Windows

Início rápido

Coloque um TsgcWebView2 em um formulário e navegue até uma URL:

```
sgcWebView21 := TsgcWebView2.Create(Self);
sgcWebView21.Parent := Self;
sgcWebView21.Align := alClient;
sgcWebView21.DefaultURL := 'https://www.example.com';
```

Features

- **Navegação**
 - [Navigate](#), [NavigateToString](#), [GoBack](#), [GoForward](#), [Reload](#), [Stop](#), [navegação POST](#)
- **JavaScript**
 - [ExecuteScript \(async\)](#), [ExecuteScriptSync](#), [AddInitScript](#)
- **Gerenciamento de cookies**
 - [GetCookies](#), [AddOrUpdateCookie](#), [DeleteCookie](#), [DeleteAllCookies](#)
- **Controle de Download**
 - [OnDownloadStarting](#), [OnDownloadProgress](#), [OnDownloadCompleted](#)
- **Settings & Configuration**
 - [ScriptEnabled](#), [DevToolsEnabled](#), [ContextMenuEnabled](#) e mais
- **Recursos Avançados**
 - [Print](#), [Audio/Mute](#), [Certificados](#), [Favicon](#), [Virtual Host](#), [Screenshot](#)

Properties

- **DefaultURL**: string — URL para navegar após a inicialização.
- **AutoInitialize**: Boolean — quando True, a WebView2 é criada automaticamente quando o handle da janela é alocado. Padrão: True.
- **ZoomFactor**: Double — nível de zoom (1.0 = 100%).
- **UserDataFolder**: string — caminho para armazenamento de cookies, cache e permissões.
- **BrowserExecutableFolder**: string — caminho para uma distribuição de versão fixa do WebView2 Runtime.
- **Settings**: TsgcWebView2Settings — configurações do navegador em tempo de design (ScriptEnabled, DevToolsEnabled, etc.).
- **URL**: string (somente leitura) — URL da página atual.
- **DocumentTitle**: string (somente leitura) — título da página atual.
- **CanGoBack**: Boolean (somente leitura) — se a navegação para trás é possível.
- **CanGoForward**: Boolean (somente leitura) — se a navegação para frente é possível.
- **Initialized**: Boolean (somente leitura) — indica se o WebView2 está pronto.
- **IsMuted**: Boolean — silenciar/reactivar áudio.
- **IsDocumentPlayingAudio**: Boolean (somente leitura) — se a página está reproduzindo áudio.
- **FaviconURI**: string (somente leitura) — URI do favicon da página atual.
- **StatusBarText**: string (somente leitura) — texto atual da barra de status.
- **CookieManager**: TsgcWebView2CookieManager (somente leitura) — objeto de gerenciamento de cookies.
- **WebView**: ICoreWebView2 (somente leitura) — acesso direto à interface COM.

- **Controller:** `ICoreWebView2Controller` (somente leitura) — acesso direto à interface COM.
- **Environment:** `ICoreWebView2Environment` (somente leitura) — acesso direto à interface COM.

Métodos

- **Navigate**(aURL: string) — navega até uma URL.
- **NavigateToString**(aHTML: string) — carrega conteúdo HTML diretamente.
- **GoBack / GoForward** — navega pelo histórico.
- **Reload / Stop** — recarrega ou para o carregamento.
- **ExecuteScript**(aScript: string) — executa JavaScript de forma assíncrona (resultado em `OnScriptExecuted`).
- **ExecuteScriptSync**(aScript: string): string — executa JavaScript de forma síncrona, retorna o resultado JSON.
- **AddInitScript**(aScript: string) — adiciona JavaScript que é executado a cada carregamento de página.
- **RemoveInitScript**(aScriptId: string) — remove um init script adicionado anteriormente.
- **NavigateWithPostData**(aURI, aMethod, aPostData, aHeaders: string) — navega com método HTTP, corpo e cabeçalhos personalizados.
- **PrintToPdf**(aFilePath: string) — salva a página como PDF.
- **ShowPrintUI** — exibe a caixa de diálogo de impressão nativa.
- **CapturePreviewToFile**(aFilePath: string; aFormat: Integer) — captura de tela (0=PNG, 1=JPEG).
- **OpenDevToolsWindow** — abre o DevTools do Edge.
- **OpenTaskManagerWindow** — abre o gerenciador de tarefas do Edge.
- **SetVirtualHostNameToFolderMapping**(aHostName, aFolderPath: string; aAccessKind: Integer) — mapeia o nome do host para uma pasta local.
- **ClearVirtualHostNameToFolderMapping**(aHostName: string) — remove o mapeamento.
- **PostWebMessageAsString**(aMessage: string) — envia mensagem ao conteúdo web.
- **PostWebMessageAsJson**(aJson: string) — envia JSON para o conteúdo web.
- **GetProfileName**: string — obtém o nome do perfil do navegador.
- **ClearBrowsingData**(aKinds: Cardinal) — limpa tipos específicos de dados de navegação.
- **ClearAllBrowsingData** — limpa todos os dados de navegação.
- **PostSharedBufferToScript**(aSharedBuffer: IUnknown; aAccess: Integer; aAdditionalDataAsJson: string) — compartilha o buffer de memória com o conteúdo web.
- **InitializeWebView / FinalizeWebView** — controle manual de inicialização.

Eventos

- **OnInitialized** — disparado quando o `WebView2` está pronto.
- **OnNavigationStarting**(aURI; alsUserInitiated, alsRedirected: Boolean; var aCancel: Boolean) — antes da navegação.
- **OnNavigationCompleted**(alsSuccess: Boolean; aWebErrorStatus: Integer) — após a navegação.
- **OnSourceChanged**(alsNewDocument: Boolean) — a URL mudou.
- **OnDocumentTitleChanged**(aTitle: string) — o título da página foi alterado.
- **OnWebMessageReceived**(aSource, aWebMessageAsJson, aWebMessageAsString: string) — mensagem do conteúdo web.
- **OnNewWindowRequested**(aURI: string; alsUserInitiated: Boolean; var aHandled: Boolean) — popup/nova aba solicitada.
- **OnScriptExecuted**(aErrorCode: HRESULT; aResultAsJson: string) — resultado JS assíncrono.
- **OnContentLoading**(alsErrorPage: Boolean) — o conteúdo começou a carregar.
- **OnHistoryChanged** — o histórico de navegação mudou.
- **OnProcessFailed**(aKind: Integer) — o processo do navegador travou.
- **OnPermissionRequested**(aURI: string; aKind: Integer; alsUserInitiated: Boolean; var aState: Integer) — requisição de permissão.
- **OnWindowCloseRequested** — `window.close()` chamado.
- **OnDownloadStarting**(aURI, aResultFilePath: string; var aCancel, aHandled: Boolean; var aFilePath: string) — download iniciado.
- **OnDownloadProgress**(aBytesReceived, aTotalBytes: Int64) — progresso do download.
- **OnDownloadCompleted**(aFilePath: string; aState: Integer) — download concluído.
- **OnContextMenuRequested**(aMenuItems: string; aContextKind: Integer; aLocation: TPoint; var aHandled: Boolean) — menu do botão direito.
- **OnFaviconChanged**(aFaviconURI: string) — ícone da página alterado.
- **OnClientCertificateRequested**(aHost: string; aPort: Integer; var aHandled: Boolean) — certificado de cliente necessário.
- **OnServerCertificateError**(aRequestURI: string; aErrorStatus: Integer; var aAction: Integer) — erro de certificado.

- **OnStatusBarTextChanged**(aText: string) — o texto da barra de status foi alterado.
- **OnDOMContentLoaded** — DOM pronto.
- **OnBasicAuthRequested**(aURI: string; var aUserName, aPassword: string; var aHandled: Boolean) — autenticação http basic.
- **OnCapturePreviewCompleted**(aErrorCode: HRESULT) — screenshot concluído.

TsgcWebView2 | Navigation

O TsgcWebView2 fornece vários métodos para navegar até URLs, carregar conteúdo HTML e controlar o histórico do navegador.

Navegação Básica

Use o método **Navigate** para carregar uma URL. O componente dispara **OnNavigationStarting** antes da requisição e **OnNavigationCompleted** quando ela termina.

```
// Navigate to a URL
sgcWebView21.Navigate('https://www.example.com');
```

Você também pode definir a propriedade **DefaultURL** em tempo de design ou antes da inicialização. O componente navega para esta URL automaticamente após o WebView2 estar pronto.

```
// Set the default URL before initialization
sgcWebView21.DefaultURL := 'https://www.example.com';
```

HTML Content

Utilize **NavigateToString** para carregar conteúdo HTML diretamente, sem um servidor ou arquivo.

```
// Load HTML content directly
sgcWebView21.NavigateToString(
  '<html><body><h1>Hello from Delphi</h1>' +
  '<p>This content was loaded with NavigateToString.</p>' +
  '</body></html>');
```

History Navigation

Utilize **GoBack** e **GoForward** para navegar pelo histórico do navegador. Verifique **CanGoBack** e **CanGoForward** antes de chamar esses métodos.

```
// Navigate back
if sgcWebView21.CanGoBack then
  sgcWebView21.GoBack;

// Navigate forward
if sgcWebView21.CanGoForward then
  sgcWebView21.GoForward;
```

Utilize **Reload** para atualizar a página atual e **Stop** para cancelar uma navegação pendente.

```
// Reload the current page
sgcWebView21.Reload;

// Stop loading
sgcWebView21.Stop;
```

Navegação POST

Utilize **NavigateWithPostData** para enviar uma requisição HTTP com método, corpo e cabeçalhos personalizados. Isso é útil para enviar dados de formulário ou chamar REST APIs diretamente no navegador.

```
// 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');
```

Eventos de Navegação

Use **OnNavigationStarting** para inspecionar ou cancelar uma navegação antes de ela começar. Defina **aCancel** como True para bloquear a requisição.

```
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;
```

Utilize **OnNavigationCompleted** para verificar se a navegação foi bem-sucedida ou falhou.

```
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;
```

Tratamento de Nova Janela

Quando a página abre um popup ou um link com `target="_blank"`, o evento **OnNewWindowRequested** dispara. Defina **aHandled** como True para impedir o popup padrão e navegar na mesma janela.

```
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

O TsgcWebView2 fornece métodos para executar JavaScript no contexto do navegador, recuperar resultados e estabelecer comunicação bidirecional entre sua aplicação Delphi e o conteúdo web.

Execução Assíncrona

Utilize **ExecuteScript** para executar JavaScript de forma assíncrona. O resultado é retornado no evento **OnScriptExecuted** como uma string 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;
```

Você pode executar qualquer expressão JavaScript válida. O valor de retorno é sempre serializado como 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"');
```

Sync Execution

Utilize **ExecuteScriptSync** para execução bloqueante que retorna o resultado diretamente. Isso é mais simples quando você precisa do valor imediatamente, mas bloqueia a thread chamadora até que o script seja concluído.

```
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;
```

Init Scripts

Use **AddInitScript** para registrar JavaScript que é executado automaticamente em cada carregamento de página, antes de quaisquer outros scripts na página. Isso é útil para injetar polyfills, sobrescrever APIs do navegador ou configurar handlers de mensagens.

```
// 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);' +
  '};');
```

Utilize **RemoveInitScript** para cancelar o registro de um init script adicionado anteriormente pelo seu ID.

Web Messaging

O web messaging fornece comunicação bidirecional entre sua aplicação Delphi e o JavaScript em execução no navegador. Use **PostWebMessageAsString** ou **PostWebMessageAsJson** para enviar dados do Delphi ao JavaScript, e trate **OnWebMessageReceived** para receber dados do JavaScript.

Enviando mensagens do Delphi para o JavaScript:

```
// Send a plain string message
sgcWebView21.PostWebMessageAsString('Hello from Delphi!');

// Send a JSON message
sgcWebView21.PostWebMessageAsJson('{"action":"refresh","id":42}');
```

Recebendo mensagens em 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);' +
  '});');
```

Recebendo mensagens do JavaScript no 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;
```

A partir do JavaScript, envie mensagens ao Delphi usando:

```
// JavaScript code inside the web page:
// window.chrome.webview.postMessage('Hello from JavaScript!');
// window.chrome.webview.postMessage({action: 'save', data: [1,2,3]});
```

TsgcWebView2 | Cookie Management

O TsgcWebView2 expõe uma propriedade **CookieManager** para ler, criar, atualizar e excluir cookies no contexto do navegador.

Obtendo Cookies

Utilize **CookieManager.GetCookies** para recuperar os cookies de uma determinada URI. O método retorna uma lista de objetos de cookie que você pode iterar.

```
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;
```

Adding Cookies

Utilize **CookieManager.AddOrUpdateCookie** para criar um novo cookie ou atualizar um existente. Especifique no mínimo o nome, o valor, o domínio e o caminho.

```
// 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
```

Deleting Cookies

Use **DeleteCookie** para remover um cookie específico por nome e URI. Use **DeleteAllCookies** para limpar todos os cookies do navegador.

```
// 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 | Download Control

TsgcWebView2 fornece eventos para interceptar, monitorar e controlar downloads de arquivos iniciados pelo navegador.

Download Events

O evento **OnDownloadStarting** dispara quando um download começa. Você pode cancelar o download, alterar o caminho do arquivo de destino, ou deixá-lo prosseguir com o comportamento padrão.

```
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.');
```

Download Progress

O evento **OnDownloadProgress** dispara periodicamente durante o download, reportando os bytes recebidos e o total de bytes esperados.

```
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 Concluído

O evento **OnDownloadCompleted** dispara quando o download termina. Verifique **aState** para determinar se o download foi bem-sucedido, foi cancelado ou falhou.

```
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.');
```

Caminho de download personalizado

Defina o parâmetro **aFilePath** em `OnDownloadStarting` para redirecionar o download para um local personalizado. Defina **aHandled** como `True` para suprimir a caixa de diálogo de salvamento padrão.

```
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 | Settings

O TsgcWebView2 expõe uma propriedade **Settings** do tipo TsgcWebView2Settings que controla o comportamento do navegador. Essas configurações podem ser definidas em tempo de design ou alteradas em tempo de execução após a inicialização.

TsgcWebView2Settings Properties

- **ScriptEnabled**: Boolean — habilita ou desabilita a execução de JavaScript. Padrão: True.
- **WebMessageEnabled**: Boolean — habilita ou desabilita o web messaging (postMessage). Padrão: True.
- **DefaultScriptDialogsEnabled**: Boolean — habilita ou desabilita os diálogos JavaScript padrão (alert, confirm, prompt). Padrão: True.
- **StatusBarEnabled**: Boolean — mostra ou oculta a barra de status do navegador. Padrão: True.
- **DevToolsEnabled**: Boolean — permite ou bloqueia o acesso ao Edge DevTools. Padrão: True.
- **ContextMenuEnabled**: Boolean — habilita ou desabilita o menu de contexto do botão direito do navegador. Padrão: True.
- **ZoomControlEnabled**: Boolean — permite ou bloqueia o zoom iniciado pelo usuário (Ctrl+scroll, Ctrl+mais/menos). Padrão: True.
- **BuiltInErrorPageEnabled**: Boolean — exibe ou oculta a página de erro integrada para falhas de navegação. Padrão: True.

Configuração em Tempo de Design

Defina as propriedades de Settings no Object Inspector. Elas são aplicadas automaticamente depois que o ambiente WebView2 é inicializado.

```
// 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;
```

Runtime Configuration

Você pode alterar configurações em tempo de execução após o componente ser inicializado. As alterações entram em vigor imediatamente.

```
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

A propriedade **UserDataFolder** especifica o diretório onde o navegador armazena cookies, cache, permissões e outros dados de perfil. Cada pasta única cria um perfil de navegador isolado.

```
// Use a custom data folder for isolated profiles
sgcWebView21.UserDataFolder := 'C:\AppData\MyApp\Profile1';
```

Defina esta propriedade antes da inicialização (antes de o componente criar o ambiente WebView2). Se não for definida, uma pasta padrão é usada no diretório temporário da aplicação.

BrowserExecutableFolder

A propriedade **BrowserExecutableFolder** aponta para uma distribuição de WebView2 Runtime de versão fixa. Utilize isto para distribuir uma versão específica do navegador com sua aplicação, em vez de depender do runtime instalado no sistema.

```
// Use a fixed-version runtime bundled with the application
sgcWebView21.BrowserExecutableFolder :=
  ExtractFilePath(ParamStr(0)) + 'WebView2Runtime';
```

Defina esta propriedade antes da inicialização. Se deixada vazia, o componente utiliza o Evergreen WebView2 Runtime instalado no sistema.

TsgcWebView2 | Recursos avançados

O TsgcWebView2 fornece acesso a recursos avançados do navegador, incluindo impressão, capturas de tela, controle de áudio, tratamento de certificados e mais.

Suporte a Impressão

Use **PrintToPdf** para salvar a página atual como um arquivo PDF, ou **ShowPrintUI** para exibir a caixa de diálogo nativa de impressão.

```
// Save the current page as PDF
sgcWebView21.PrintToPdf('C:\Output\page.pdf');
```

```
// Show the native print dialog
sgcWebView21.ShowPrintUI;
```

Captura de tela

Use **CapturePreviewToFile** para tirar uma captura de tela da página atual. O parâmetro format especifica o tipo de imagem: 0 para PNG, 1 para JPEG. O evento **OnCapturePreviewCompleted** dispara quando a captura termina.

```
// 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;
```

Controle de Áudio / Mudo

Utilize a propriedade **IsMuted** para silenciar ou reativar a reprodução de áudio. Verifique **IsDocumentPlayingAudio** para detectar se a página está reproduzindo áudio no momento.

```
// Toggle mute
sgcWebView21.IsMuted := not sgcWebView21.IsMuted;

// Check if audio is playing
if sgcWebView21.IsDocumentPlayingAudio then
  LabelStatus.Caption := 'Audio is playing';
```

Tratamento de Certificados

Use **OnClientCertificateRequested** para responder quando um servidor exigir um certificado de cliente. Use **OnServerCertificateError** para tratar erros de certificado TLS (por exemplo, certificados autoassinados em desenvolvimento).

```
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;
```

Mapeamento de host virtual

Utilize **SetVirtualHostNameToFolderMapping** para mapear um hostname para uma pasta local. Isso permite que o conteúdo web faça referência a arquivos locais utilizando uma URL virtual em vez de caminhos 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');
```

Gerenciamento de perfil

Utilize **GetProfileName** para recuperar o nome do perfil de navegador atual. Utilize **ClearBrowsingData** ou **ClearAllBrowsingData** para remover dados em cache, cookies e outros artefatos de navegação.

```
// Get the profile name
ShowMessage('Profile: ' + sgcWebView21.GetProfileName);

// Clear all browsing data
sgcWebView21.ClearAllBrowsingData;

// Clear specific browsing data types
sgcWebView21.ClearBrowsingData($0001); // cache only
```

Basic Authentication

O evento **OnBasicAuthRequested** dispara quando um servidor solicita autenticação HTTP Basic. Forneça as credenciais e defina **aHandled** como True.

```
procedure TFormMain.sgcWebView21BasicAuthRequested(Sender: TObject;
  const aURI: string; var aUserName, aPassword: string;
  var aHandled: Boolean);
begin
  aUserName := 'admin';
  aPassword := 'secret';
  aHandled := True;
end;
```

Context Menu

O evento **OnContextMenuRequested** dispara quando o usuário clica com o botão direito no navegador. Defina **aHandled** como True para suprimir o menu padrão e exibir o seu próprio.

```
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

A propriedade **FaviconURI** retorna a URI do favicon da página atual. O evento **OnFaviconChanged** dispara quando o favicon muda.

```
procedure TFormMain.sgcWebView21FaviconChanged(Sender: TObject;
  const aFaviconURI: string);
begin
  LabelFavicon.Caption := 'Favicon: ' + aFaviconURI;
end;
```

Status Bar

A propriedade **StatusBarText** retorna o texto atual da barra de status (tipicamente a URL de um link sob o cursor). O evento **OnStatusBarTextChanged** dispara quando o texto muda.

```
procedure TFormMain.sgcWebView21StatusBarTextChanged(Sender: TObject;
  const aText: string);
begin
  StatusBar1.SimpleText := aText;
end;
```

Task Manager

Utilize **OpenTaskManagerWindow** para abrir o gerenciador de tarefas do navegador Edge, que mostra o uso de memória e CPU de cada processo do navegador.

```
sgcWebView21.OpenTaskManagerWindow;
```

Buffer compartilhado

Utilize **PostSharedBufferToScript** para compartilhar um buffer de memória entre sua aplicação Delphi e o conteúdo web para transferência de dados de alto desempenho.

```
// 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}');
```

Direct COM Access

Para cenários avançados não cobertos pela API do componente, utilize as propriedades **WebView**, **Controller** e **Environment** para acessar diretamente as interfaces COM subjacentes do WebView2.

```
var
  vWebView: ICoreWebView2;
begin
  vWebView := sgcWebView21.WebView;
  if Assigned(vWebView) then
  begin
    // Call any ICoreWebView2 method directly
```

```
end;  
end;
```

TsgcUDPClient

O TsgcUDPClient implementa o Cliente UDP baseado na biblioteca Indy.

Introdução

O TsgcUDPClient implementa o Cliente UDP baseado na biblioteca Indy.

UDP é um protocolo sem conexão onde não há garantia de que a mensagem enviada chegue ao destino, mas, ao contrário do protocolo TCP, é muito mais rápido.

1. Coloque um componente **TsgcUDPClient** no formulário
2. Defina **Host** e **Port** (o padrão é 80) para conectar a um Servidor UDP disponível.

```
oClient = new TsgcUDPClient();  
oClient->Host = "127.0.0.1";  
oClient->Port = 80;
```

3. Você pode conectar através de um HTTP Proxy Server; você precisa definir as propriedades do proxy:

Host: nome de host do servidor proxy.

Port: número da porta do servidor proxy.

Username: nome de usuário para autenticação, deixe em branco para anônimo.

Password: senha para autenticação, deixe em branco para anônimo.

4. Se quiser, você pode tratar os eventos
5. Chame o método **WriteData** para enviar qualquer mensagem ao servidor UDP.

Referência

- Propriedades — 9 propriedades disponíveis no componente.
- Methods — 2 métodos públicos.
- Eventos — 3 eventos que você pode tratar.

TsgcUDPServer

TsgcUDPServer implementa o Servidor UDP baseado na biblioteca Indy.

Introdução

TsgcUDPServer implementa o Servidor UDP baseado na biblioteca Indy.

UDP é um protocolo sem conexão onde não há garantia de que a mensagem enviada chegue ao destino, mas, ao contrário do protocolo TCP, é muito mais rápido.

1. Solte um componente **TsgcUDPServer** no formulário
2. Defina a **Port** de escuta.

```
oClient = new TsgcUDPServer();  
oClient->Port = 80;
```

3. Para **iniciar** o servidor, defina a propriedade **Active = true**.

Referência

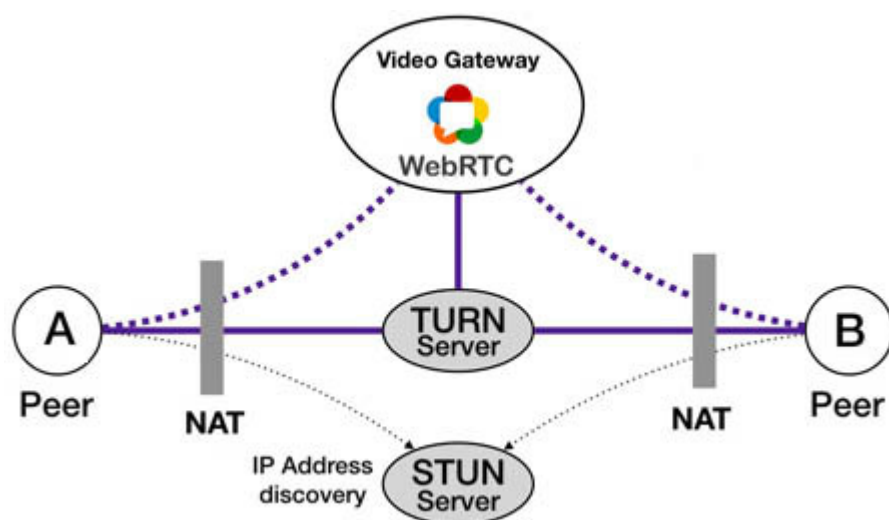
- Propriedades — 11 propriedades disponíveis no componente.
- Métodos — 7 métodos públicos.
- Eventos — 6 eventos que você pode tratar.

STUN

O STUN (Session Traversal Utilities for NAT) é um protocolo da IETF usado para áudio e vídeo em tempo real em redes IP. O STUN é um protocolo servidor-cliente; um servidor STUN geralmente opera tanto em UDP quanto em TCP e escuta na porta 3478.

O principal propósito do protocolo STUN é permitir que um dispositivo em execução atrás de um NAT descubra seu IP público e que tipo de NAT está por trás dele.

O STUN fornece um mecanismo para comunicação entre peers que estão atrás de um NAT. Os peers enviam uma requisição a um servidor STUN para saber qual é o endereço IP público e a Porta. As requisições de binding enviadas do cliente ao servidor são usadas para determinar os bindings de IP e portas alocados pelos NATs. O cliente STUN envia uma requisição de Binding ao servidor STUN, o servidor examina o IP de origem e a Porta usados pelo cliente, e retorna essa informação ao cliente.



O servidor STUN basicamente envia dois tipos de resposta: bem-sucedida ou de erro. Cada resposta tem uma lista de atributos que contêm informações sobre o endereço IP de binding, o código de erro, o motivo do erro, etc.

Componentes

- **TsgcSTUNClient**: o componente de cliente que implementa o protocolo STUN e permite que você envie binding requests a servidores STUN.
- **TsgcSTUNServer**: o componente servidor que implementa o protocolo STUN.

TsgcSTUNClient

Componente *TsgcSTUNClient*.

Introdução

TsgcSTUNClient é o cliente que implementa o protocolo STUN e permite que você envie requisições de binding para servidores STUN.

O componente permite que você use **UDP** e **TCP** como transporte. Ao usar UDP como transporte, ele implementa um **mecanismo de Retransmissão** para reenviar requisições se a resposta não tiver chegado após um curto período.

Uso básico

Normalmente os servidores STUN rodam na porta UDP 3478 e não requerem autenticação, portanto, para enviar uma requisição de binding STUN, preencha as propriedades do servidor para que o cliente saiba onde conectar e trate os eventos em que o componente receberá a resposta do servidor.

Configure o servidor

- Host: o IP ou nome DNS do servidor, exemplo: `stun.sgcwebsockets.com`
- Port: a porta de escuta do Servidor, exemplo: `3478`

Chame o método **SendRequest**, para enviar uma requisição de binding ao servidor STUN.

Trate os eventos

- Se o servidor retornar uma resposta bem-sucedida, o evento **OnSTUNResponseSuccess** será chamado e você pode acessar as informações de binding lendo o objeto **aBinding**.
- Se o servidor retornar um erro, o evento **OnSTUNResponseError** será chamado e você pode acessar o Error Code e Reason lendo o objeto **aError**.

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

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

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

Referência

- Propriedades — 9 propriedades disponíveis no componente.
- Métodos — 3 métodos públicos.
- Eventos — 4 eventos que você pode tratar.

Guides

- **Bindings**
 - [UDP Retransmissions](#)
 - [Long Term Credentials](#)
- - [Attributes](#)

STUN Client | UDP Retransmissions

Ao executar **STUN** sobre **UDP**, é possível que a **mensagem STUN** seja **descartada** pela rede. A confiabilidade das transações de requisição/resposta STUN é alcançada através de retransmissões da mensagem de requisição pela própria aplicação cliente.

Um cliente deve retransmitir uma mensagem de requisição STUN começando com um intervalo de RTO ("Retransmission TimeOut"), dobrando após cada retransmissão. O RTO é uma estimativa do tempo de ida e volta.

Por padrão, o Cliente STUN do sgcWebSockets já vem configurado com um RTO de 500 ms e um valor de Max Retries de 7.

Por exemplo, supondo um RTO de 500 ms, as requisições seriam enviadas nos tempos 0 ms, 500 ms, 1500 ms, 3500 ms, 7500 ms, 15500 ms e 31500 ms. Se o cliente não tiver recebido uma resposta após 39500 ms, o cliente considerará que a transação atingiu o tempo limite.

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

STUN Client | Long Term Credentials

O mecanismo de credencial de longo prazo depende de uma credencial de longo prazo, na forma de um nome de usuário e senha que são compartilhados entre cliente e servidor. A credencial é considerada de longo prazo, uma vez que se assume que ela é provisionada para um usuário e permanece em vigor até que o usuário deixe de ser assinante do sistema ou até que seja alterada.

Você pode configurar as credenciais Long-term no cliente STUN do sgcWebSockets utilizando o seguinte código.

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

Se o servidor requer credenciais de longo prazo e as credenciais enviadas pelo cliente estiverem incorretas, o cliente receberá um erro 401 Unauthorized como resposta no evento **OnSTUNResponseError**.

STUN Client | Attributes

Toda vez que um servidor envia uma mensagem ao cliente, como uma mensagem de resposta a uma requisição de binding, a mensagem STUN contém uma lista de atributos com informações sobre a resposta.

Você pode acessar esses atributos usando a classe `TsgcSTUN_Message` e sua propriedade `Attributes`, que contém uma lista de objetos `TsgcSTUN_Attribute` com informações úteis.

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

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

TsgcSTUNServer

O servidor STUN pode ser configurado com ou sem Autenticação, pode verificar o Atributo Fingerprint, enviar um servidor alternativo e muito mais.

Introdução

TsgcSTUNServer é o servidor que implementa o protocolo STUN e permite que você processe requisições de binding de clientes STUN.

O servidor STUN pode ser configurado com ou sem Autenticação, pode verificar o Atributo Fingerprint, enviar um servidor alternativo e muito mais.

Uso básico

Normalmente, os servidores STUN rodam na porta UDP 3478 e não exigem autenticação, portanto, para configurar um servidor STUN, defina a porta de escuta (por padrão 3478) e inicie o servidor.

Configure o servidor

- Port: a porta de escuta do Servidor, exemplo: 3478

Defina a propriedade **Active = True** para iniciar o servidor STUN.

```
TsgcSTUNServer oSTUN = new TsgcSTUNServer();  
oSTUN->Port = 3478;  
oSTUN->Active = true;
```

Referência

- Properties — 8 propriedades disponíveis no componente.
- Methods — 2 métodos públicos.
- Eventos — 4 eventos que você pode tratar.

Guides

- **Configurations**
 - [Long-Term Credentials](#)
 - [Alternate Server](#)

STUN Server | Long-Term Credentials

Normalmente os Servidores STUN são configurados sem Autenticação, de modo que qualquer cliente STUN pode enviar uma requisição de binding e esperar uma resposta do servidor sem Autenticação.

O servidor STUN `sgcWebSockets` suporta Long-Term Credentials, então você pode configurar o `TsgcSTUNServer` para permitir somente requisições de binding com informações de credenciais Long-Term.

Para configurá-lo, acesse a propriedade `STUNOptions.Authorization` e habilite-a.

Em seguida, acesse a propriedade `LongTermCredentials` e habilite-a. Por padrão, este tipo de autorização já vem configurado com uma string `Realm` e com um valor `StaleNonce` padrão de 10 minutos (= 600 segundos).

```
TsgcSTUNServer oSTUN = new TsgcSTUNServer();
oSTUN->STUNOptions->Authentication->Enabled = true;
oSTUN->STUNOptions->Authentication->LongTermCredentials->Enabled = true;
oSTUN->STUNOptions->Authentication->LongTermCredentials->Realm = "sgcWebSockets";
oSTUN->STUNOptions->Authentication->LongTermCredentials->StaleNonce = 600;
oSTUN->Port = 3478;
oSTUN->Active = true;

private void OnSTUNRequestAuthorization(TObject *Sender, const TsgcSTUN_Message *aRequest,
    const string aUsername, const string aRealm, ref string Password)
{
    if (aUsername == "my-user")
    {
        Password = "my-password";
    }
}
```

STUN Server | Alternate Server

O servidor alternativo representa um endereço de transporte alternativo que identifica um servidor STUN diferente que o cliente STUN deve tentar.

O Servidor STUN pode ser configurado para enviar um servidor alternativo como resposta a uma binding request; para configurar esse comportamento, basta acessar a propriedade `STUNOptions.BindingAttributes.AlternateServer` e configurar aqui os valores necessários.

```
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 o cliente recebe o atributo de resposta Alternate Server, ele tentará enviar uma requisição de binding ao novo servidor.

TURN

O protocolo Traversal Using Relays around NAT (TURN) permite que um servidor retransmita pacotes de dados entre dispositivos.

Se o endereço IP público tanto do chamador quanto do chamado não for descoberto, o TURN fornece uma técnica de fallback para retransmitir a chamada entre os endpoints.

Conectar uma sessão WebRTC é um esforço orquestrado feito com a assistência de múltiplos servidores WebRTC. Os servidores de travessia de NAT no WebRTC são responsáveis por garantir que a mídia seja conectada corretamente. Esses servidores são STUN e TURN.

Como as sessões WebRTC se conectam

Diretamente

Se ambos os dispositivos estão na rede local, nenhum esforço especial é necessário para conectá-los um ao outro. Se um dispositivo tiver o endereço IP local do outro dispositivo, então eles podem se comunicar diretamente.

Diretamente com Endereço IP público

Conectar o WebRTC diretamente usando o endereço IP público obtido via protocolo [STUN](#).

Rotear por meio de um Servidor TURN

Quando os peers estão atrás de um NAT e há Firewalls, a conexão direta não é possível, portanto um servidor TURN é necessário para rotear os dados entre os peers.

Componentes

- [TsgcTURNClient](#): o componente cliente que implementa o protocolo TURN e permite que você faça alocação, crie permissões, envie indications etc. a um servidor TURN.
- [TsgcTURNServer](#): o componente servidor que implementa o protocolo TURN.

TsgcTURNClient

Componente *TsgcTURNClient*.

Introdução

TsgcTURNClient é o cliente que implementa o protocolo TURN e permite que você envie requisições de alocação a servidores TURN. O cliente herda do STUN Client, então todos os métodos suportados pelo [STUN client](#) já são suportados pelo TURN Client.

Uso básico

Normalmente, os servidores TURN executam na porta UDP 3478 e não requerem autenticação. Portanto, para enviar uma requisição TURN, preencha as propriedades do servidor para permitir que o cliente saiba onde se conectar e trate os eventos onde o componente receberá a resposta do servidor.

Configure o servidor

- Host: o IP ou nome DNS do servidor, exemplo: turn.sgcwebsockets.com
- Port: a porta de escuta do Servidor, exemplo: 3478

Chame o método **Allocate**, para enviar uma requisição de alocação de um Endereço IP e uma Porta ao servidor TURN.

Trate os eventos

- Se o servidor retornar uma resposta bem-sucedida, o evento **OnTURNAllocateSuccess** será chamado e você poderá acessar as informações de Allocation lendo o objeto **aAllocation**.
- Se o servidor retornar um erro, o evento **OnSTUNResponseError** será chamado e você pode acessar o Error Code e Reason lendo o objeto **aError**.

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

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

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

Referência

- Propriedades — 10 propriedades disponíveis no componente.
- Métodos — 9 métodos públicos.
- Eventos — 11 eventos que você pode tratar.

Guides

- **Allocation**
 - [Allocate IP Address](#)
 - [Create Permissions](#)
- **Indications**
 - [Send Indication](#)
- **Channels**
 - [TURN Client Channels](#)

TURN Client | Allocate IP Address

O Protocolo TURN permite que você utilize um Relayed IP Address para trocar dados entre peers que estão atrás de NATs.

Para criar um novo Relayed IP Address em um servidor TURN, o cliente deve primeiro chamar o método **Allocate**; este método envia uma requisição ao servidor TURN para criar um novo Relayed IP Address. Se o servidor TURN puder criar um novo Relayed IP Address, o cliente receberá uma resposta bem-sucedida. O cliente poderá se comunicar com outros peers durante o tempo definido no lifetime da Allocation.

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

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

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

O tempo de vida pode ser atualizado para evitar a expiração usando o método **Refresh**. O Lifetime é o número de segundos até expirar. Se o valor for zero, a Allocation será excluída.

```
oTURN->Refresh(600);
```

TURN Client | Create Permissions

Quando uma nova Allocation é criada em um servidor TURN, esta allocation não pode processar nenhum pacote recebido de outros peers se não tiver permissões. Portanto, para permitir que outros peers se comuniquem usando um Relayed IP Address, primeiro o TURN Client deve criar permissões para os Endereços IP que têm permissão para trocar Dados.

Para criar uma nova Permission, basta chamar o método **CreatePermission** e passar como parâmetro o Endereço IP do peer. Se o IP do Peer já existir no servidor TURN, ele será atualizado; caso contrário, será criado. As Permissions expiram após 5 minutos, a menos que sejam atualizadas.

O cliente TURN só permite que você chame o método CreatePermission se existir uma alocação ativa.

Se a permissão for criada com sucesso, o evento **OnTURNCreatePermission** é chamado.

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

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

TURN Client | Send Indication

O Protocolo TURN suporta 2 mecanismos para enviar e receber dados de peers; um deles são os mecanismos Send e Data.

O cliente TURN pode usar o método **SendIndication** para enviar dados ao servidor para retransmissão a um peer. O cliente TURN deve garantir que haja uma permissão para o Endereço IP do Peer para o qual a Send Indication será enviada.

As respostas a um método SendIndication são recebidas no evento **OnTURNDataIndication**.

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

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

TURN Client | Canais

Os canais fornecem uma maneira para o TURN Client e Server enviarem dados de aplicação utilizando mensagens ChannelData, que têm menos sobrecarga do que as Indications [Send e Data](#).

Antes de usar mensagens ChannelData para trocar dados entre peers, o cliente TURN deve criar um novo canal. Para fazer isso, basta chamar o método **ChannelBind** passando o Endereço IP e a Porta do Peer como parâmetros.

Se o servidor TURN puder vincular um novo canal, o cliente TURN receberá uma resposta bem-sucedida no evento **OnTURNChannelBind**.

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

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

Um channel binding dura 10 minutos, a menos que seja atualizado. Para atualizar um canal, basta chamar o método **ChannelBind** novamente.

Quando o cliente TURN recebe uma nova ChannelMessage, o evento **OnTURNChannelData** é chamado.

```
void OnTURNChannelData(TObject *Sender, const TsgcSocketConnection *aSocket,
    const TsgcTURNChannelData *aChannelData)
{
    DoLog("#Channel Data: [" + IntToStr(aChannelData->ChannelID) + "] " +
        sgcGetStringFromBytes(aChannelData->Data));
}
```

TsgcTURNServer

O TURN Server suporta Long-Term Authentication, Allocation, Permissions, Channel Data e mais.

Introdução

TsgcTURNServer é o servidor que implementa o protocolo TURN e permite que você processe requisições de clientes TURN. O componente herda de [TsgcSTUNServer](#), portanto todos os métodos e propriedades estão disponíveis no TsgcTURNServer.

O TURN Server suporta Long-Term Authentication, Allocation, Permissions, Channel Data e mais.

Uso básico

Normalmente, os servidores TURN executam na porta UDP 3478 e requerem credenciais de Longo Prazo. Portanto, para configurar um servidor TURN, defina a porta de escuta (por padrão 3478) e inicie o servidor.

Configure o servidor

- Port: a porta de escuta do Servidor, exemplo: 3478
- Defina as propriedades de Long-Term Credentials em `TURNOptions.Authentication.LongTermCredentials`
- Manipule o `OnSTUNRequestAuthorization` para definir a senha quando um cliente TURN envia uma requisição ao servidor TURN.

Defina a propriedade **Active = True** para iniciar o servidor 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";
    }
}
```

Referência

- Propriedades — 9 propriedades disponíveis no componente.
- Métodos — 2 métodos públicos.
- Eventos — 11 eventos que você pode tratar.

Guides

- **Configurations**
 - [Long-Term Credentials](#)
- **Allocations**
 - [Allocations](#)

TURN Server | Long Term Credentials

Normalmente, os Servidores TURN são configurados COM Autenticação para requisições TURN e sem Autenticação para requisições STUN.

O Servidor TURN do sgcWebSockets suporta Long-Term Credentials, então você pode configurar o TsgcTURN-Server para permitir apenas requisições com informações de credenciais Long-Term.

Para configurá-lo, acesse a propriedade `TURNOptions.Authorization` e habilite-a.

Em seguida, acesse a propriedade `LongTermCredentials` e habilite-a. Por padrão, este tipo de autorização já vem configurado com uma string `Realm` e com um valor `StaleNonce` padrão de 10 minutos (= 600 segundos).

```
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 | Allocations

Todas as operações TURN giram em torno de alocações e todas as mensagens TURN estão associadas a uma Allocation. Uma alocação consiste em:

- O endereço de transporte retransmitido
- A 5-Tupla: Endereço IP do cliente, porta IP do cliente, endereço IP do servidor, porta do servidor e protocolo de transporte.
- As informações de autenticação.
- O tempo de expiração de cada endereço de transporte retransmitido.
- Uma lista de permissões para cada endereço de transporte retransmitido (relayed).
- Uma lista de bindings de canais para cada endereço de transporte retransmitido.

Quando um cliente TURN envia uma requisição Allocate, esta mensagem TURN é processada pelo servidor, que tenta criar um novo Relayed Transport Address. Por padrão, se houver alguma porta UDP disponível, ele criará um novo Relayed Address, mas você pode utilizar o evento **OnTURNBeforeAllocate** para rejeitar uma nova requisição de Allocation.

```
void OnTURNBeforeAllocate(TObject *Sender, const TsgcSocketConnection *aSocket,
    const string aIP, Word aPort, ref bool Reject)
{
    if (your own rules) == false
    {
        Reject = false;
    }
}
```

Se o processo continuar, o servidor cria uma nova allocation e o evento **OnTURNCreateAllocation** é chamado. Este evento fornece informações sobre a Allocation através da classe TsgcTURNAllocationItem.

```
void OnTURNCreateAllocation(TObject *Sender, const TsgcSocketConnection *aSocket,
    const TsgcTURNAllocationItem *Allocation)
{
    DoLog("New Allocation: " + Allocation->RelayIP + ":" + IntToStr(Allocation->RelayPort));
}
```

Quando a Allocation expira ou é excluída ao receber uma Refresh Request do cliente com um lifetime de zero, o evento **OnTURNDeleteAllocation** é disparado.

```
void OnTURNDeleteAllocation(TObject *Sender, const TsgcSocketConnection *aSocket,
    const TsgcTURNAllocationItem *Allocation)
{
    DoLog("Allocation Deleted: " + Allocation->RelayIP + ":" + IntToStr(Allocation->RelayPort));
}
```

ICE

O Interactive Connectivity Establishment (ICE) Protocol é utilizado para travessia de NAT. O ICE usa uma combinação de métodos, incluindo Session Traversal Utility for NAT (STUN) e Traversal Using Relay NAT (TURN). A presença de um Network Address Translator (NAT) apresenta problemas para implementações de Voice over IP (VoIP) e WebRTC.

Componentes

- **TsgcICEClient:** o componente cliente que implementa o protocolo ICE e permite que você obtenha, troque e verifique candidatos.

TsgcICEClient

ICEOptions.

Introdução

TsgcICEClient é o cliente que implementa o protocolo ICE e permite que você envie requisições de alocação a servidores TURN. O cliente requer o [TsgcTURNClient](#) e um [TsgcWebSocketClient](#).

Configuração

O cliente ICE tem as seguintes propriedades

- **ICEOptions.CheckList.MaxCandidates:** é o número máximo de candidatos que o cliente pode tratar, por padrão 100.
- **ICEOptions.GatherCandidates.STUN:** por padrão true, obterá os candidatos utilizando o protocolo STUN (endereço reflexivo).
- **ICEOptions.GatherCandidates.TURN:** por padrão true, obterá os candidatos usando o protocolo TURN (endereço retransmitido).

O cliente ICE requer um **cliente TURN** para coletar os candidatos, então vincule um [TsgcTURNClient](#) à propriedade TURN do [TsgcICEClient](#) antes de chamar o método **GatherCandidates**. Você pode obter mais informações sobre como configurar o [cliente TURN](#).

```
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();
```

Referência

- Propriedades — 3 propriedades disponíveis no componente.
- Métodos — 3 métodos públicos.
- Eventos — 6 eventos que você pode tratar.

Guides

- **Candidatos**
 - [Coletar Candidatos](#)
 - [Parear Candidatos](#)

ICE | Gather Candidates

O ICE começa a coletar candidatos, normalmente obterá endereços IP locais, endereço reflexivo usando o protocolo STUN e endereço relayed usando o protocolo TURN.

Para iniciar a coleta (gathering), chame o método **GatherCandidates**; isso iniciará um timer interno que primeiro obterá os endereços IP locais, depois conectará ao servidor STUN para obter o endereço IP reflexivo e, finalmente, conectará ao servidor TURN para obter o endereço IP retransmitido (relayed).

Sempre que um novo candidato é obtido, o evento **OnICECandidate** será chamado de forma assíncrona; se houver algum erro durante a coleta dos candidatos, o evento **OnICECandidateError** será acionado.

```
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();

void OnICECandidate(TObject *Sender, const TsgcICE_Candidate *aCandidate)
{
    DoLog("#Candidate " + aCandidate->AsString);
}
```

ICE | Pair Candidates

Uma vez que os Candidatos foram obtidos (local e remoto) e as descrições SDP foram definidas, o cliente caller ICE pode começar a processar todos os pares de candidatos para encontrar aqueles que podem trocar dados. Para iniciar esse processo, chame o método **ProcessCandidates**.

O método **ProcessCandidates** avalia todos os pares de candidatos enviando um pacote de binding STUN; se o pacote de binding STUN for recebido como resposta do outro peer, significa que a conexão é possível entre esses 2 peers, então o par é nomeado.

Quando o pareamento é **bem-sucedido**, o evento **OnICECandidatePairNominated** é disparado de forma assíncrona. Se o pareamento tiver um **erro** ou **não conseguir conectar após um timeout**, o evento **OnICECandidatePairFailed** é disparado.

TsgcRTCPeerConnection

O TsgcRTCPeerConnection é um componente cliente que permite que você conecte peers utilizando P2P por meio de UDP.

Introdução

[*Atualmente este componente está em desenvolvimento]

O TsgcRTCPeerConnection é um componente de cliente que permite conectar peers utilizando P2P através de UDP. O fluxo pode ser dividido em 4 etapas:

- Sinalização
- Conectando
- Securing
- Comunicando

Para implementar essas etapas, o cliente utiliza os seguintes protocolos:

- **WebSocket:** este protocolo é usado para sinalização, os clientes trocam o Session Description Protocol e os endereços IP local, público e Relayed.
- **UDP:** este é o protocolo de transporte; o cliente utiliza UDP para enviar/receber mensagens entre peers.
- **DTLS:** similar ao TLS, é uma especificação de criptografia que protege a mensagem entre peers, evitando que terceiros leiam/escrevam mensagens.
- **STUN:** protocolo para obter o endereço IP público.
- **TURN:** protocolo para retransmitir endereços IP quando os peers estão atrás de NATs.
- **ICE:** protocolo para descobrir quais Endereços IP e Portas são acessíveis entre os peers.

Sinalização

Quando o cliente inicia, ele não faz ideia de com quem vai se comunicar nem sobre o que vai se comunicar. A sinalização usa o SDP (Session Description Protocol), que contém detalhes como:

- IPs e Portas onde o peer está acessível.
- Certificado do Fingerprint utilizado para proteger a comunicação.
- User e Password.
- ...

A Sinalização faz uso do protocolo WebSocket para trocar os dados, funciona por meio de um subprotocolo e é implementada no componente TsgcWSPServer_RTCPeerConnection no lado do servidor.

O TsgcRTCPeerConnection já cria internamente um cliente websocket com TsgcWSPClient_RTCPeerConnection anexado.

Para obter os IPs e Ports, o cliente faz uso dos protocolos STUN/TURN para obter essas informações. Portanto, um servidor STUN/TURN também é necessário.

Links:

- [RTCPeerConnection WebSocket Server](#)
- [RTCPeerConnection WebSocket Client](#)
- [RTCPeerConnection STUN TURN](#)
- [Sinalização de RTCPeerConnection](#)

Conectando

Uma vez que os 2 peers conheçam os candidatos e SDPs, o cliente utiliza outro protocolo padrão chamado ICE.

O ICE (Interactive Connection Establishment) permite o estabelecimento de uma conexão entre 2 peers. Os peers podem estar na mesma rede ou atrás de um NAT... O ICE é uma solução para estabelecer uma conexão direta sem um servidor central. Se a conexão não puder ser P2P, o ICE usará TURN para retransmitir os dados usando um servidor TURN.

Uma vez que o ICE encontra um candidato válido que pode conectar entre os dois peers, o próximo passo é criptografar a comunicação.

Links:

- [RTCPeerConnection ICE](#)

Securing

Após os peers terem conectado, a comunicação deve ser segura. Isso é feito usando o DTLS, que é um protocolo criptográfico usado para proteger a comunicação sobre UDP.

Uma vez que o handshake DTLS tenha sido processado com sucesso, outro protocolo é usado, o SRTP (Secure Real-Time Transport Protocol); atualmente o SRTP não está implementado.

Links:

- [RTCPeerConnection DTLS](#)

Comunicando

Uma vez que os 2 peers estejam utilizando um protocolo seguro, a comunicação é feita utilizando 2 protocolos:

- **RTP:** Real Time Transport Protocol: utilizado para trocar mídia criptografada com SRTP.
- **SCTP:** Stream Control Transmission Protocol, utilizado para enviar e receber mensagens de DataChannel criptografadas com dTLS.

Atualmente esses protocolos não estão implementados, mas você pode enviar/receber dados utilizando DTLS sobre UDP.

Links:

- [RTCPeerConnection Data](#)

Referência

- Propriedades — 2 propriedades disponíveis no componente.
- Métodos — 3 métodos públicos.
- Eventos — 14 eventos que você pode tratar.

RTCPeerConnection | WebSocket Server

O cliente `TsgcRTCPeerConnection` requer um Servidor WebSocket para sinalização. O cliente utiliza o protocolo WebSocket para trocar o SDP dos peers e os candidatos (IPs e portas), o que permitirá que os peers se comuniquem.

Para configurar um **servidor WebSocket** você pode usar qualquer um dos servidores WebSocket disponíveis na biblioteca `sgcWebSockets` e anexar um **`TsgcWSPServer_RTCPeerConnection`** que é o subprotocolo usado pelo `RTCPeerConnection`.

```
TsgcWebSocketServer *oServer = new TsgcWebSocketServer();
TsgcWSPServer_RTCPeerConnection *oProtocol = new TsgcWSPServer_RTCPeerConnection();
oProtocol->Server = oServer;
oServer->Port = 8080;
oServer->Active = true;
```

Toda vez que um novo cliente websocket conecta ao servidor websocket, o servidor verifica se há algum outro peer escutando no mesmo canal e encaminhará os dados de acordo.

RTCPeerConnection | Cliente WebSocket

O `RTCPeerConnection` cria internamente um cliente websocket com um subprotocolo personalizado para se comunicar com um servidor websocket. Na propriedade `RTCOptions.WebSocket`, você encontra os valores que definem a conexão websocket

- **Host:** dns ou endereço ip do servidor, exemplo: 127.0.0.1 ou www.esgece.com.
- **Port:** porta de escuta do servidor websocket.
- **TLS:** habilite-o se o servidor estiver usando uma conexão segura.
- **Channel:** o nome do canal usado para trocar dados entre peers (ambos os peers devem ter o mesmo nome de canal e o número máximo de peers é 2).

RTCPeerConnection | STUN TURN

O `TsgcRTCPeerConnection` utiliza o protocolo STUN/TURN para obter o endereço IP público e o endereço IP retransmitido (relayed) (se necessário). Então você precisa de um Servidor STUN/TURN para obter essa informação. Você pode ler mais sobre o servidor STUN/TURN no link a seguir: [TURN Server](#).

Depois que o servidor STUN/TURN estiver em execução, você pode configurar as propriedades do TURN Server na propriedade `RTCOptions.ICE` do `TsgcRTCPeerConnection`.

- **Host:** endereço ip ou dns do servidor TURN. Exemplo: 127.0.0.1 ou [www.esgece.com](#).
- **Port:** normalmente é a porta padrão 3478.
- **Username:** nome de usuário se o servidor TURN estiver usando Long-term credentials (o padrão).
- **Password:** senha se o servidor TURN estiver utilizando credenciais Long-term (o padrão).

RTCPeerConnection | Signaling

Uma vez que o `TsgcRTCPeerConnection` tenha configurado a propriedade `RTCOptions` e os Servidores (WebSocket e STUN/TURN) estejam em execução, o cliente pode iniciar o processo de coleta de candidatos.

O cliente primeiro conecta ao servidor websocket; se a conexão for bem-sucedida, ele envia o SDP local. Em seguida, tenta obter os endereços IP local e público; para obter os endereços IP públicos, enviará uma requisição de binding ao servidor STUN para obter o endereço IP público e o endereço IP retransmitido do servidor TURN. Toda vez que obtém um novo candidato, essa informação é passada ao servidor websocket, que a encaminhará ao outro peer.

Quando o `RTCPeerConnection` tem o SDP Local, o SDP Remoto e os candidatos, ele iniciará um processo de verificação de cada par de candidatos para ver se pode conectar entre eles. Quando um par de candidatos conecta com sucesso, ele é um par de candidatos válido e o processo de Securing continua o fluxo.

RTCPeerConnection | ICE

ICE (Interactive Connectivity Establishment) é o protocolo usado para conectar 2 peers, ele determina todas as rotas possíveis entre os 2 peers e então garante que estejam conectados. Essas rotas também são conhecidas como Candidate Pairs, que são um pareamento de endereço de transporte local e remoto. Esses endereços podem ser o Endereço IP local, o Endereço IP público ou o Relayed Transport Address. Cada peer coleta todos os endereços que deseja usar, os troca e então tenta conectar.

Coletando Endereços

Os seguintes eventos podem ser chamados ao coletar Addresses

OnRTCLocalCandidate

O evento é chamado quando um novo candidato local é encontrado.

OnRTCRemoteCandidate

O evento é chamado quando o servidor websocket envia um candidato remoto para este peer.

OnRTCLocalDescription

O evento é chamado quando o TsgcRTCPeerConnection requer o SDP local

OnRTCRemoteDescription

O evento é chamado quando o servidor websocket envia o SDP remoto a este peer.

Teste de Conectividade

Quando o peer envia requisições de binding para o outro peer para testar se consegue conectar, os seguintes eventos podem ser chamados

OnRTCCandidatePairNominated

Quando ambos os peers conseguem conectar usando este par de candidatos, o evento é chamado.

OnRTCCandidatePairFailed

Quando os peers não conseguem conectar usando este par de candidatos, este evento é chamado.

OnRTCConnect

Este evento é chamado quando há um par de candidatos válido e o DTLS não está habilitado. Se o DTLS estiver habilitado, este evento é chamado após um Handshake DTLS bem-sucedido.

RTCPeerConnection | DTLS

Uma vez que haja um par de candidatos válido (ambos os peers conseguem se conectar e trocar dados entre si), é hora de tornar a conexão segura. DTLS é um protocolo criptográfico que criptografa os dados para evitar a inspeção ou modificação do conteúdo dos dados trocados.

O DTLS requer as bibliotecas openssl (a partir do openssl 1.1+)

A configuração do DTLS pode ser encontrada na propriedade `RTCOptions.DTLSOptions` do `TsgcRTCPeerConnection`. Para habilitar o DTLS, defina a propriedade `RTCOptions.DTLS` como `True`. Veja abaixo as principais propriedades:

RootCertFile: caminho para o arquivo de certificado raiz.

CertFile: caminho para o arquivo de certificado.

KeyFile: caminho para o arquivo de chave do certificado.

VerifyCertificate: se o certificado deve ser verificado, habilite esta propriedade.

VerifyDepth: é uma propriedade Integer que representa o número máximo de links permitidos quando a verificação está

OpenSSL_Options: configuração das bibliotecas openssl.

APIVersion: permite definir qual API OpenSSL será utilizada. Somente a API OpenSSL 1.1+ suporta DTLS.

oslAPI_1_1: usa a API 1.1 do OpenSSL, requer nossa biblioteca Indy personalizada e permite usar as bibliotecas OpenSSL 1.1.1 (com suporte a TLS 1.3).

oslAPI_3_0: utiliza a API 3.0 do OpenSSL, requer nossa biblioteca Indy personalizada e permite utilizar as bibliotecas OpenSSL 3.0.0 (com suporte a TLS 1.3).

LibPath: aqui você pode configurar onde estão localizadas as bibliotecas openssl

oslpNone: este é o padrão, as bibliotecas openssl devem estar na mesma pasta onde está o binário ou em um caminho conhecido.

oslpDefaultFolder: define automaticamente o caminho openssl onde as bibliotecas devem estar localizadas para todas as personalidades do IDE.

oslpCustomFolder: se esta for a opção selecionada, defina o caminho completo na propriedade `LibPathCustom`.

LibPathCustom: quando `LibPath = oslpCustomFolder`, defina aqui o caminho completo onde estão localizadas as bibliotecas openssl.

UnixSymLinks: habilita ou desabilita o carregamento de SymLinks em sistemas Unix (por padrão está habilitado, exceto no OSX64):

oslsSymLinksDefault: por padrão estão habilitados, exceto sob OSX64 (após o MacOS Monterey falha ao tentar carregar a biblioteca sem versão.).

oslsSymLinksLoadFirst: Carrega os SymLinks e faz isso antes de tentar carregar as bibliotecas de versão.

oslsSymLinksLoad: Carrega os SymLinks após tentar carregar as bibliotecas de versão.

oslsSymLinksDontLoad: não carrega os SymLinks.

RTCPeerConnection | Data

Uma vez que a conexão seja bem-sucedida, você pode utilizar o método **WriteData** para enviar alguma mensagem ao outro peer. A conexão fará uso de DTLS sobre UDP e, se possível, a conexão será P2P (ou utilizando um endereço retransmitido quando P2P não for possível).

Send Data

Utilize o método WriteData para enviar qualquer dado ao outro peer. Você pode enviar uma string ou um array de bytes.

```
TsgcRTCPeerConnection *oRTCPeerConnection = new TsgcRTCPeerConnection();  
...  
oRTCPeerConnection->WriteData("Hello from sgcWebSockets!!!");
```

Receber Dados

Toda vez que o TsgcRTCPeerConnection recebe qualquer dado do outro peer, o evento OnRTCMMessage será chamado.

```
void OnRTCMMessage(TObject *Sender, const TBytes aBytes)  
{  
    ShowMessage(TEncoding->UTF8->GetString(aBytes));  
}
```

Datasnap

Por padrão, o DataSnap usa o **TIdHttpWebBrokerBridge** como servidor para tratar requisições HTTP. Mas você pode expandir as possibilidades da sua aplicação DataSnap substituindo esse servidor por outro que suporte mais protocolos e ofereça um desempenho muito melhor, tirando proveito dos protocolos mais recentes como HTTP/2, que melhora o desempenho do servidor, IOCP, que permite tratar muito mais conexões, e muito mais.

Servidores

Servidor	Recursos principais	Descrição
TsgcWSHTTPWebBrokerBridgeServer	WebSocket Protocol HTTP 1.* Protocol XHR Protocol IOCP	Baseado na biblioteca Indy, suporta os protocolos WebSocket e HTTP na mesma porta. IOCP também pode ser habilitado.
TsgcWSHTTP2WebBrokerBridgeServer	Protocolo WebSocket Protocolo HTTP 1.* Protocolo HTTP/2 Protocolo XHR IOCP	Baseado na biblioteca Indy, suporta os protocolos WebSocket e HTTP/2 na mesma porta. O IOCP também pode ser habilitado.
TsgcWSServer_HTTPAPI_WebBrokerBridge	Protocolo WebSocket Protocolo HTTP 1.* Protocolo HTTP/2 Protocolo XHR IOCP	Baseado na Microsoft HTTP API HTTP.SYS, suporta os protocolos WebSocket e HTTP/2 na mesma porta. O IOCP é usado por padrão. Recomendado para melhor desempenho.

TsgcWSHTTPWebBrokerBridgeServer

O **TsgcWSHTTPWebBrokerBridgeServer** faz uso do **TIdHttpWebBrokerBridge** como servidor base e é útil se você quiser usar um único servidor para conexões DataSnap, HTTP e WebSocket.

TsgcWSHTTPWebBrokerBridgeServer herda de **TsgcWebSocketHTTPServer**, então você pode consultar este servidor.

Siga os próximos passos para substituir **TIdHttpWebBrokerBridge** por **TsgcWSHTTPWebBrokerBridgeServer** :

1. Crie uma nova instância de **TsgcWSHTTPWebBrokerBridgeServer**.
2. Substitua todas as chamadas a **TIdHttpWebBrokerBridge** por **TsgcWSHTTPWebBrokerBridgeServer**.
3. Para tratar conexões WebSocket, basta consultar [TsgcWebSocketHTTPServer](#).

Configuração

Os componentes **Dataspnap** estão **localizados apenas na pasta Source**; você não os encontrará nas pastas compiladas, porque esses objetos não estão incluídos no pacote **sgcWebSockets**, portanto você deve criá-los em tempo de execução.

Basta adicionar os arquivos necessários ao seu projeto ou definir o caminho para a pasta **Source** do pacote **sgcWebSockets**. Arquivos necessários:

- **sgcWebSocket_Server_WebBrokerBridge**

Se o projeto fizer uso de **IdHTTPWebBrokerBridge**, altere para **sgcIdHTTPWebBrokerBridge** (isto se aplica apenas à Enterprise Edition).

Eventos

```
FServer = new TsgcWSHTTPWebBrokerBridgeServer();
FServer->OnCommandRequest = OnCommandRequestEvent;
FServer->OnCommandGet = OnCommandGetevent;

void OnCommandRequestEvent(TIdContext *AThread, TIdHTTPRequestInfo *ARequestInfo,
TIdHTTPResponseInfo *AResponseInfo, ref bool aHandled)
{
    if (ARequestInfo->Document == "/test.html")
    {
        aHandled = true;
    }
}

void OnCommandGetevent(TIdContext *AThread, TIdHTTPRequestInfo *ARequestInfo, TIdHTTPResponseInfo
*AResponseInfo)
{
    if (ARequestInfo->Document == "/test.html")
    {
        AResponseInfo->ResponseNo = 200;
        AResponseInfo->ContentText = "hello all";
    }
}
```

Load Balancer

Se o servidor estiver atrás do [TsgcWebSocketLoadBalancerServer](#), você pode ter problemas com CORS; para evitar esses problemas, utilize o código a seguir


```
void __fastcall TWebModule1::WebModuleBeforeDispatch(TObject *Sender, TWebRequest *Request, TWebResponse *Response)
{
    Response->SetCustomHeader("Access-Control-Allow-Origin", "*");
    if (Trim(Request->GetFieldByName("Access-Control-Request-Headers")) != "")
    {
        Response->SetCustomHeader("Access-Control-Allow-Headers", Request->GetFieldByName("Access-Control-Request-Headers"));
        Response->Handled = true;
    }
    if (FServerFunctionInvokerAction != nullptr)
    {
        FServerFunctionInvokerAction->Enabled = AllowServerFunctionInvoker;
    }
}
```

TsgcWSHTTP2WebBrokerBridgeServer

TsgcWSHTTP2WebBrokerBridgeServer usa o **TsgcWebSocketHTTPServer** com **HTTP/2** protocolo habilitado como servidor base e é útil se você quiser usar um único servidor para conexões DataSnap, HTTP/2 e WebSocket.

O **TsgcWSHTTP2WebBrokerBridgeServer** herda de **TsgcWebSocketHTTPServer**, portanto você pode consultar este servidor.

Siga os próximos passos para substituir **TIdHttpWebBrokerBridge** por **TsgcWSHTTP2WebBrokerBridgeServer** :

1. Crie uma nova instância de **TsgcWSHTTP2WebBrokerBridgeServer**.
2. Substitua todas as chamadas a **TIdHttpWebBrokerBridge** por **TsgcWSHTTP2WebBrokerBridgeServer**.
3. Para tratar conexões WebSocket, basta consultar [TsgcWebSocketHTTPServer](#).

Configuração

Os componentes **Datasnap** estão **localizados apenas na pasta Source**; você não os encontrará nas pastas compiladas, porque esses objetos não estão incluídos no pacote **sgcWebSockets**, portanto você deve criá-los em tempo de execução.

Basta adicionar os arquivos necessários ao seu projeto ou definir o caminho para a pasta **Source** do pacote **sgcWebSockets**. Arquivos necessários:

- **sgcWebSocket_Server_Base_WebBrokerBridge**
- **sgcWebSocket_Server_WebBrokerBridge_HTTP2**

Se o projeto fizer uso de **IdHTTPWebBrokerBridge**, altere para **sgcIdHTTPWebBrokerBridge** (isto se aplica apenas à Enterprise Edition).

Eventos

```
FServer = new TsgcWSHTTP2WebBrokerBridgeServer();
FServer->OnCommandRequest = OnCommandRequestEvent;
FServer->OnCommandGet = OnCommandGetevent;

void OnCommandRequestEvent(TIdContext *AThread, TIdHTTPRequestInfo *ARequestInfo,
TIdHTTPResponseInfo *AResponseInfo, ref bool aHandled)
{
    if (ARequestInfo->Document == "/test.html")
    {
        aHandled = true;
    }
}

void OnCommandGetevent(TIdContext *AThread, TIdHTTPRequestInfo *ARequestInfo, TIdHTTPResponseInfo
*AResponseInfo)
{
    if (ARequestInfo->Document == "/test.html")
    {
        AResponseInfo->ResponseNo = 200;
        AResponseInfo->ContentText = "hello all";
    }
}
```

TsgcWSServer_HTTPAPI_WebBrokerBridge

TsgcWSServer_HTTPAPI_WebBrokerBridge usa **TsgcWebSocketServer_HTTPAPI** com **HTTP/2** protocolo habilitado como base do servidor e é útil se você quiser usar um único servidor para conexões DataSnap, HTTP/2 e WebSocket.

TsgcWSServer_HTTPAPI_WebBrokerBridge herda de **TsgcWebSocketServer_HTTPAPI**, portanto você pode consultar este servidor.

Siga os próximos passos para substituir **TIdHttpWebBrokerBridge** por **TsgcWSServer_HTTPAPI_WebBrokerBridge**:

1. Crie uma nova instância de **TsgcWSServer_HTTPAPI_WebBrokerBridge**.
2. Substitua todas as chamadas a **TIdHttpWebBrokerBridge** por **TsgcWSServer_HTTPAPI_WebBrokerBridge**.
3. Para tratar conexões WebSocket basta consultar [TsgcWSServer_HTTPAPI_WebBrokerBridge](#).

Configuração

Os componentes **Datasnap** estão **localizados apenas na pasta Source**; você não os encontrará nas pastas compiladas, porque esses objetos não estão incluídos no pacote **sgcWebSockets**, portanto você deve criá-los em tempo de execução.

Basta adicionar os arquivos necessários ao seu projeto ou definir o caminho para a pasta **Source** do pacote **sgcWebSockets**. Arquivos necessários:

- **sgcWebSocket_Server_Base_WebBrokerBridge**
- **sgcWebSocket_Server_HTTPAPI_WebBrokerBridge**

Se o projeto fizer uso de **IdHTTPWebBrokerBridge**, altere para **sgcIdHTTPWebBrokerBridge** (isto se aplica apenas à Enterprise Edition).

Eventos

```
TsgcWSServer_HTTPAPI_WebBrokerBridge FServer = new TsgcWSServer_HTTPAPI_WebBrokerBridge();
FServer->OnCommandRequest += OnCommandRequestEvent;
FServer->OnMessage += OnWebSocketMessage;

void OnCommandRequestEvent(TsgcWSConnection_HTTPAPI *aConnection,
    const THttpRequest *aRequestInfo, ref THttpResponse *aResponseInfo, ref bool aHandled)
{
    if (ARequestInfo->Document == "/test.html")
    {
        AResponseInfo->ResponseNo = 200;
        AResponseInfo->ContentText = "... body ...";
        aHandled = true;
    }
}

void OnWebSocketMessage(TsgcWSConnection *aConnection, const string aText)
{
    aConnection->WriteData(aText);
}
```

OpenAPI

OpenAPI 3.0

A **OpenAPI Specification**, anteriormente conhecida como **Swagger Specification**, é uma especificação para arquivos de interface legíveis por máquina para descrever, produzir, consumir e visualizar serviços web RESTful. Anteriormente parte do framework Swagger, tornou-se um projeto separado em 2016, supervisionado pela OpenAPI Initiative, um projeto de colaboração de código aberto da Linux Foundation. O Swagger e algumas outras ferramentas podem gerar código, documentação e casos de teste a partir de um arquivo de interface.

Aplicações implementadas com base em arquivos de interface OpenAPI podem gerar automaticamente documentação de métodos, parâmetros e modelos. Isso ajuda a manter a documentação, as bibliotecas cliente e o código-fonte em sincronia.

Pascal Parser

sgcOpenAPI Generator permite a geração de bibliotecas cliente de API (geração de SDK) automaticamente a partir de uma OpenAPI Spec; as seguintes especificações OpenAPI são suportadas:

- **OpenAPI 3.***
- **Swagger 2.*** (convertido automaticamente de 2.0 para 3.0)
- **Swagger 1.*** (convertido automaticamente de 1.0 para 3.0)

O sgcOpenAPI permite que você gere **automaticamente** a interface da API cliente em **linguagem Pascal nativa** a partir de um OpenAPI ou Swagger JSON/YAML. Atualmente suporta do Delphi 7 até a versão mais recente do Delphi.

sgcOpenAPI Generator permite que você crie um arquivo de documentação a partir de uma especificação OpenAPI / Swagger.

Leia mais sobre o [OpenAPI Parser Pascal](#).

OpenAPI Client

A Interface de Cliente gerada contém todas as funções/métodos definidos na especificação OpenAPI. As constantes e enumerações também são criadas.

Os seguintes métodos de **Autenticação** são suportados:

- Basic Authentication
- [OAuth2 Code](#) (interativo)
- [Credenciais OAuth2](#) (não-interativo)
- [JWT](#)

Leia mais sobre o [OpenAPI Client](#).

OpenAPI Server

Você também pode hospedar um serviço OpenAPI 3.0 em Delphi usando o servidor OpenAPI integrado. O servidor roteia automaticamente as requisições com base na sua especificação, serve a especificação em `/openapi.json` e uma Swagger UI em `/docs`, e suporta CORS e validação de requisição/corpo prontos para uso.

Dois maneiras de definir a API:

- **Spec-First** – carrega um arquivo OpenAPI 3.0 JSON/YAML existente em tempo de execução.
- **Code-First** – anote uma classe Delphi com atributos RTTI; a especificação é gerada automaticamente (requer Delphi XE7+).

Leia mais sobre o [OpenAPI Server](#).

APIs

As seguintes APIs foram compiladas e são suportadas:

- [Amazon AWS](#)
- [Google Cloud APIs](#)
- [Microsoft Azure](#)
- [Outras APIs](#)

OpenAPI | Parser Pascal

O **Parser sgcOpenAPI** lê a **Especificação OpenAPI 3.0 em JSON** e cria automaticamente um **Cliente Delphi em Código Pascal Nativo**.

O **Parser sgcOpenAPI** é compatível com as seguintes especificações:

- **OpenAPI 3.***
- **Swagger 2.*** (convertido automaticamente de 2.0 para 3.0)
- **Swagger 1.*** (convertido automaticamente de 1.0 para 3.0)

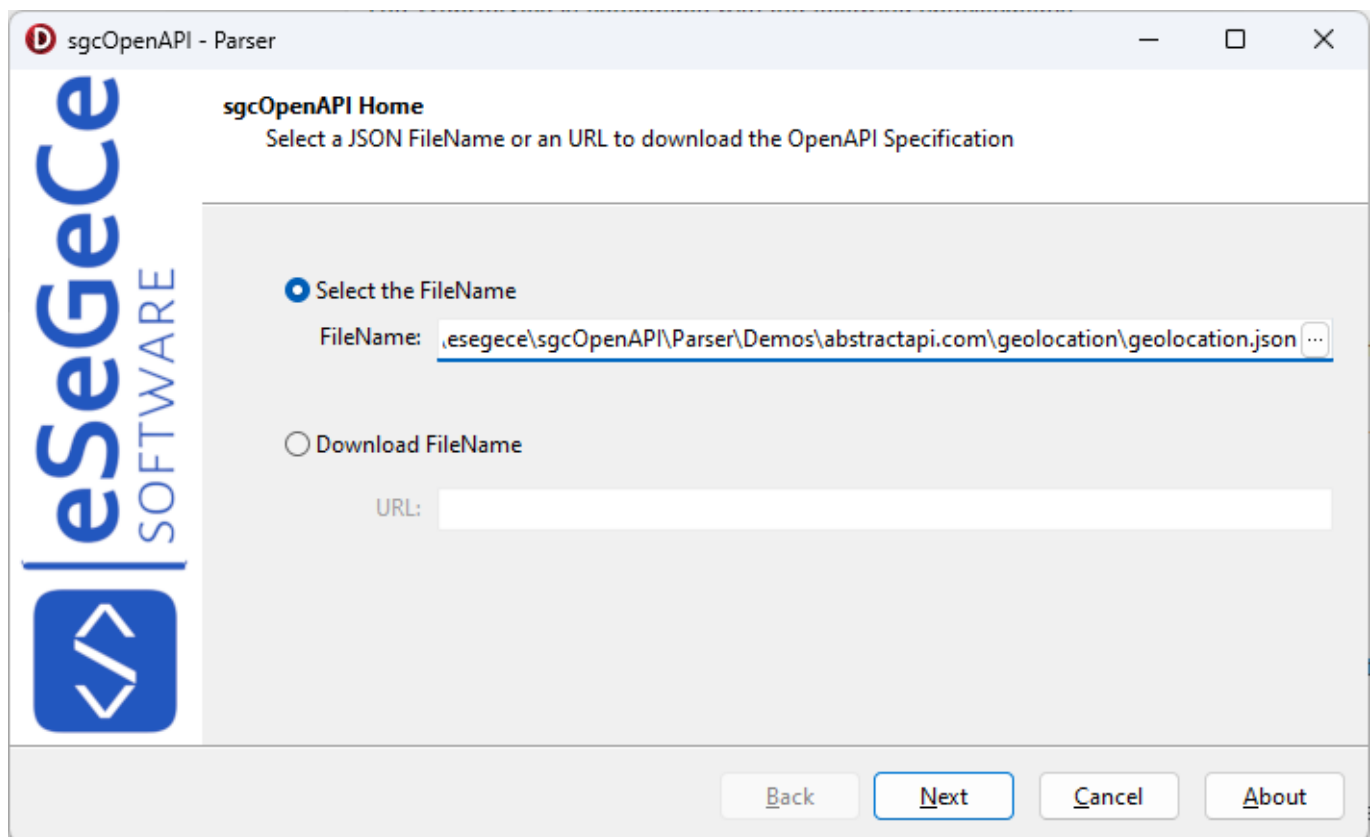
O arquivo de especificação deve estar no formato **JSON**; se a especificação estiver no formato **YAML**, ela será convertida automaticamente. O parser permite importar especificações divididas em **múltiplos schemas**; se você fornecer uma URL, ele tentará baixar as especificações, e se você fornecer um nome de arquivo, ele tentará carregar os schemas externos do seu drive local.

O executável do OpenAPI Parser pode tratar [parâmetros de linha de comando](#) ou ser utilizado como uma [biblioteca](#).

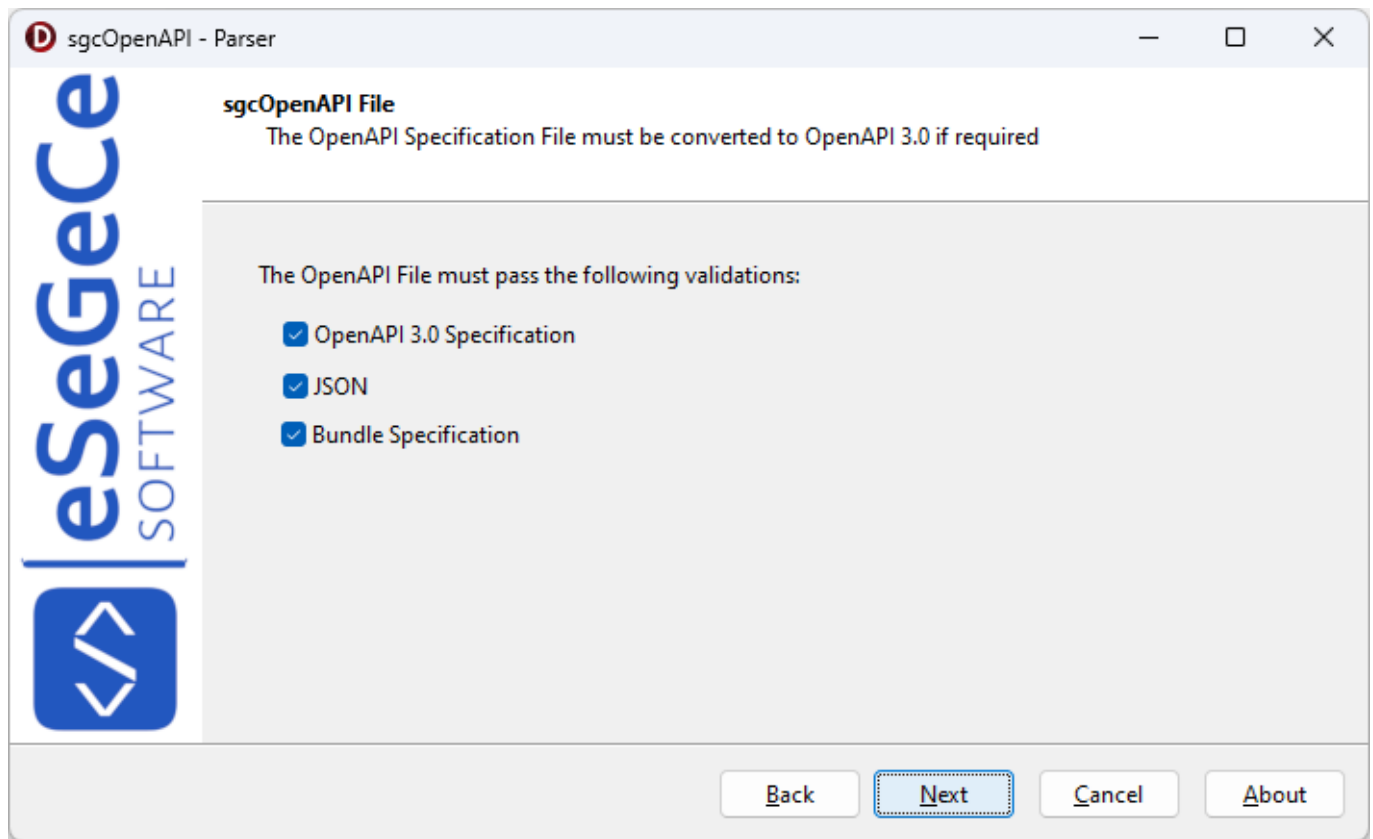
Importando a Especificação OpenAPI

O primeiro passo é importar a especificação openAPI. Uma vez que você tenha a especificação openAPI 3.0 em formato JSON, pode gerar os arquivos Delphi necessários utilizando nosso **OpenAPI WebService**. Siga os passos abaixo:

- **Execute o sgcOpenAPI Parser** e selecione a especificação a importar ou a URL a baixar.



- A **especificação é verificada** e, se for compatível com o Parser, você pode continuar para as próximas etapas. Se a especificação utilizar uma versão antiga como o swagger 2.0, ela será convertida automaticamente. Se a especificação estiver dividida em múltiplos schemas, o parser tentará agrupar em uma única especificação. Se houver algum problema durante a conversão do arquivo, uma mensagem de erro aparecerá.



- Agora você pode **personalizar as seguintes opções** antes de analisar o documento
 - **ClassName:** este é o ClassName principal; por padrão, começa com "sgcOpenAPI_" e adiciona o nome do arquivo da especificação.
 - **Namespace:** este é o nome do arquivo pascal gerado; por padrão, é o mesmo nome do arquivo de especificação com a extensão ".pas".
 - **Create Classes:** se marcado, criará as classes a partir do arquivo de especificação. Exemplo: se uma requisição exige enviar um objeto JSON, e este objeto JSON é especificado, o parser criará uma classe com os campos do objeto JSON.
 - **Enable Classes:** se marcado, habilita as Classes Geradas a partir do arquivo de especificação para uso com objetos JSON (Requer Rad Studio XE7+).
 - **Documentation:** se marcado, o parser adicionará comentários aos campos, classes e métodos.
 - **Authentication:** selecione qualquer autenticação, se existir.
 - **None:** a API não faz uso de nenhum método de autenticação.
 - **Basic:** a API faz uso de autenticação BASIC.
 - **Token:** é necessário enviar um Token como um Cabeçalho HTTP. Este token é obtido de qualquer outro método externo.
 - **OAuth2:** a requisição usará OAuth2 para autenticar as Requisições HTTP.
 - **JWT:** a requisição usará JWT para autenticar as Requisições HTTP.

sgcOpenAPI Parser Options
Customize the options of the OpenAPI parser.

Classname:

Namespace:

Options: ☒ Create Classes ☒ Enable Classes ☒ Documentation

Authentication: ☒ None ☐ Basic ☐ Token ☐ OAuth2 ☐ JWT

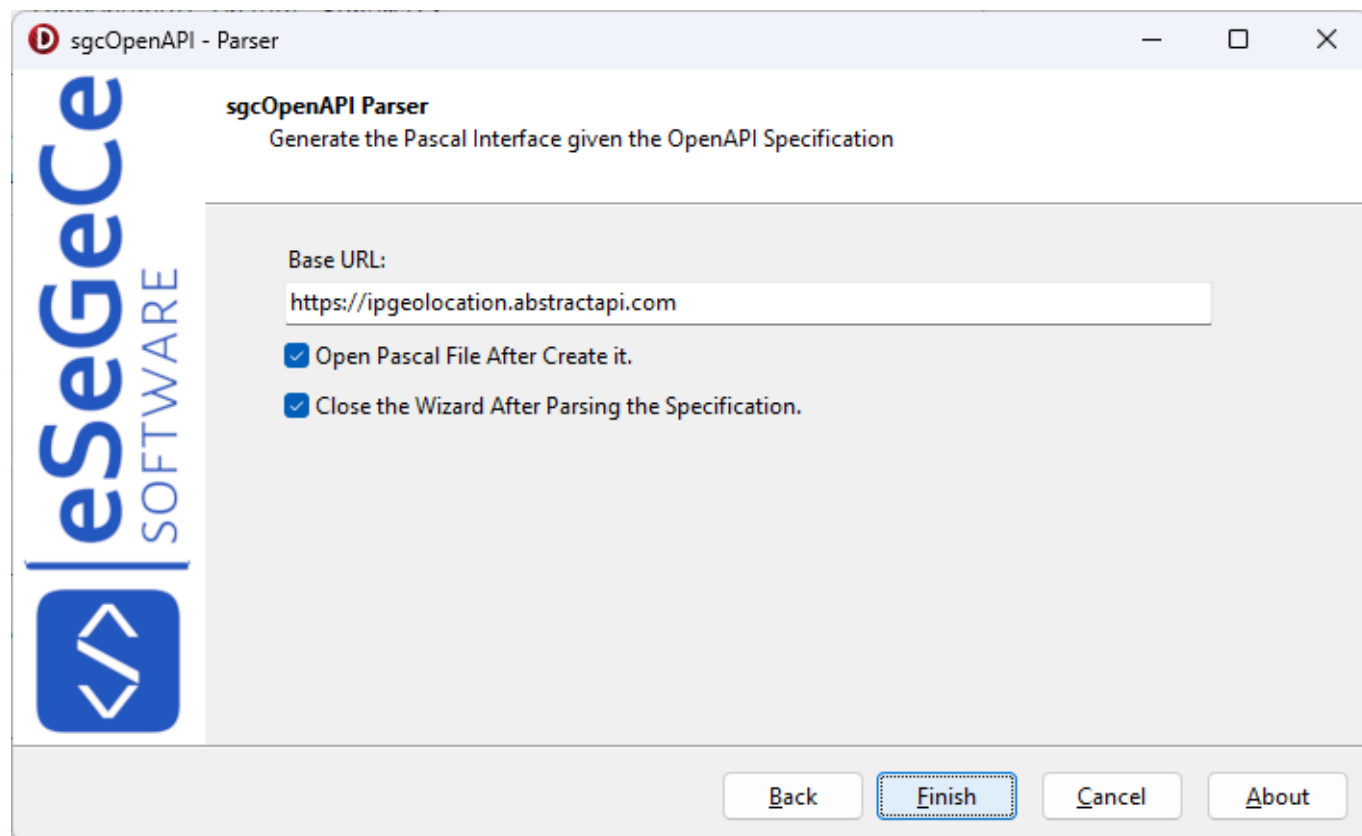
- Agora, uma **grade com uma lista de métodos analisados** será exibida como informação. Por padrão, o parser usa o OperationId como nome para os métodos criados, mas aqui você pode usar o summary como nome do método se o OperationId não estiver definido ou o valor não for válido.

sgcOpenAPI Parsed Methods
Find below the list of parsed methods.

Import	Method	Description
☒	Retrieve_the_location_of_an_IP_address	Retrieve the location of an IP address

☐ Method Name From Summary

- Por fim, verifique se a **Base URL** tem o valor correto e aqui você pode personalizar se o arquivo gerado será aberto automaticamente após a criação e se o wizard será fechado após gerar o arquivo.



- Pressione **Finish** para analisar a especificação e gerar o arquivo pascal.

Exemplo

Utilizarei uma especificação openAPI simples utilizada pela abstractapi.com para recuperar a localização de um Endereço IP.

Antes de testar a demo, você deve criar uma conta gratuita em abstractapi.com para obter uma API Key.

<https://app.abstractapi.com/users/signup>

Instale o [sgcOpenAPI Parser Setup](#), abra o sgcOpenAPI.exe e importe a especificação OpenAPI localizada na pasta "Demos\abstractapi.com\geolocation". Uma vez importada e armazenada na mesma pasta da demo com o nome "geolocation.pas", abra o projeto da demo, compile e execute-o. Preencha a chave de API obtida do site Abstractapi e pressione Geolocation.

sgcOpenAPI - Geolocation

API Key: *****

IP Address: 1.1.1.1

Geolocation

```
{
  "ip_address": "1.1.1.1",
  "city": "Sydney",
  "city_geoname_id": 2147714,
  "region": "New South Wales",
  "region_iso_code": "NSW",
  "region_geoname_id": 2155400,
  "postal_code": "1001",
  "country": "Australia",
  "country_code": "AU",
  "country_geoname_id": 2077456,
  "country_is_eu": false,
  "continent": "Oceania",
  "continent_code": "OC",
  "continent_geoname_id": 6255151,
  "longitude": 151.209,
  "latitude": -33.8688,
  "security": {
    "is_vpn": false
  },
  "timezone": {
    "name": "",
    "abbreviation": "",
    "gmt_offset": "",
    "current_time": "",
    "is_dst": ""
  },
  "flag": {
    "emoji": "ð\u0080",
    "unicode": "U+1F1E6 U+1F1FA",
    "png": "https://static.abstractapi.com/country-flags/AU_flag.png",
    "svg": "https://static.abstractapi.com/country-flags/AU_flag.svg"
  },
  "currency": {
    "currency_name": "Australian Dollars",
    "currency_code": "AUD"
  },
  "connection": {
    "autonomous_system_number": 13335,
    "autonomous_system_organization": "CLOUDFLARENET",
    "connection_type": "Corporate",
    "isp_name": "Cloudflare, Inc.",
    "organization_name": null
  }
}
```

OpenAPI | Additional Properties

Um dicionário (também conhecido como mapa, hashmap ou array associativo) é um conjunto de pares chave/valor. O OpenAPI permite que você defina dicionários onde as chaves são strings. Para definir um dicionário, use `type: object` e use a palavra-chave `additionalProperties` para especificar o tipo dos valores nos pares chave/valor. Por exemplo, um dicionário string-para-string como este:

```
{
  "en": "English",
  "fr": "French"
}
```

O OpenAPI Parser utiliza as classes JSON da Embarcadero, e ao converter um `TDictionary` para JSON, em vez de criar pares chave/valor, ele cria todos os objetos internos do `TDictionary`, de modo que a saída fica incorreta. O mesmo se aplica ao tentar converter uma string json em um objeto `TDictionary`.

Convert AdditionalProperties to JSON

O Parser OpenAPI cria as classes `AdditionalProperties` como do tipo `TsgcAdditionalProperties`, portanto, se você precisar converter uma classe para string json, deve criar uma nova classe que herde de `TsgcAdditionalProperties`, criar todos os campos que precisar e então atribuir à propriedade da classe.

Exemplo: dada uma classe com 2 propriedades, uma delas é uma propriedade `Additional`, e a saída json deve ser

```
{"Property1":"value1", "AdditionalProperties":{"key1":"value1", "key2":"value"}
```

```
TMyClass = class(TsgcOpenAPIClass)
private
  FProperty1: string;
  FAdditionalProperties: TsgcAdditionalProperties;
public
  property Property1: string read FProperty1 write FProperty1;
  property AdditionalProperties: TsgcAdditionalProperties read FAdditionalProperties write FAdditionalProperties;
end;
```

Crie uma nova classe que herda de `TsgcAdditionalProperties` e crie 2 propriedades

```
TCustomAdditionalProperties = class(TsgcAdditionalProperties)
private
  FKey1: string;
  FKey2: string;
public
  property Key1: string read FKey1 write FKey1;
  property Key2: string read FKey2 write FKey2;
end;
```

Por fim, atribua esta classe à propriedade `AdditionalProperties`:

```
TMyClass *oClass = new TMyClass();
oClass->Property1 = "value1";
TCustomAdditionalProperties *oAdditionalProperties = new TCustomAdditionalProperties();
oAdditionalProperties->Key1 = "value1";
oAdditionalProperties->Key2 = "value2";
oClass->AdditionalProperties = oAdditionalProperties;
```

Converter JSON em AdditionalProperties

Devido às limitações das classes JSON, o OpenAPI Generator cria um método adicional para carregar os AdditionalProperties na propriedade `TsgcAdditionalProperties.Dictionary` após a string JSON ser analisada. Então, para acessar o conteúdo dos AdditionalProperties, basta acessar a propriedade `TsgcAdditionalProperties.Dictionary`.

OpenAPI | Command Line

O **sgcOpenAPI Parser** inclui uma interface de linha de comando que converte uma especificação OpenAPI em um arquivo de cliente Delphi (Pascal). O parser valida e normaliza a especificação (OpenAPI 3 JSON + esquemas agrupados) antes de gerar a saída.

Usage

Sintaxe básica

```
sgcOpenAPI.exe -i "c:\openapi.json" -o "c:\openapi.pas"
```

Parâmetros

Todos os parâmetros são insensíveis a maiúsculas e minúsculas. As formas curta e longa são equivalentes.

- **-i -input** : **[obrigatório]** Caminho para a especificação OpenAPI ou uma URL para download. Se a entrada for YAML ou Swagger 1/2, será convertida automaticamente para OpenAPI 3 JSON.
- **-o -output** : **[obrigatório]** Arquivo Pascal de saída a gerar (cliente Delphi).
- **-nc -nclasses** : **[opcional]** Não gerar classes Delphi para schemas (apenas interfaces / métodos de cliente).
- **-nj -njson** : **[opcional]** Não habilite a serialização JSON para as classes geradas.
- **-nd -ndoc** : **[opcional]** Não gera comentários de documentação na saída.
- **-m -method** : **[opcional]** Estratégia de nomenclatura de métodos para os métodos gerados:
 - **0** : Utilizar **OperationId** (padrão).
 - **1** : Utilizar **Summary**.
 - **2** : Utilizar o caminho do **Endpoint**.
- **-a -auth** : **[opcional]** Modo de autenticação a habilitar no cliente gerado:
 - **0** : Nenhum.
 - **1** : Basic.
 - **2** : Token.
 - **3** : OAuth2.
 - **4** : JWT.
- **-u -url** : **[opcional]** Sobrepuê a base URL detectada no arquivo OpenAPI.
- **-l -log** : **[opcional]** Mostra a saída de log durante o parsing.
- **-user** : **[opcional]** Nome de usuário da licença (necessário apenas quando a licença não foi ativada).
- **-password** : **[opcional]** Senha da licença (necessária apenas quando a licença não foi ativada).
- **-h -help** : Mostra a ajuda da linha de comando.

Exemplos

1) Gerar cliente Delphi a partir de um arquivo local

```
sgcOpenAPI.exe -i "c:\specs\openapi.json" -o "c:\clients\orders.pas"
```

2) Gerar cliente a partir de URL, desabilitar classes e documentação

```
sgcOpenAPI.exe -i "https://api.example.com/openapi.json" -o "c:\clients\api.pas" -nc -nd
```

3) Use Summary como nomes de método e habilite OAuth2

```
sgcOpenAPI.exe -i "c:\specs\openapi.json" -o "c:\clients\auth.pas" -m 1 -a 3
```

4) Sobrescrever a Base URL e mostrar a saída de log

```
sgcOpenAPI.exe -i "c:\specs\openapi.json" -o "c:\clients\override.pas" -u "https://api.example.com/v2" -l
```

Notas

- Se a especificação de entrada não for válida ou não for um arquivo OpenAPI 3 JSON empacotado, o parser tentará convertê-la e empacotá-la automaticamente antes da geração.
- Ao utilizar URLs, o parser baixa a especificação antes de validar e gerar a saída.

OpenAPI | Biblioteca

O Parser OpenAPI pode ser usado como biblioteca compartilhada (DLL), permitindo gerar clientes Pascal a partir das suas próprias aplicações sem invocar a ferramenta de linha de comando. A biblioteca exporta um pequeno conjunto de funções que encapsulam os mesmos recursos do parser disponíveis na CLI.

Library functions

- **CreateOpenAPIPascalFile** - cria um arquivo de cliente Pascal a partir de uma especificação OpenAPI usando as opções padrão.
- **CreateOpenAPIPascalFileEx** - cria um arquivo de cliente Pascal com controle total sobre as opções do parser.
- **GetOpenAPILastErrors** - retorna a última mensagem de erro produzida pelo parser.

CreateOpenAPIPascalFile

Gera um arquivo de cliente Pascal utilizando as opções padrão do parser (gerar classes, suporte a JSON, documentação, nomes de método a partir de **OperationId**, sem autenticação, sem substituição de base URL).

```
typedef bool (__stdcall *TCreateOpenAPIPascalFile)(const char* aInputFile, const char* aOutputFile);
```

Parâmetros

- **aInputFile**: caminho completo (ou URL) da especificação OpenAPI a ser analisada. Se o arquivo for YAML ou Swagger 1/2, o parser irá normalizá-lo para OpenAPI 3 JSON antes de gerar a saída.
- **aOutputFile**: caminho completo da unidade Pascal a criar.

Valor de retorno

Retorna **True** se o arquivo for gerado com sucesso, caso contrário **False**. Se falhar, chame **GetOpenAPILastErrors** para recuperar a mensagem de erro.

Exemplo

CreateOpenAPIPascalFileEx

Gera um arquivo de cliente Pascal com controle explícito sobre as opções do parser.

```
typedef bool (__stdcall *TCreateOpenAPIPascalFileEx)(
    const char* aInputFile, const char* aOutputFile,
    bool aGenerateClasses, bool aGenerateJSON, bool aGenerateDocumentation,
    int aMethodName, int aAuthentication, const char* aBaseURL, bool aEnableLog);
```

Parâmetros

- **aInputFile**: caminho completo (ou URL) da especificação OpenAPI a ser analisada.
- **aOutputFile**: caminho completo da unidade Pascal a criar.
- **aGenerateClasses**: quando **True**, o gerador cria os modelos Pascal (classes) definidos no schema. Quando **False**, somente a interface do cliente é gerada.
- **aGenerateJSON**: quando **True**, helpers de serialização JSON são gerados nas classes. Quando **False**, o suporte a JSON é omitido.

- **aGenerateDocumentation:** quando **True**, um arquivo de documentação para a especificação OpenAPI é gerado.
- **aMethodName:** controla como os nomes de métodos são gerados:
 - **0** = OperationId (padrão).
 - **1** = Summary.
 - **2** = Endpoint.
- **aAuthentication:** adiciona auxiliares de autenticação ao cliente:
 - **0** = None.
 - **1** = Basic.
 - **2** = Token.
 - **3** = OAuth2.
 - **4** = JWT.
- **aBaseURL:** sobrescreve a URL base da especificação OpenAPI. Passe uma string vazia para manter o valor original.
- **aEnableLog:** quando **True**, o parser escreve mensagens de progresso e de validação que podem ajudar na solução de problemas.

Valor de retorno

Retorna **True** se o arquivo for gerado com sucesso, caso contrário **False**. Se falhar, chame **GetOpenAPILastErrors** para recuperar a mensagem de erro.

Exemplo

GetOpenAPILastErrors

Retorna uma mensagem descrevendo o último erro (erro de validação, conversão ou parsing) produzido pelo parser. Chame esta função depois que **CreateOpenAPIPascalFile** ou **CreateOpenAPIPascalFileEx** retornar **False**.

```
typedef const char* (__stdcall *TGetOpenAPILastErrors)();
```

Valor de retorno

Retorna a contagem de erros

Exemplo

OpenAPI | API

O Parser do sgcOpenAPI permite utilizar uma dll personalizada para obter informações mais detalhadas do arquivo pascal analisado; basta criar uma nova dll com o método **sgcOpenAPI_CreateSection** e copiar a biblioteca para a mesma pasta onde está o parser. Toda vez que uma nova seção de arquivo é criada, o método CreateSection da api será chamado e você pode modificar o conteúdo, se necessário. Veja abaixo um exemplo simples em delphi:

```
enum TsgcOpenAPIPascalSectionType {
    soapistHeaderStart, soapistHeaderEnums,
    soapistHeaderClasses, soapistHeaderMainClass, soapistHeaderClassesStructure,
    soapistHeaderMainInstance, soapistHeaderEnd, soapistImplementationStart,
    soapistImplementationClasses, soapistImplementationEnd
};

extern "C" __declspec(dllexport) void __stdcall sgcOpenAPI_CreateSection(
    int aType, wchar_t* &aValue)
{
    switch ((TsgcOpenAPIPascalSectionType)aType) {
        case soapistHeaderStart: break;
        case soapistHeaderEnums: break;
        case soapistHeaderClasses: break;
        case soapistHeaderMainClass: break;
        case soapistHeaderClassesStructure: break;
        case soapistHeaderMainInstance: break;
        case soapistHeaderEnd: break;
        case soapistImplementationStart: break;
        case soapistImplementationClasses: break;
        case soapistImplementationEnd: break;
    }
}
```

OpenAPI | Client

TsgcOpenAPI_Client é um componente não visual que encapsula os principais métodos e propriedades para fazer requisições HTTP a partir de uma especificação OpenAPI.

Toda interface OpenAPI criada com o sgcOpenAPI Parser possui **2 métodos**

1. **GetOpenAPIClient:** é uma função singleton que retorna uma instância da classe principal; se não existir, ela a cria automaticamente.
2. **FreeOpenAPIClient:** libera a classe principal se ela estiver criada.

Exemplo

Utilize a Abstractapi para recuperar a localização de um Endereço IP.

```
GetOpenAPIClient->Retrieve_the_location_of_an_IP_address("your api", "80.258.15.2");
```

Autenticação

- **Basic:** usa um nome de usuário/senha como cabeçalho HTTP para autenticar.
 - **UserName:** nome do usuário.
 - **Password:** segredo.
- **OAuth2:** autentica utilizando OAuth2, suporta 2 tipos de Autorização:
 - **auth2Code:** É utilizado para realizar autenticação e autorização na maioria dos tipos de aplicações, incluindo aplicações de página única, aplicações web e aplicações instaladas nativamente. O fluxo permite que as aplicações adquiram com segurança access_tokens que podem ser utilizados para acessar recursos protegidos, bem como refresh tokens para obter access_tokens adicionais, e ID tokens para o usuário autenticado.
 - **auth2ClientCredentials:** Este tipo de grant é comumente utilizado para interações server-to-server que devem ser executadas em segundo plano, sem interação imediata com um usuário. Esses tipos de aplicações são frequentemente chamados de daemons ou contas de serviço.
 - Leia mais sobre [OAuth2](#).
- **JWT:** autentica usando JWT. Leia mais sobre [JWT](#).
- **Token:** você pode passar um bearer token ou qualquer outro valor de token personalizado.

TLSOptions

Permite configurar como se conectar a servidores SSL/TLS seguros usando o protocolo HTTP/1

ALPNProtocols: lista dos protocolos ALPN que serão enviados ao servidor.

RootCertFile: caminho para o arquivo de certificado raiz.

CertFile: caminho para o arquivo de certificado.

KeyFile: caminho para o arquivo de chave do certificado.

Password: se o certificado está protegido com uma senha, defina-a aqui.

VerifyCertificate: se o certificado deve ser verificado, habilite esta propriedade.

VerifyDepth: é uma propriedade Integer que representa o número máximo de links permitidos quando a verificação é realizada para o certificado X.509.

Version: por padrão usa TLS 1.0; se o servidor exigir uma versão TLS superior, ela pode ser selecionada aqui.

IOHandler: selecione qual biblioteca você usará para a conexão usando TLS.

iohOpenSSL: utiliza a biblioteca OpenSSL e é o padrão para componentes Indy. Requer implantar as bibliotecas openssl para win32/win64.

iohSChannel: utiliza o Secure Channel, que é um protocolo de segurança implementado pela Microsoft para Windows, não requer a implantação de bibliotecas openssl. Funciona apenas no Windows 32/64 bits.

OpenSSL_Options: configuração das bibliotecas openssl.

APIVersion: permite definir qual API OpenSSL será usada.

oslAPI_1_0: usa a API 1.0 do OpenSSL, é a mais recente suportada pelo Indy

oslAPI_1_1: usa a API 1.1 do OpenSSL, requer nossa biblioteca Indy personalizada e permite usar as bibliotecas OpenSSL 1.1.1 (com suporte a TLS 1.3).

oslAPI_3_0: utiliza a API 3.0 do OpenSSL, requer nossa biblioteca Indy personalizada e permite utilizar as bibliotecas OpenSSL 3.0.0 (com suporte a TLS 1.3).

LibPath: aqui você pode configurar onde estão localizadas as bibliotecas openSSL

oslpNone: este é o padrão, as bibliotecas openSSL devem estar na mesma pasta onde está o binário ou em um caminho conhecido.

oslpDefaultFolder: define automaticamente o caminho openSSL onde as bibliotecas devem estar localizadas para todas as personalidades do IDE.

oslpCustomFolder: se esta for a opção selecionada, defina o caminho completo na propriedade LibPathCustom.

LibPathCustom: quando LibPath = oslpCustomFolder, defina aqui o caminho completo onde estão localizadas as bibliotecas openSSL.

UnixSymLinks: habilita ou desabilita o carregamento de SymLinks em sistemas Unix (por padrão está habilitado, exceto no OSX64):

oslsSymLinksDefault: por padrão estão habilitados, exceto sob OSX64 (após o MacOS Monterey falha ao tentar carregar a biblioteca sem versão.).

oslsSymLinksLoadFirst: Carrega os SymLinks e faz isso antes de tentar carregar as bibliotecas de versão.

oslsSymLinksLoad: Carrega os SymLinks após tentar carregar as bibliotecas de versão.

oslsSymLinksDontLoad: não carrega os SymLinks.

SChannel_Options: permite que você utilize um certificado do Windows Certificate Store.

CertHash: é o Hash do certificado. Você pode encontrar o Hash do certificado executando um comando dir no powershell.

CipherList: aqui você pode definir quais Ciphers serão usados (separados por ":"). Exemplo: CALG_AES_256:CALG_AES_128

CertStoreName: o nome do store onde o certificado está armazenado. Selecione um dos abaixo:

scsnMY (o padrão)

scsnCA

scsnRoot

scsnTrust

CertStorePath: o caminho do store onde o certificado está armazenado. Selecione um dos abaixo:

scspStoreCurrentUser (o padrão)

scspStoreLocalMachine

Opções de Proxy

Utilize esta propriedade para configurar as conexões através de um proxy.

Enabled: defina como true para habilitar conexões de proxy.

Host: Endereço do servidor proxy

Port: Porta do servidor proxy

UserName/Password: Autenticação para conectar ao proxy, apenas se necessário.

ProxyType: os seguintes proxies são suportados:

- HTTP
- Socks4
- Socks4A
- Socks5

Log

Se a propriedade Log estiver habilitada, ela salva as mensagens de socket em um arquivo de log especificado, útil para depuração.

Log: habilite se você quiser salvar as requisições HTTP em um arquivo de texto.

LogFileName: caminho completo para o nome do arquivo.

Properties

Outras propriedades que podem ser utilizadas para personalizar o cliente OpenAPI:

EncodeBodyAsUTF8: se habilitado, o corpo JSON é codificado como UTF8 (por padrão false).

Eventos

Encontre abaixo a lista dos eventos que você pode tratar ao utilizar o OpenAPI Client.

OnBeforeRequest

Este evento é chamado antes que a requisição HTTP seja chamada. Permite personalizar os nomes dos Parâmetros, Headers, segurança... Veja abaixo um exemplo de como substituir o nome de alguns parâmetros.

```
void __fastcall OnBeforeRequestEvent(TObject *Sender, const TsgcOpenAPIRequest *aRequest)
{
    for (int i = 0; i < aRequest->Parameters->Count; i++)
    {
        TsgcOpenAPIParameter *oParameter = aRequest->Parameters->Items[i];
        if (oParameter->_Name == "meta-modified-from")
            oParameter->_Name = "eventDateTime-from";
        if (oParameter->_Name == "meta-modified-to")
            oParameter->_Name = "eventDateTime-to";
    }
}
```

OnUpload

Este evento é chamado quando um arquivo é enviado (upload); você pode utilizar este evento para saber o progresso do upload.

OnDownload

Este evento é chamado quando um arquivo é baixado, você pode usar este evento para conhecer o progresso do download.

OnSSLVerifyPeer

Se a verificação de certificado estiver habilitada, neste evento você pode verificar e decidir se aceita o certificado do servidor.

OnSSLGetHandler

Este evento é gerado antes do handler SSL ser criado; você pode criar aqui seu próprio SSL Handler (precisa ser herddado de TIdServerIOHandlerSSLBase ou TIdIOHandlerSSLBase) e definir as propriedades necessárias

OnSSLAfterCreateHandler

Se nenhum objeto SSL personalizado foi criado, ele é criado por padrão usando o handler OpenSSL. Você pode acessar as propriedades do SSL Handler e modificá-las se necessário

OpenAPI | Server

O **sgcOpenAPI Server** hospeda um serviço HTTP cujo contrato é descrito por uma especificação OpenAPI 3.0. As rotas são construídas automaticamente a partir da seção **paths** da especificação, uma Swagger UI é exposta pronta para uso e as requisições recebidas podem ser validadas contra os JSON-Schemas declarados por cada operação. Dois componentes são fornecidos para hospedar o servidor:

- **TsgcHTTPServer_OpenAPI**: servidor HTTP autônomo. Autocontido, incorpora um servidor HTTP Indy, não requer sgcWebSockets. Recomendado quando você só precisa expor um serviço OpenAPI.
- **TsgcWSAPIServer_OpenAPI**: compatível com sgcWebSockets. Anexa-se a um **TsgcWebSocketHTTPServer** existente, de modo que o mesmo servidor trate tanto o tráfego WebSocket quanto o tráfego HTTP/OpenAPI.

Ambos os componentes compartilham o mesmo engine, opções e eventos.

TsgcHTTPServer_OpenAPI

Servidor standalone definido na unit **sgcHTTPServer_OpenAPI**. Configure a porta de escuta, carregue uma spec OpenAPI e ative o servidor.

TsgcWSAPIServer_OpenAPI

Compatível com sgcWebSockets, definido na unit **sgcWebSocket_Server_API_OpenAPI**. Atribua um **TsgcWebSocketHTTPServer** existente à propriedade **Server** para que uma única instância de servidor possa despachar tanto quadros WebSocket quanto requisições HTTP OpenAPI. Internamente, ele delega ao mesmo engine utilizado pelo componente autônomo, portanto a superfície da API é idêntica.

Built-in Endpoints

O servidor serve automaticamente os seguintes endpoints (cada um pode ser desabilitado via **OpenAPIOptions.Endpoint**):

/openapi.json: retorna a especificação OpenAPI carregada, controlada por **OpenAPIOptions.Endpoint.ServeSpec**.
/docs: Swagger UI embutido para navegar e testar a API, controlado por **OpenAPIOptions.Endpoint.ServeSwaggerUI**.
OPTIONS *: respostas de preflight CORS, enviadas quando **OpenAPIOptions.CORS.Enabled** é True.

OpenAPIOptions

Propriedade de configuração de nível superior, do tipo **TsgcOpenAPIServer_Options**. Agrupa três sub-objetos:

Endpoint: endpoints integrados e carregamento de spec.

BasePath: prefixo de URL opcional prepended a cada rota declarada na spec.

SpecFile: se atribuído, o arquivo é carregado automaticamente quando o servidor se torna **Active**.

ServeSpec: se True (padrão), a especificação carregada é servida em **/openapi.json**.

ServeSwaggerUI: se True (padrão), o Swagger UI é servido em **/docs**.

Validation: flags de validação de requisição, todas com padrão False.

ValidateRequest: interruptor principal que habilita a validação para toda a requisição.

ValidateRequestBody: valida o corpo da requisição em relação ao JSON-Schema declarado na spec.

ValidateQueryParams: valida os parâmetros de query contra seu schema declarado.

ValidatePathParams: valida os parâmetros de caminho contra o esquema declarado.

ValidateRequired: rejeita a requisição quando um parâmetro declarado como obrigatório está ausente.

CORS: suporte cross-origin.

Enabled: se True, os cabeçalhos CORS são emitidos em cada resposta e as requisições de preflight **OPTIONS** são respondidas automaticamente.

AllowOrigins: valor de **Access-Control-Allow-Origin** (padrão *).

AllowHeaders: valor de **Access-Control-Allow-Headers** (padrão **Content-Type, Authorization**).

AllowMethods: valor de **Access-Control-Allow-Methods** (padrão **GET, POST, PUT, DELETE, PATCH, OPTIONS**).

Eventos

Encontre abaixo a lista de eventos que você pode tratar ao usar o OpenAPI Server.

OnBeforeRequest: disparado antes de a requisição ser despachada para **OnRequest**. Defina o parâmetro **Accept** como **False** para rejeitar a requisição com uma resposta 403.

OnRequest: manipulador principal. Inspeção **aOperationId** e utilize **aContext** para ler a requisição e escrever a resposta. Defina **Handled** como **True** após produzir uma resposta.

OnAfterRequest: disparado depois que o handler do usuário retorna. Útil para log ou métricas.

OnAuthenticate: dispara antes do despacho quando a autenticação é necessária. Defina **Authenticated** como **False** para rejeitar com uma resposta 401.

OnValidationError: disparado quando uma ou mais regras de validação falham. Inspeção **aErrors** e defina **Continue** como **False** para rejeitar a requisição com uma resposta 400.

OnException: disparado quando uma exceção não tratada se propaga a partir do handler do usuário. Ajuste **aResponseCode** se o padrão 500 deve ser substituído.

Spec-First vs Code-First

O servidor suporta duas abordagens de autoria:

- **Spec-First:** o arquivo JSON/YAML do OpenAPI 3.0 é a fonte da verdade; o servidor descobre as rotas analisando-o.
- **Code-First:** uma classe Delphi anotada com atributos RTTI é a fonte da verdade; a especificação OpenAPI é gerada na inicialização. Requer Delphi XE7 ou posterior.

OpenAPI | Server | Spec-First

Na abordagem **Spec-First**, a especificação OpenAPI 3.0 (JSON ou YAML) é a fonte da verdade: ela descreve cada caminho, operação, parâmetro e schema. Quando o servidor inicia, ele analisa a especificação e constrói a tabela de rotas automaticamente; você só precisa fornecer a lógica de negócio para cada operação.

Este fluxo de trabalho é o recomendado quando o contrato da API já existe, quando é compartilhado com consumidores ou quando é mantido fora da base de código Delphi.

Carregando a Spec

Use **LoadFromFile** para carregar a spec do disco, ou **LoadFromString** para carregá-la de uma string em memória (por exemplo, uma spec recuperada de um banco de dados ou gerada em tempo de execução).

Uma vez carregada, a tabela de rotas é construída a partir da seção **paths** da spec. Segmentos de template como `/pets/{petId}` são reconhecidos automaticamente e os valores capturados são expostos via `aContext.PathParamAsXxx`.

Tratamento de Operações

O servidor despacha cada requisição de entrada para o evento **OnRequest**, passando o **operationId** declarado na spec. Use uma cadeia **if/else** simples (ou uma consulta em dicionário) para rotear a operação ao manipulador correto.

Reading Request Data

O objeto **TsgcOpenAPIServerContext** expõe acessadores tipados para cada parte da requisição:

Escrevendo respostas

O objeto de contexto também fornece helpers para escrever a resposta. Os helpers mais comuns são **RespondJSON** e **RespondError**; para controle total sobre a resposta, defina `aContext.Response.Code` e `aContext.Response.ContentText` diretamente.

Validação

Quando os flags de validação em **OpenAPIOptions.Validation** estão habilitados, o servidor valida cada requisição contra os JSON-Schemas declarados na especificação antes de invocar **OnRequest**:

ValidateRequestBody: o corpo da requisição é verificado em relação ao esquema referenciado pelo **requestBody** da operação.

ValidateQueryParams: os parâmetros da query string são verificados em relação aos seus esquemas declarados.

ValidatePathParams: os parâmetros de caminho são verificados em relação aos seus schemas declarados.

ValidateRequired: a requisição é rejeitada quando um parâmetro declarado como obrigatório está ausente.

Quando a validação falha, o evento **OnValidationError** dispara com a lista de erros. Defina **Continue** como **False** para rejeitar a requisição com uma resposta 400, ou deixe-o como **True** para permitir que seu manipulador **OnRequest** decida.

Demo

Um exemplo completo spec-first é fornecido em **Demos/30.Server/02.SpecFirst/**.

OpenAPI | Server | Code-First

Na abordagem **Code-First**, a classe Delphi é a fonte da verdade: métodos, parâmetros e atributos RTTI descrevem o contrato da API, e a especificação OpenAPI 3.0 é gerada a partir deles na inicialização. Uma vez gerada, a spec é carregada no servidor com **LoadFromString** e a tabela de rotas é construída como se tivesse vindo de um arquivo.

O Code-First requer **Delphi XE7 ou posterior** porque depende do novo RTTI para inspecionar atributos. A funcionalidade reside na unit **sgcHTTP_OpenAPI_Server_CodeFirst** e é protegida pela diretiva de compilador **{\$IFDEF DXE7}**.

Definindo o serviço

Anote uma classe Delphi com atributos para descrever a API. Os corpos dos métodos são stubs: a lógica real fica no evento **OnRequest** do servidor; os métodos só existem para que o scanner RTTI possa ler suas assinaturas.

Atributos Disponíveis

Os seguintes atributos são reconhecidos pelo scanner. Atributos de nível de classe descrevem o serviço como um todo; atributos de nível de método descrevem uma única operação; atributos de nível de parâmetro descrevem de onde vem o valor.

sgcServiceContract(name, description, version): em nível de classe. Preenche o bloco **info** do OpenAPI.

sgcRoute(path): caminho base em nível de classe, ou rota em nível de método relativa ao caminho base da classe.

sgcHttpGet, sgcHttpPost, sgcHttpPut, sgcHttpDelete, sgcHttpPatch, sgcHttpHead, sgcHttpOptions: nível de método. Declara o verbo HTTP da operação.

sgcSummary(text): nível de método. Resumo curto exibido na especificação e na Swagger UI.

sgcDescription(text): de nível de método. Descrição longa para a spec.

sgcTag(name): nível de método. Agrupa operações sob a mesma tag na Swagger UI.

sgcResponse(code, description): nível de método. Declara uma resposta com seu código de status HTTP e descrição. Pode ser repetido para declarar múltiplas respostas.

sgcFromPath, sgcFromQuery, sgcFromBody, sgcFromHeader: nível de parâmetro. Declare de onde o parâmetro é lido.

sgcRequired: nível de parâmetro. Marca o parâmetro como obrigatório; parâmetros ausentes são relatados pela validação quando **ValidateRequired** está habilitado.

Gerando e Carregando a Spec

Utilize **TsgcOpenAPICodeFirstScanner** para gerar a especificação OpenAPI 3.0 a partir da classe anotada e, em seguida, carregue o resultado no servidor.

Tratamento de Operações

O scanner deriva o **operationId** a partir do nome do método: o método **ListTasks** torna-se a operação **ListTasks**. Faça o dispatch em **OnRequest** utilizando o mesmo **operationId**.

Demo

Um exemplo completo de code-first é fornecido em **Demos/30.Server/01.CodeFirst/**.

OpenAPI | Amazon AWS

O **sgcOpenAPI Amazon AWS Client** (`TsgcOpenAPI_Amazon_Client`) tem seu próprio OpenAPI Client que herda de [TsgcOpenAPI_Client](#).

Este componente tem uma propriedade chamada **AmazonOptions** que inclui todas as configurações necessárias para se conectar aos servidores da Amazon AWS.

AmazonOptions

Em `AmazonOptions` você pode definir as **AccessKey** e **SecretKey** necessárias (que devem ser geradas previamente a partir da sua Conta Amazon), para autenticar contra os Servidores Amazon AWS.

Uma access key concede acesso programático aos seus recursos. Isso significa que você deve proteger a access key com o mesmo cuidado que as credenciais de login do usuário raiz da conta AWS.

É uma boa prática fazer o seguinte:

1. **Crie um usuário IAM**, e então **defina as permissões desse usuário** da forma mais restrita possível.
2. **Crie a access key** sob esse usuário IAM.

Depois de obter as credenciais, defina nas seguintes propriedades:

- `AmazonOptions.AccessKey`
- `AmazonOptions.SecretKey`

A propriedade `AmazonOptions.JSON` permite definir se as respostas estão em JSON ou XML.

Funções IAM, usuários no AWS IAM Identity Center (sucessor do AWS Single Sign-On) e usuários federados possuem credenciais de segurança temporárias. As credenciais de segurança temporárias expiram após um período de tempo definido ou quando o usuário encerra sua sessão. Você pode definir o token para credenciais temporárias na propriedade:

- `AmazonOptions.SessionToken`

Most common uses

- **Configuration**
 - [Amazon AWS Credentials](#)
- **APIs**
 - [Amazon AWS S3](#)

sgcOpenAPI AWS SDK

Encontre abaixo uma lista das APIs atualmente disponíveis.

- Access Analyzer
- Alexa For Business
- Amazon API Gateway
- Amazon AppConfig
- Amazon Appflow
- Amazon AppIntegrations Service
- Amazon AppStream

- Amazon Athena
- Amazon Augmented AI Runtime
- Amazon Chime
- Amazon Chime SDK Identity
- Amazon Chime SDK Messaging
- Amazon CloudDirectory
- Amazon CloudFront
- Amazon CloudHSM
- Amazon CloudSearch
- Amazon CloudSearch Domain
- Amazon CloudWatch
- Amazon CloudWatch Application Insights
- Amazon CloudWatch Events
- Amazon CloudWatch Logs
- Amazon CodeGuru Profiler
- Amazon CodeGuru Reviewer
- Amazon Cognito Identity
- Amazon Cognito Identity Provider
- Amazon Cognito Sync
- Amazon Comprehend
- Amazon Connect Contact Lens
- Amazon Connect Customer Profiles
- Amazon Connect Participant Service
- Amazon Connect Service
- Amazon Data Lifecycle Manager
- Amazon Detective
- Amazon DevOps Guru
- Amazon DocumentDB with MongoDB compatibility
- Amazon DynamoDB
- Amazon DynamoDB Accelerator (DAX)
- Amazon DynamoDB Streams
- Amazon EC2 Container Registry
- Amazon EC2 Container Service
- Amazon Elastic Inference
- Amazon Elastic Block Store
- Amazon Elastic Compute Cloud
- Amazon Elastic Container Registry Public
- Amazon Elastic File System
- Amazon Elastic Kubernetes Service
- Amazon Elastic Transcoder
- Amazon ElastiCache
- Amazon Elasticsearch Service
- Amazon EMR
- Amazon EMR Containers
- Amazon EventBridge
- Amazon Forecast Query Service
- Amazon Forecast Service
- Amazon Fraud Detector
- Amazon FSx
- Amazon GameLift
- Amazon Glacier
- Amazon GuardDuty
- Amazon HealthLake
- Amazon Honeycode
- Amazon Import/Export Snowball
- Amazon Inspector
- Amazon Interactive Video Service
- Amazon Kinesis
- Amazon Kinesis Analytics
- Amazon Kinesis Firehose
- Amazon Kinesis Video Signaling Channels
- Amazon Kinesis Video Streams
- Amazon Kinesis Video Streams Archived Media
- Amazon Kinesis Video Streams Media
- Amazon Lex Model Building Service

- Amazon Lex Model Building V2
- Amazon Lex Runtime Service
- Amazon Lex Runtime V2
- Amazon Lightsail
- Amazon Location Service
- Amazon Lookout for Equipment
- Amazon Lookout for Metrics
- Amazon Lookout for Vision
- Amazon Machine Learning
- Amazon Macie
- Amazon Macie 2
- Amazon Managed Blockchain
- Amazon Mechanical Turk
- Amazon MemoryDB
- Amazon Mobile Analytics
- Amazon Neptune
- Amazon OpenSearch Service
- Amazon Personalize
- Amazon Personalize Events
- Amazon Personalize Runtime
- Amazon Pinpoint
- Amazon Pinpoint Email Service
- Amazon Pinpoint SMS and Voice Service
- Amazon Polly
- Amazon Prometheus Service
- Amazon QLDB
- Amazon QLDB Session
- Amazon QuickSight
- Amazon Redshift
- Amazon Rekognition
- Amazon Relational Database Service
- Amazon Route 53
- Amazon Route 53 Domains
- Amazon Route 53 Resolver
- Amazon S3 on Outposts
- Amazon SageMaker Edge Manager
- Amazon SageMaker Feature Store Runtime
- Amazon SageMaker Runtime
- Amazon SageMaker Service
- Amazon Simple Email Service
- Amazon Simple Notification Service
- Amazon Simple Queue Service
- Amazon Simple Storage Service
- Amazon Simple Systems Manager (SSM)
- Amazon Simple Workflow Service
- Amazon SimpleDB
- Amazon Textract
- Amazon Timestream Query
- Amazon Timestream Write
- Amazon Transcribe Service
- Amazon Translate
- Amazon WorkDocs
- Amazon WorkLink
- Amazon WorkMail
- Amazon WorkMail Message Flow
- Amazon WorkSpaces
- AmazonApiGatewayManagementApi
- AmazonApiGatewayV2
- AmazonMQ
- AmazonMWAA
- AmazonNimbleStudio
- AmplifyBackend
- Application Auto Scaling
- Application Migration Service
- Auto Scaling

- AWS Amplify
- AWS App Mesh
- AWS App Runner
- AWS Application Cost Profiler
- AWS Application Discovery Service
- AWS AppSync
- AWS Audit Manager
- AWS Auto Scaling Plans
- AWS Backup
- AWS Batch
- AWS Budgets
- AWS Certificate Manager
- AWS Certificate Manager Private Certificate Authority
- AWS Cloud Map
- AWS Cloud9
- AWS CloudFormation
- AWS CloudHSM V2
- AWS CloudTrail
- AWS CodeBuild
- AWS CodeCommit
- AWS CodeDeploy
- AWS CodePipeline
- AWS CodeStar
- AWS CodeStar connections
- Notificações AWS CodeStar
- AWS Comprehend Medical
- AWS Compute Optimizer
- AWS Config
- AWS Cost and Usage Report Service
- AWS Cost Explorer Service
- AWS Data Exchange
- AWS Data Pipeline
- AWS Database Migration Service
- AWS DataSync
- AWS Device Farm
- AWS Direct Connect
- AWS Directory Service
- AWS EC2 Instance Connect
- AWS Elastic Beanstalk
- AWS Elemental MediaConvert
- AWS Elemental MediaLive
- AWS Elemental MediaPackage
- AWS Elemental MediaPackage VOD
- AWS Elemental MediaStore
- AWS Elemental MediaStore Data Plane
- AWS Fault Injection Simulator
- AWS Global Accelerator
- AWS Glue
- AWS Glue DataBrew
- AWS Greengrass
- AWS Ground Station
- AWS Health APIs and Notifications
- AWS Identity and Access Management
- AWS Import/Export
- AWS IoT
- AWS IoT 1-Click Devices Service
- AWS IoT 1-Click Projects Service
- AWS IoT Analytics
- AWS IoT Core Device Advisor
- AWS IoT Data Plane
- AWS IoT Events
- AWS IoT Events Data
- AWS IoT Fleet Hub
- AWS IoT Greengrass V2
- AWS IoT Jobs Data Plane

- AWS IoT Secure Tunneling
- AWS IoT SiteWise
- AWS IoT Things Graph
- AWS IoT Wireless
- AWS Key Management Service
- AWS Lake Formation
- AWS Lambda
- AWS License Manager
- AWS Marketplace Catalog Service
- AWS Marketplace Commerce Analytics
- AWS Marketplace Entitlement Service
- AWS MediaConnect
- AWS MediaTailor
- AWS Migration Hub
- AWS Migration Hub Config
- AWS Mobile
- AWS Network Firewall
- AWS Network Manager
- AWS OpsWorks
- AWS OpsWorks CM
- AWS Organizations
- AWS Outposts
- AWS Performance Insights
- AWS Price List Service
- AWS Proton
- AWS RDS DataService
- AWS Resource Access Manager
- AWS Resource Groups
- AWS Resource Groups Tagging API
- AWS RoboMaker
- AWS Route53 Recovery Control Config
- AWS Route53 Recovery Readiness
- AWS S3 Control
- AWS Savings Plans
- AWS Secrets Manager
- AWS Security Token Service
- AWS SecurityHub
- AWS Server Migration Service
- AWS Service Catalog
- AWS Service Catalog App Registry
- AWS Shield
- AWS Signer
- AWS Single Sign-On
- AWS Single Sign-On Admin
- AWS Snow Device Management
- AWS SSO Identity Store
- AWS SSO OIDC
- AWS Step Functions
- AWS Storage Gateway
- AWS Support
- AWS Systems Manager Incident Manager
- AWS Systems Manager Incident Manager Contacts
- AWS Transfer Family
- AWS WAF
- AWS WAF Regional
- AWS WAFV2
- AWS Well-Architected Tool
- AWS X-Ray
- AWSKendraFrontendService
- AWSMarketplace Metering
- AWSServerlessApplicationRepository
- Braket
- CodeArtifact
- EC2 Image Builder
- Elastic Load Balancing

- FinSpace Public API
- FinSpace User Environment Management service
- Firewall Management Service
- Managed Streaming for Kafka
- Managed Streaming for Kafka Connect
- Redshift Data API Service
- Route53 Recovery Cluster
- Schemas
- Service Quotas
- Synthetics

OpenAPI Amazon AWS | Credentials

A AWS exige diferentes tipos de credenciais de segurança dependendo de como você acessa a AWS. Por exemplo, você precisa de um nome de usuário e senha para fazer login no AWS Management Console e precisa de **access keys** para fazer **chamadas programáticas à AWS**.

Considerações

- Certifique-se de salvar o seguinte em um local seguro: o endereço de email associado à sua conta AWS, o AWS account ID, a senha do root user e suas access keys de conta. Se você esquecer ou perder a senha do root user, deve ter acesso ao endereço de email associado à sua conta para redefini-la. Se você esquecer ou perder suas access keys, deve fazer login na sua conta para criar novas.
- Recomendamos fortemente que você crie um usuário IAM com permissões de administrador para usar nas tarefas diárias da AWS e guarde a senha e as chaves de acesso do usuário root. Use o usuário root apenas para as tarefas restritas ao usuário root.
- As credenciais de segurança são específicas da conta. Se você tiver acesso a várias contas AWS, terá credenciais separadas para cada conta.
- Não forneça suas credenciais AWS a terceiros.

Acesso programático

Você deve fornecer suas chaves de acesso da AWS para fazer chamadas programáticas à AWS.

Ao criar suas access keys, você cria o access key ID (por exemplo, AKIAIOSFODNN7EXAMPLE) e a secret access key (por exemplo, wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY) como um conjunto. A secret access key está disponível para download apenas quando você a cria. Se você não baixar sua secret access key ou se a perder, deverá criar uma nova.

Você pode atribuir até duas access keys por usuário (root user ou IAM user). Ter duas access keys é útil quando você quer rotacioná-las. Quando você desabilita uma access key, não pode utilizá-la, mas ela conta para o seu limite de duas access keys. Após excluir uma access key, ela desaparece para sempre e não pode ser restaurada, mas pode ser substituída por uma nova access key.

Para gerenciar access keys ao estar logado como root user

1. Faça login no AWS Management Console como usuário root.
2. Na barra de navegação no canto superior direito, escolha o nome ou número da sua conta e então escolha **My Security Credentials**.
3. Expanda a seção **Access keys (access key ID and secret access key)**.
4. Faça uma das seguintes opções:
 - Para criar uma access key, escolha **Create New Access Key**. Se você já tiver duas access keys, este botão fica desabilitado e você deve excluir uma access key antes de poder criar uma nova. Quando solicitado, escolha **Show Access Key** ou **Download Key File**. Esta é a sua única oportunidade de salvar sua secret access key. Depois de salvar sua secret access key em um local seguro, escolha **Close**.
 - Para desativar uma access key, escolha **Make Inactive**. Quando solicitada a confirmação, escolha **Deactivate**. Uma access key desativada ainda conta para o seu limite de duas access keys.
 - Para ativar uma access key, escolha **Make Active**.
 - Para excluir uma access key quando você não precisar mais dela, copie o access key ID e então escolha **Delete**. Antes de poder excluir a access key, você deve escolher **Deactivate**. Recomendamos que você verifique se a access key não está mais em uso antes de excluí-la permanentemente. Para confirmar a exclusão, cole o access key ID no campo de entrada de texto e então escolha **Delete**.

Para gerenciar chaves de acesso quando conectado como um usuário IAM

1. Faça login no AWS Management Console como um usuário IAM.
2. Na barra de navegação no canto superior direito, escolha seu nome de usuário e então escolha **My Security Credentials**.
3. Faça uma das seguintes opções:

- Para criar uma access key, escolha **Create access key**. Se você já tiver duas access keys, este botão está desabilitado e você deve excluir uma access key antes de poder criar uma nova. Quando solicitado, escolha **Show secret access key** ou **Download .csv file**. Esta é a sua única oportunidade de salvar a sua secret access key. Depois de salvar a sua secret access key em um local seguro, escolha **Close**.
- Para desativar uma access key, escolha **Make inactive**. Quando solicitada a confirmação, escolha **Deactivate**. Uma access key desativada ainda conta para o seu limite de duas access keys.
- Para ativar uma access key, escolha **Make active**. Quando solicitada a confirmação, escolha **Make active**.
- Para excluir uma access key quando você não precisar mais dela, copie o access key ID e então escolha **Delete**. Isso desativa a access key. Recomendamos que você verifique se a access key não está mais em uso antes de excluí-la permanentemente. Para confirmar a exclusão, cole o access key ID no campo de entrada de texto e então escolha **Delete**.

Configuração do sgcOpenAPI

Uma vez que você tenha suas próprias AWS Access Keys, você deve configurá-las no OpenAPI Amazon Client antes de fazer qualquer Request aos Servidores Amazon AWS.

```
GetOpenAPIClient->AmazonOptions->AccessKey = "AKIAIOSFODNN7EXAMPLE";
GetOpenAPIClient->AmazonOptions->SecretKey = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY";
```


OpenAPI Amazon AWS | S3

O Amazon Simple Storage Service (Amazon S3) é um serviço de armazenamento de objetos que oferece escalabilidade, disponibilidade de dados, segurança e desempenho líderes do setor. Clientes de todos os tamanhos e setores podem usar o Amazon S3 para armazenar e proteger qualquer quantidade de dados para uma variedade de casos de uso, como data lakes, sites, aplicações móveis, backup e restauração, arquivamento, aplicações empresariais, dispositivos IoT e análise de big data. O Amazon S3 fornece recursos de gerenciamento para que você possa otimizar, organizar e configurar o acesso aos seus dados a fim de atender aos seus requisitos específicos de negócios, organizacionais e de conformidade.

ListBuckets

```
GetOpenAPIClient->AmazonOptions->AccessKey = "AKIAIOSFODNN7EXAMPLE";  
GetOpenAPIClient->AmazonOptions->SecretKey = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY";  
GetOpenAPIClient->ListBuckets();
```

GetObject

```
GetOpenAPIClient->AmazonOptions->AccessKey = "AKIAIOSFODNN7EXAMPLE";  
GetOpenAPIClient->AmazonOptions->SecretKey = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY";  
GetOpenAPIClient->GetObject("bucket_name");
```

PutObject

```
GetOpenAPIClient->AmazonOptions->AccessKey = "AKIAIOSFODNN7EXAMPLE";  
GetOpenAPIClient->AmazonOptions->SecretKey = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY";  
GetOpenAPIClient->PutObject("bucket_name", "MyFile.txt", "payload");
```

OpenAPI | Google Cloud

O **sgcOpenAPI Google Client** (TsgcOpenAPI_Google_Client) tem seu próprio OpenAPI Client, que herda de [TsgcOpenAPI_Client](#).

Este componente tem uma propriedade chamada **GoogleOptions** que inclui todas as configurações necessárias para conectar aos servidores do Google Cloud.

GoogleOptions

O cliente OpenAPI Google permite que você autentique utilizando os seguintes métodos:

1. **OAuth2 Code**: é interativo, o que significa que requer a intervenção do usuário.
2. **JWT (contas de serviço)**: é não interativo, então pode ser executado como um serviço, por exemplo.

A autenticação é configurada na propriedade `GoogleOptions.Authentication`, que permite os seguintes valores:

- **oagaOAuth2**: interativo.
- **oagaJWT**: não interativo. Você pode importar as configurações de um arquivo JSON, utilizando o método **LoadSettingsFromFile**. Este método preencherá as seguintes propriedades automaticamente:
 - **ClientEmail**
 - **PrivateKeyId**
 - **PrivateKey**
 - **ServiceAccountOptions**: ao executar o cliente utilizando uma conta de serviço, as seguintes propriedades são obrigatórias:
 - **TokenURI**: por padrão o valor é "https://oauth2.googleapis.com/token", mas o valor é atualizado ao utilizar o método `LoadSettingsFromFile`.
 - **Subject**: esta é a conta de e-mail ao utilizar a delegação em todo o domínio (Domain-Wide)
 - **Scopes**: uma lista dos escopos.

Most common uses

- **Configuração**
 - [Google Cloud OAuth2](#)
 - [Google Cloud Service Accounts](#)
- **APIs**
 - [Google Cloud PubSub](#)
 - [Google Cloud Calendar](#)
 - [Google Cloud Drive](#)

sgcOpenAPI Google Cloud SDK

Encontre abaixo uma lista das APIs atualmente disponíveis.

- Abusive Experience Report API
- API de URL Accelerated Mobile Pages (AMP)
- Access Approval API
- Access Context Manager API
- Ad Exchange Buyer API
- Ad Exchange Buyer API II
- Ad Experience Report API
- Admin SDK API
- AdMob API
- AdSense Host API
- AdSense Management API
- AI Platform Training & Prediction API

- API Analytics Reporting
- Android Device Provisioning Partner API
- Android Management API
- API Discovery Service
- API Gateway API
- API Keys API
- Apigee API
- App Engine Admin API
- Apps Script API
- Area120 Tables API
- Artifact Registry API
- Assured Workloads API
- Authorized Buyers Marketplace API
- Backup for GKE API
- Bare Metal Solution API
- BigQuery API
- BigQuery Connection API
- BigQuery Data Transfer API
- BigQuery Reservation API
- Binary Authorization API
- Blogger API v3
- Books API
- Calendar API
- Campaign Manager 360 API
- Certificate Authority API
- Certificate Manager API
- Chrome Management API
- Chrome Policy API
- Chrome UX Report API
- Chrome Verified Access API
- Cloud Asset API
- Cloud AutoML API
- API Cloud Bigtable Admin
- Cloud Billing API
- Cloud Billing Budget API
- Cloud Build API
- Cloud Channel API
- API Cloud Composer
- Cloud Data Fusion API
- Cloud Data Loss Prevention (DLP) API
- Cloud Dataplex API
- Cloud Dataproc API
- Cloud Datastore API
- Cloud Debugger API
- Cloud Deployment Manager V2 API
- Cloud DNS API
- Cloud Document AI API
- Cloud Domains API
- Cloud Filestore API
- Cloud Firestore API
- Cloud Functions API
- Cloud Healthcare API
- Cloud Identity API
- Cloud Identity-Aware Proxy API
- Cloud IDS API
- Cloud IoT API
- Cloud Key Management Service (KMS) API
- Cloud Life Sciences API
- Cloud Logging API
- Cloud Memorystore for Memcached API
- Cloud Monitoring API
- Cloud Natural Language API
- API Cloud OS Login
- Cloud Private Catalog
- Cloud Private Catalog Producer

- Cloud Pub/Sub API
- Cloud Resource Manager API
- Cloud Run Admin API
- Cloud Runtime Configuration API
- Cloud Scheduler API
- Cloud Search API
- Cloud Shell API
- Cloud Source Repositories API
- Cloud Spanner API
- Cloud Speech-to-Text API
- Cloud SQL Admin API
- Cloud Storage for Firebase API
- Cloud Storage JSON API
- Cloud Talent Solution API
- Cloud Tasks API
- Cloud Testing API
- API Cloud Text-to-Speech
- Cloud Tool Results API
- Cloud TPU API
- Cloud Trace API
- Cloud Translation API
- Cloud Video Intelligence API
- Cloud Vision API
- Compute Engine API
- Connectors API
- Contact Center AI Insights API
- Container Analysis API
- Content API for Shopping
- Custom Search API
- Data Labeling API
- Data pipelines API
- Database Migration API
- Dataflow API
- Dataproc Metastore API
- Datastream API
- API Dialogflow
- Digital Asset Links API
- Display & Video 360 API
- Domains RDAP API
- DoubleClick Bid Manager API
- Drive Activity API
- Drive API
- Enterprise License Manager API
- Error Reporting API
- Essential Contacts API
- Eventarc API
- Fact Check Tools API
- Firebase App Check API
- Firebase Cloud Messaging API
- Firebase Cloud Messaging Data API
- Firebase Dynamic Links API
- Firebase Hosting API
- Firebase Management API
- Firebase ML API
- Firebase Realtime Database Management API
- Firebase Rules API
- Fitness API
- Game Services API
- Genomics API
- GKE Hub API
- Gmail API
- Gmail Postmaster Tools API
- Google Analytics Admin API
- Google Analytics API
- Google Analytics Data API

- Google Chat API
- Google Civic Information API
- Google Classroom API
- Google Cloud Data Catalog API
- Google Cloud Deploy API
- Google Cloud Memorystore for Redis API
- Google Cloud Support API
- Google Docs API
- Google Forms API
- API Google Identity Toolkit
- Google Keep API
- Google Mirror
- Google My Business API
- Google OAuth2 API
- Google Pay Passes API
- Google Play Android Developer API
- Google Play Custom App Publishing API
- Google Play Developer Reporting API
- Google Play EMM API
- Google Play Game Management
- Google Play Game Services
- Google Play Game Services Publishing API
- Google Play Integrity API
- Google Search Console API
- Google Sheets API
- Google Site Verification API
- Google Slides API
- Google Vault API
- Google Workspace Alert Center API
- Google Workspace Reseller API
- Google+ API
- Groups Migration API
- Groups Settings API
- HomeGraph API
- IAM Service Account Credentials API
- API Idea Hub
- Identity and Access Management (IAM) API
- Indexing API
- Knowledge Graph Search API
- Kubernetes Engine API
- Library Agent API
- Local Services API
- API Managed Service for Microsoft Active Directory
- Manufacturer Center API
- My Business Account Management API
- API My Business Business Calls
- API My Business Business Information
- My Business Lodging API
- My Business Notifications API
- My Business Place Actions API
- My Business Q&A API
- My Business Verifications API
- Network Connectivity API
- Network Management API
- Network Security API
- API Network Services
- Notebooks API
- On-Demand Scanning API
- Organization Policy API
- OS Config API
- PageSpeed Insights API
- Payments Reseller Subscription API
- People API
- Perspective Comment Analyzer API
- Playable Locations API

- Policy Analyzer API
- Policy Simulator API
- Policy Troubleshooter API
- Poly API
- Proximity Beacon API
- Pub/Sub Lite API
- Real-time Bidding API
- reCAPTCHA Enterprise API
- Recommendations AI (Beta)
- Recommender API
- API Remote Build Execution
- Replica Pool
- Resource Settings API
- Retail API
- API Safe Browsing
- SAS Portal API
- SAS Portal API (Testing)
- Search Ads 360 API
- Search Console API
- Secret Manager API
- Security Command Center API
- Security Token Service API
- Semantic Tile API
- Service Broker
- Service Consumer Management API
- Service Control API
- Service Directory API
- Service Management API
- Service Networking API
- Service Usage API
- Smart Device Management API
- Stackdriver Profiler API
- Storage Transfer API
- Street View Publish API
- Tag Manager API
- Tasks API
- Traffic Director API
- Transcoder API
- Version History API
- VM Migration API
- Web Fonts Developer API
- Web Risk API
- Web Security Scanner API
- Workflow Executions API
- API Workflows
- API YouTube Analytics
- YouTube Data API v3
- YouTube Reporting API

OpenAPI Google Cloud | OAuth2

Para utilizar os componentes OpenAPI Google Cloud e Autenticar utilizando OAuth2, primeiro você deve obter a OAuth2 Key do Google Cloud.

Encontre abaixo os passos para obter as chaves OAuth2 do Google e como configurá-las em nossa aplicação de exemplo PubSub.

Primeiro **faça login** na sua **Google Cloud Account** e utilize um projeto existente ou crie um novo.

Depois disso, vá ao menu **Credentials** e pressione o botão **CREATE CREDENTIALS**, selecione a opção **OAuth Client ID**.

Selecione o tipo de sua aplicação e defina um nome de descrição

Se bem-sucedido, você obterá seu **Client Id** e **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-ehcskpfsp4180r1bdfoe6mc32e3ncmn0.apps.gc



Your Client Secret

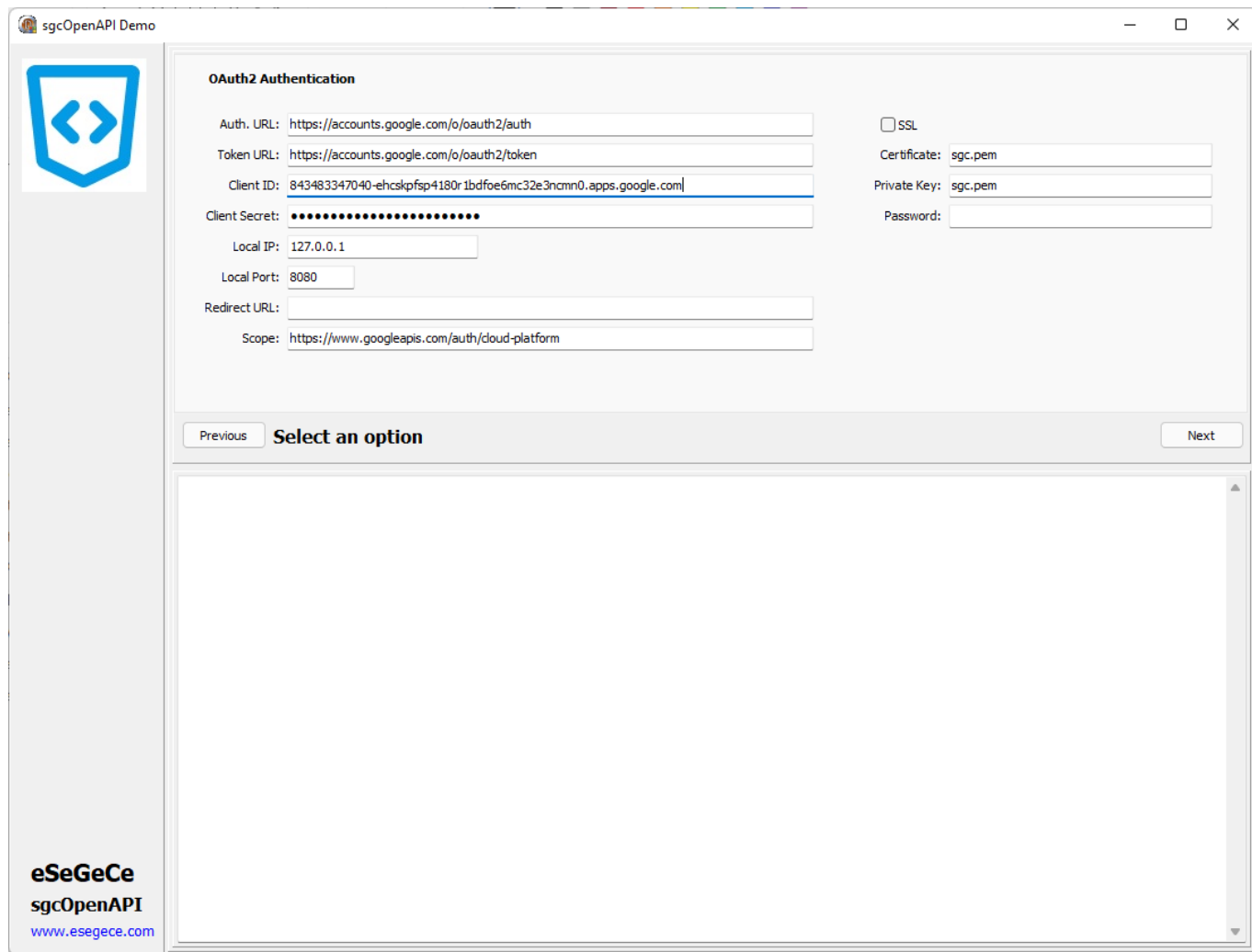
pvogD9reE0t9i1L6eR1jE60Z



OK

Não compartilhe seus dados OAuth2 com ninguém!

Agora copie para o sample do OpenAPI Google Cloud



The image shows a screenshot of the 'sgcOpenAPI Demo' application window. The window has a title bar with standard Windows controls (minimize, maximize, close). On the left side, there is a vertical sidebar containing a blue shield logo with a white 'X' and the text 'eSeGeCe sgcOpenAPI www.esegece.com' at the bottom. The main area of the window is titled 'OAuth2 Authentication'. It contains several input fields for configuring OAuth2 settings. The fields are arranged in two columns. The left column includes: 'Auth. URL' (https://accounts.google.com/o/oauth2/auth), 'Token URL' (https://accounts.google.com/o/oauth2/token), 'Client ID' (843483347040-ehcspkfp4180r1bdf0e6mc32e3ncmn0.apps.google.com), 'Client Secret' (masked with dots), 'Local IP' (127.0.0.1), 'Local Port' (8080), 'Redirect URL' (empty), and 'Scope' (https://www.googleapis.com/auth/cloud-platform). The right column includes: an unchecked 'SSL' checkbox, 'Certificate' (sgc.pem), 'Private Key' (sgc.pem), and 'Password' (empty). Below the input fields, there is a 'Previous' button, a 'Select an option' label, and a 'Next' button. The bottom of the window is a large empty area with a scrollbar on the right.

sgcOpenAPI Demo

OAuth2 Authentication

Auth. URL:

Token URL:

Client ID:

Client Secret:

Local IP:

Local Port:

Redirect URL:

Scope:

☐ SSL

Certificate:

Private Key:

Password:

Select an option

eSeGeCe
sgcOpenAPI
www.esegece.com

Leia mais sobre a [Configuração OAuth2](#).

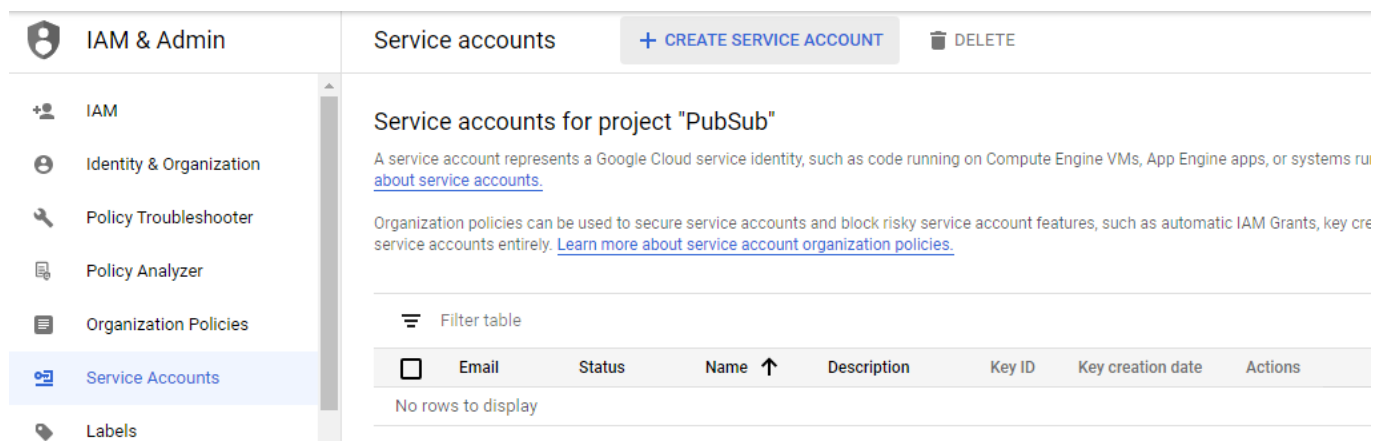
Uma vez autenticado, você pode **reautenticar** chamando primeiro o método **ClearOAuth2Token** (limpa todos os Tokens OAuth2 internos) e depois chamando qualquer requisição OpenAPI; um novo navegador web será exibido para reautenticar nos servidores do google.

OpenAPI Google Cloud | Service Accounts

Para usar os **componentes OpenAPI Google Cloud e Autenticar usando Service Accounts**, primeiro você deve obter o Certificado de Chave Privada do Google Cloud.

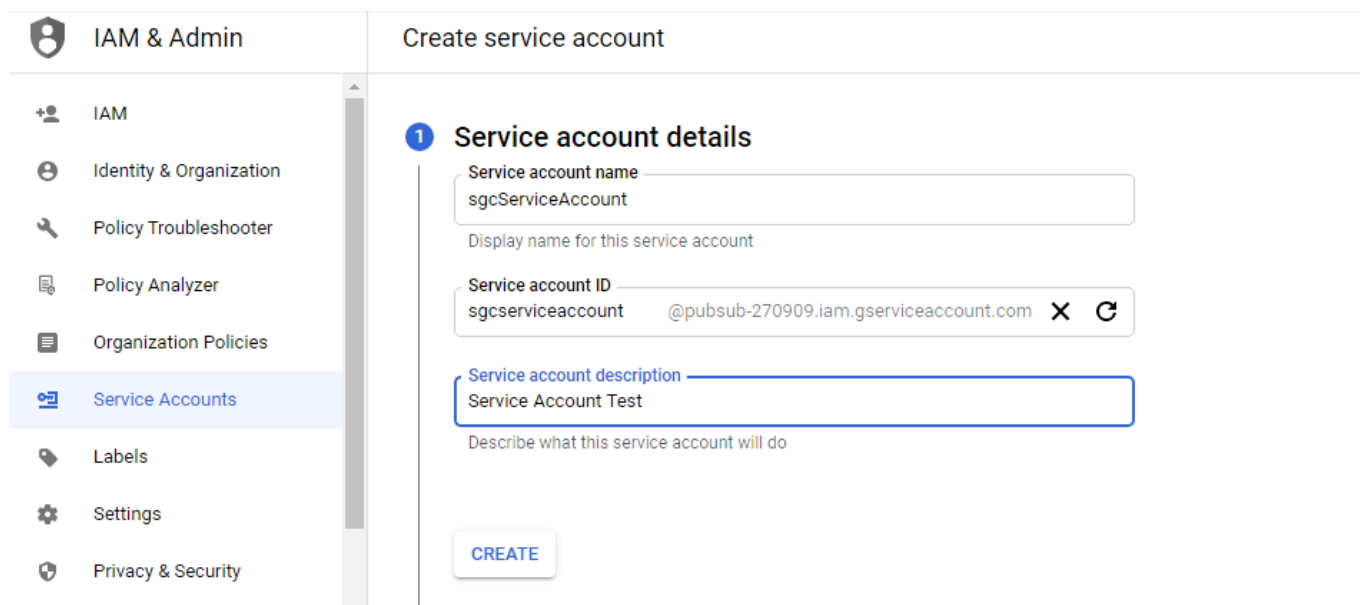
Encontre abaixo os passos para obter o Google Private Key Certificate e como configurá-lo em nossa aplicação de exemplo.

Primeiro **faça login** na sua **Google Cloud Account** e utilize um projeto existente ou crie um novo.



The screenshot shows the Google Cloud IAM & Admin console. On the left sidebar, 'Service Accounts' is selected under the 'IAM & Admin' section. The main content area is titled 'Service accounts for project "PubSub"'. It includes a '+ CREATE SERVICE ACCOUNT' button and a 'DELETE' button. Below this, there is explanatory text about service accounts and a link to 'Learn more about service account organization policies'. A table with columns 'Email', 'Status', 'Name', 'Description', 'Key ID', 'Key creation date', and 'Actions' is shown, but it contains 'No rows to display'.

Selecione **CREATE SERVICE ACCOUNT** e uma nova página será exibida, onde você deve definir o nome e a descrição da conta de serviço



The screenshot shows the 'Create service account' page in the Google Cloud IAM & Admin console. The left sidebar is the same as the previous screenshot. The main content area is titled 'Create service account' and has a step indicator '1 Service account details'. It contains three input fields: 'Service account name' with the value 'sgcServiceAccount', 'Service account ID' with the value 'sgcserviceaccount' and a domain '@pubsub-270909.iam.gserviceaccount.com', and 'Service account description' with the value 'Service Account Test'. A 'CREATE' button is at the bottom.

Em seguida, selecione ao menos um Role; eu seleciono PubSub Admin para permitir que o cliente publique e assine tópicos, mas você pode selecionar outro role com menos privilégios

IAM & Admin

- IAM
- Identity & Organization
- Policy Troubleshooter
- Policy Analyzer
- Organization Policies
- Service Accounts**
- Labels
- Settings
- Privacy & Security
- Identity-Aware Proxy

Create service account

✓ **Service account details**

2 **Grant this service account access to project (optional)**

Grant this service account access to PubSub so that it has permission to complete specific actions on the resources in your project. [Learn more](#)

Role: **Pub/Sub Admin** Condition: [Add condition](#)

Full access to topics, subscriptions, and snapshots.

[+ ADD ANOTHER ROLE](#)

CONTINUE

Pressione CONTINUE e, por fim, você pode conceder acesso a outros usuários

IAM & Admin

- IAM
- Identity & Organization
- Policy Troubleshooter
- Policy Analyzer
- Organization Policies
- Service Accounts**
- Labels
- Settings
- Privacy & Security
- Identity-Aware Proxy
- Roles

Create service account

✓ **Service account details**

✓ **Grant this service account access to project (optional)**

3 **Grant users access to this service account (optional)**

Grant access to users or groups that need to perform actions as this service account. [Learn more](#)

Service account users role ?
Grant users the permissions to deploy jobs and VMs with this service account

Service account admins role ?
Grant users the permission to administer this service account

DONE CANCEL

Pressione DONE quando terminar e um novo registro será exibido

IAM & Admin

IAM
Identity & Organization
Policy Troubleshooter
Policy Analyzer
Organization Policies
Service Accounts
Labels

Service accounts

+ CREATE SERVICE ACCOUNT

DELETE

Service accounts for project "PubSub"

A service account represents a Google Cloud service identity, such as code running on Compute Engine VMs, App Engine apps, or system [about service accounts](#).

Organization policies can be used to secure service accounts and block risky service account features, such as automatic IAM Grants, ke service accounts entirely. [Learn more about service account organization policies](#).

Filter table

☐	Email	Status	Name ↑	Description	Key ID
☐	 sgcserviceaccount@pubsub-270909.iam.gserviceaccount.com	✓	sgcServiceAccount	Service Account Test	No keys

O próximo passo é criar uma nova Key; portanto, selecione a opção Create Key na coluna de ações. Selecione JSON para baixar a configuração em formato JSON e uma nova Key será criada

Service accounts for project "PubSub"

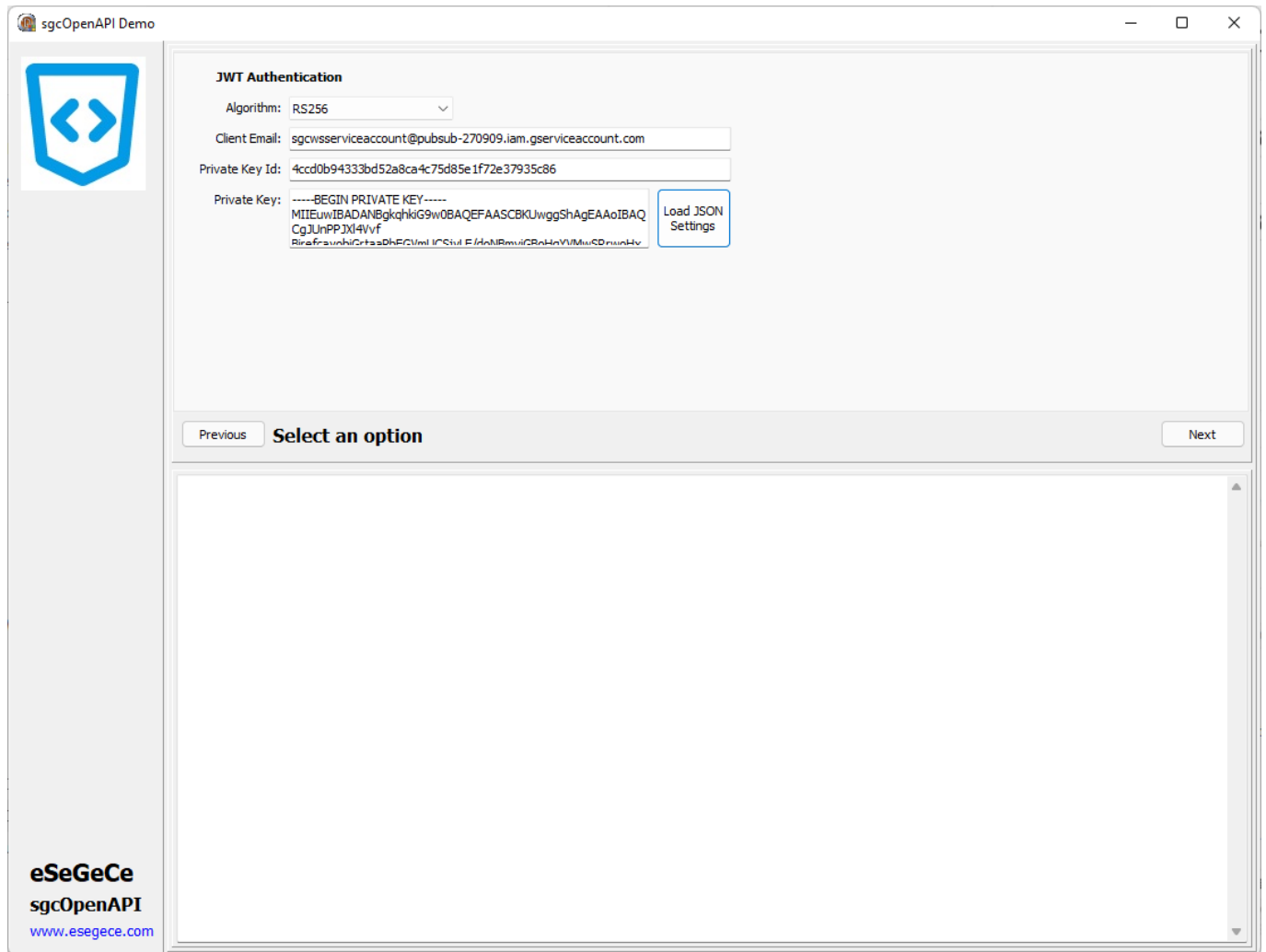
A service account represents a Google Cloud service identity, such as code running on Compute Engine VMs, App Engine apps, or systems running outside Google. [Learn more about service accounts](#).

Organization policies can be used to secure service accounts and block risky service account features, such as automatic IAM Grants, key creation/upload, or the creation of service accounts entirely. [Learn more about service account organization policies](#).

Filter table

☐	Email	Status	Name ↑	Description	Key ID	Key creation date	Actions
☐	 sgcserviceaccount@pubsub-270909.iam.gserviceaccount.com	✓	sgcServiceAccount	Service Account Test	425a876f68b8b2a66f2fcc5b2dee8e918b0eb9ab	Dec 20, 2020	⋮

Por fim, você só precisa preencher os dados fornecidos pelo google no cliente OpenAPI PubSub. Você pode usar **LoadSettingsFromFile** para carregar o arquivo de configuração JSON.



The screenshot shows a web application window titled "sgcOpenAPI Demo". On the left is a sidebar with a blue shield logo containing a white code symbol and the text "eSeGeCe sgcOpenAPI www.esegece.com". The main area is titled "JWT Authentication" and contains the following fields:

- Algorithm:** A dropdown menu showing "RS256".
- Client Email:** A text input field containing "sgcwserviceaccount@pubsub-270909.iam.gserviceaccount.com".
- Private Key Id:** A text input field containing "4ccd0b9433bd52a8ca4c75d85e1f72e37935c86".
- Private Key:** A text area containing a long string of characters, starting with "-----BEGIN PRIVATE KEY-----". To the right of this field is a button labeled "Load JSON Settings".

At the bottom of the form, there are three buttons: "Previous", "Select an option", and "Next". Below these buttons is a large, empty text area.

Domain-Wide Delegation

Se você tiver uma conta Google Workspace, um administrador da organização pode autorizar uma aplicação a acessar dados do usuário em nome dos usuários no domínio Google Workspace. Por exemplo, uma aplicação que usa a **Google Calendar API** para adicionar eventos aos calendários de todos os usuários em um domínio Google Workspace usaria uma service account para acessar a Google Calendar API em nome dos usuários. Autorizar uma service account a acessar dados em nome dos usuários em um domínio é às vezes referido como "delegar autoridade em todo o domínio" a uma service account.

Para delegar autoridade em todo o domínio a uma conta de serviço, um superadministrador do domínio do Google Workspace deve concluir os seguintes passos:

- No Admin console do seu domínio Google Workspace, vá em **Main menu menu > Security > Access e data control > API Controls**.
- No **painel Domain wide delegation**, selecione **Manage Domain Wide Delegation**.
- Clique em **Add new**.
- No campo **Client ID**, insira o **Client ID** da conta de serviço. Você pode encontrar o client ID da sua conta de serviço na página Service accounts.
- No campo **OAuth scopes (comma-delimited)**, insira a lista de escopos aos quais sua aplicação deve receber acesso. Por exemplo, se sua aplicação precisar de acesso total domain-wide à API do Google Drive e à API do Google Calendar, insira: <https://www.googleapis.com/auth/drive>, <https://www.googleapis.com/auth/calendar>.
- Clique em **Authorize**.

Depois de vincular e autorizar a conta do workspace, configure a propriedade `GoogleOptions.ServiceAccountOptions` a partir do cliente OpenAPI:

- **Subject:** é a conta de e-mail do workspace vinculada à conta de serviço. Exemplo: `youremail@domain.com`

- **Scopes:** lista de escopos. Exemplo: <https://www.googleapis.com/auth/calendar>
- **TokenURI:** por padrão é <https://oauth2.googleapis.com/token>.

OpenAPI Google Cloud | PubSub

O Pub/Sub permite que os serviços se comuniquem de forma assíncrona, com latências na ordem de 100 milissegundos.

O Pub/Sub é utilizado para pipelines de analytics em streaming e integração de dados para ingerir e distribuir dados. É igualmente eficaz como um middleware orientado a mensagens para integração de serviços ou como uma fila para paralelizar tarefas.

Listar projetos por tópico (OAuth2)

```
GetOpenAPIClient->GoogleOptions->Authentication = oagaOAuth2;
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->AuthURL = "https://accounts.google.com/o/oauth2";
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->TokenURL = "https://accounts.google.com/o/oauth2/token";
GetOpenAPIClient->Authentication->OAuth2->OAuth2Options->ClientId = "google_client_id";
GetOpenAPIClient->Authentication->OAuth2->OAuth2Options->ClientSecret = "google_client_secret";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->IP = "127.0.0.1";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->Port = 8080;
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->Scope->Text = "https://www.googleapis.com/auth/pubsub";
GetOpenAPIClient->ListV1TopicsByProject("projects/pubsub-270909");
```

Listar Projetos por Tópico (Service Accounts)

```
GetOpenAPIClient->GoogleOptions->Authentication = oagaJWT;
GetOpenAPIClient->LoadSettingsFromFile("google.json");
GetOpenAPIClient->ListV1TopicsByProject("projects/pubsub-270909");
```

OpenAPI Google Cloud Calendar

A Calendar API permite que você integre seu app ao Google Calendar, criando novas formas de engajar seus usuários.

List Events By Calendar (OAuth2)

```
GetOpenAPIClient->GoogleOptions->Authentication = oagaOAuth2;
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->AuthURL = "https://accounts.google.com/o/oauth2";
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->TokenURL = "https://accounts.google.com/o/oauth2/token";
GetOpenAPIClient->Authentication->OAuth2->OAuth2Options->ClientId = "google_client_id";
GetOpenAPIClient->Authentication->OAuth2->OAuth2Options->ClientSecret = "google_client_secret";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->IP = "127.0.0.1";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->Port = 8080;
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->Scope->Text = "https://www.googleapis.com/calendar.events";
GetOpenAPIClient->ListCalendarsEventsByCalendarId("email@mydomain.com", true, "json", 0, "");
```

List Events By Calendar (Service Account)

```
GetOpenAPIClient->GoogleOptions->Authentication = oagaJWT;
GetOpenAPIClient->LoadSettingsFromFile("google.json");
GetOpenAPIClient->Authentication->ServiceAccountOptions->Subject = "email@mydomain.com";
GetOpenAPIClient->Authentication->ServiceAccountOptions->Scopes->Text := "https://www.googleapis.com/auth/calendar";
GetOpenAPIClient->ListCalendarsEventsByCalendarId("email@mydomain.com", true, "json", 0, "");
```

Inserir Evento de Calendário

```
GetOpenAPIClient->InsertCalendarsEventsByCalendarId("email@mydomain.com", "{ \"summary\": \"Test Event\", \"description\": \"\" }", true, "json", 0, "");
```


OpenAPI Google Cloud | Drive

Utilize a unit `sgcOpenAPI_googleapis.com_drive_helpers` se você quiser fazer upload e/ou download de arquivos da sua conta do Google Drive.

Upload Files

Use o método `UploadFile` para fazer o upload de um novo arquivo para a sua conta do google drive. O Evento `OnUpload` permite que você conheça o progresso do upload.

```
void __fastcall UploadFileToGoogleDrive()
{
    TOpenDialog *oDialog = new TOpenDialog(NULL);
    try
    {
        if (oDialog->Execute())
        {
            GetOpenAPIClient()->OnUpload = OnUploadEvent;
            ShowMessage(GetOpenAPIClient()->UploadFile(oDialog->FileName));
        }
    }
    __finally
    {
        delete oDialog;
    }
}

void __fastcall OnUploadEvent(TObject *Sender, TsgcOpenAPIProgressState aState, __int64 aValue)
{
    switch (aState)
    {
        case oahpStart:
            ProgressBar1->Visible = true;
            ProgressBar1->Position = 0;
            ProgressBar1->Max = aValue;
            break;
        case oahpProgress:
            ProgressBar1->Visible = true;
            ProgressBar1->Position = aValue;
            break;
        case oahpEnd:
            ProgressBar1->Visible = false;
            ProgressBar1->Position = 0;
            ProgressBar1->Max = 0;
            break;
    }
    Application->ProcessMessages();
}
```

Download Files

Utilize o método `DownloadFile` para baixar arquivos da sua conta do Google Drive. O evento `OnDownload` permite que você acompanhe o progresso do download.

```
void __fastcall DownloadFileFromGoogleDrive()
{
    TSaveDialog *oSaveDialog = new TSaveDialog(NULL);
    try
    {
        if (oSaveDialog->Execute())
        {
            GetOpenAPIClient()->OnDownload = OnDownloadEvent;
            GetOpenAPIClient()->DownloadFile(txtDownloadFileId->Text, oSaveDialog->FileName);
        }
    }
    __finally
    {
        delete oSaveDialog;
    }
}
```

```

}
void __fastcall OnDownloadEvent(TObject *Sender, TsgcOpenAPIProgressState aState, __int64 aValue)
{
    switch (aState)
    {
        case oahpStart:
            ProgressBar1->Visible = true;
            ProgressBar1->Position = 0;
            ProgressBar1->Max = aValue;
            break;
        case oahpProgress:
            ProgressBar1->Visible = true;
            ProgressBar1->Position = aValue;
            break;
        case oahpEnd:
            ProgressBar1->Visible = false;
            ProgressBar1->Position = 0;
            ProgressBar1->Max = 0;
            break;
    }
    Application->ProcessMessages();
}

```

List Folders

Lista todas as pastas na conta do drive. Utilize o método **ListFiles** e passe o parâmetro q como "application/vnd.google-apps.folder".

```

GetOpenAPIClient().ListFiles("", true, "", false, false, 100, "", false, false, "", "", "", "", "", "", "", "", "", ""

```

OpenAPI | Microsoft

O **sgcOpenAPI Microsoft Client** (TsgcOpenAPI_Microsoft_Client) tem seu próprio Cliente OpenAPI que herda de [TsgcOpenAPI_Client](#).

Este componente possui uma propriedade chamada **MicrosoftOptions** que inclui todas as configurações necessárias para conectar aos Servidores Microsoft.

MicrosoftOptions

O cliente OpenAPI Microsoft permite que você autentique utilizando os seguintes métodos:

1. **OAuth2 Code**: é interativo, o que significa que requer a intervenção do usuário.
2. **OAuth2 Credentials**: não é interativo, portanto pode rodar como um serviço, por exemplo.

A autenticação é configurada na propriedade `MicrosoftOptions.Authentication` e permite os seguintes valores:

- **oamaOAuth2Code**: interativo.
- **oamaOAuth2Credentials**: não interativo.

Outras propriedades exigidas pela Microsoft são as seguintes:

- **TenantId**: é um valor que identifica sua conta, você pode encontrá-lo em sua conta Microsoft/Azure.

Most common uses

- **Configuration**
 - [Microsoft Get Tenant](#)
 - [Microsoft Register Application](#)
 - [Microsoft OAuth2 Code](#)
 - [Microsoft OAuth2 Credentials](#)
- **APIs**
 - [Microsoft Graph](#)

sgcOpenAPI Microsoft APIs

Encontre abaixo uma lista das APIs atualmente disponíveis.

- AutoSuggest Client
- Computer Vision Client
- Custom Image Search Client
- Cliente Custom Search
- Custom Vision Prediction Client
- Custom Vision Training Client
- Entity Search Client
- Image Search Client
- Local Search Client
- News Search Client
- Partial Graph API
- Spell Check Client
- Video Search Client
- Visual Search Client
- Web Search Client

sgcOpenAPI Azure APIs

Encontre abaixo uma lista das APIs atualmente disponíveis.

- API Client
- ACE Provisioning ManagementPartner
- ADHybridHealthService
- AdvisorManagementClient
- Cliente Anomaly Detector
- Cliente Localizador de Anomalias
- ApiManagementClient
- AppConfigurationManagementClient
- Application Insights Data Plane
- ApplicationClient
- ApplicationInsightsManagementClient
- AppPlatformManagementClient
- AppServiceCertificateOrders API Client
- AppServiceEnvironments API Client
- AppServicePlans API Client
- Artifact
- AttestationClient
- AuthorizationManagementClient
- AutomationManagement
- AutomationManagementClient
- Azure Action Groups
- Azure Activity Log Alerts
- Azure Addons Resource Provider
- Azure Alerts Management Service Resource Provider
- Azure Bot Service
- Azure CDN WebApplicationFirewallManagement
- Azure Data Catalog Resource Provider
- Azure Data Lake Storage
- Azure Data Migration Service Resource Provider
- Azure Dedicated HSM Resource Provider
- Azure DevOps
- Azure Enterprise Knowledge Graph Service
- Azure IoT Central
- Azure Location Based Services Resource Provider
- Azure Log Analytics
- Azure Log Analytics - Operations Management
- Azure Log Analytics Query Packs
- Azure Machine Learning Datastore Management Client
- Azure Machine Learning Model Management Service
- Azure Machine Learning Workspaces
- Azure Maps Resource Provider
- Azure Media Services
- Azure Metrics
- Azure Migrate Hub
- Azure Migrate V2
- Cliente do Azure ML Commitment Plans Management
- Azure ML Web Services Management Client
- Azure Monitor Private Link Scopes
- Azure Reservation
- Azure Resource Graph
- Azure Resource Graph Query
- Azure SQL Database
- Azure SQL Database API spec
- Azure SQL Database Backup
- Azure SQL Database Backup Long Term Retention Policy
- Azure SQL Database Datamasking Policies and Rules
- Azure SQL Database disaster recovery configurations
- Azure SQL Database Import/Export spec
- Links de replicação do Azure SQL Database
- Azure SQL Server API spec
- Azure SQL Server Backup Long Term Retention Vault

- Azure Stack Azure Bridge Client
- azureactivedirectory
- AzureAnalysisServices
- AzureBridgeAdminClient
- AzureDataManagementClient
- AzureDeploymentManager
- AzureDigitalTwinsManagementClient
- AzureStack Azure Bridge Client
- BackupManagementClient
- BatchAI
- BatchManagement
- BatchService
- BillingManagementClient
- BlockchainManagementClient
- BlueprintClient
- CdnManagementClient
- CertificateRegistrationProvider API Client
- Certificates API Client
- CognitiveServicesManagementClient
- CommerceManagementClient
- Compute Admin Client
- ComputeDiskAdminManagementClient
- ComputeManagementClient
- ComputeManagementConvenienceClient
- Computer Vision
- ConsumptionManagementClient
- ContainerInstanceManagementClient
- ContainerRegistryManagementClient
- ContainerServiceClient
- Content Moderator Client
- Cosmos DB
- CostManagementClient
- Customer Lockbox
- CustomerInsightsManagementClient
- customproviders
- APIs de Database Threat Detection Policy
- DataBoxEdgeManagementClient
- DataBoxManagementClient
- DatabricksClient
- DataFactoryManagementClient
- DataLakeAnalyticsAccountManagementClient
- DataLakeAnalyticsCatalogManagementClient
- DataLakeAnalyticsJobManagementClient
- DataLakeStoreAccountManagementClient
- DataLakeStoreFileSystemManagementClient
- DataShareManagementClient
- DeletedWebApps API Client
- DeploymentAdminClient
- DeploymentScriptsClient
- DeviceServices
- DevSpacesManagement
- DevTestLabsClient
- Diagnostics API Client
- DiskResourceProviderClient
- DnsManagementClient
- Domain Services Resource Provider
- DomainRegistrationProvider API Client
- Cliente da Domains API
- Dynamics Telemetry
- Engagement.ManagementClient
- EngagementFabric
- EventGridManagementClient
- EventHub2018PreviewManagementClient
- EventHubManagementClient
- Execution Service

- ExpressRouteCrossConnection REST APIs
- FabricAdminClient
- Face Client
- FeatureClient
- Form Recognizer Client
- FrontDoorManagementClient
- GalleryManagementClient
- Guest Diagnostic Settings
- Guest Diagnostic Settings Association
- GuestConfiguration
- HanaManagementClient
- HDInsightJobManagementClient
- HDInsightManagementClient
- HealthcareApisClient
- HybridComputeManagementClient
- HybridDataManagementClient
- HyperDrive
- InfrastructureInsightsManagementClient
- Cliente Ink Recognizer
- InstanceMetadataClient
- IntuneResourceManagementClient
- iotDpsClient
- iotHubClient
- IoTSpacesClient
- KeyVaultClient
- KeyVaultManagementClient
- KustoManagementClient
- LogicAppsManagementClient
- LogicManagementClient
- LUIS Authoring Client
- LUIS Programmatic
- Machine Learning Compute Management Client
- Machine Learning Workspaces Management Client
- MaintenanceManagementClient
- ManagedLabsClient
- ManagedNetworkManagementClient
- ManagedServiceIdentityClient
- ManagedServicesClient
- Management Groups
- ManagementLinkClient
- ManagementLockClient
- MariaDBManagementClient
- Marketplace RP Service
- MarketplaceOrdering.Agreements
- MediaServicesManagementClient
- Microsoft Insights
- Microsoft NetApp
- Microsoft Storage Sync
- Microsoft.ResourceHealth
- Microsoft.Support
- MicrosoftSerialConsoleClient
- Mixed Reality
- ML Team Account Management Client
- MonitorManagementClient
- MySQLManagementClient
- NetworkAdminManagementClient
- NetworkExperiments
- NetworkManagementClient
- NotificationHubsManagementClient
- PeeringManagementClient
- Personalizer Client
- PolicyClient
- PolicyEventsClient
- PolicyMetadataClient
- PolicyStatesClient

- PolicyTrackedResourcesClient
- portal
- PostgreSQLManagementClient
- Power BI Embedded Management Client
- PowerBIDedicated
- PrivateDnsManagementClient
- Cliente da API do Provedor
- QnAMaker Client
- QnAMaker Runtime Client
- Cliente Recommendations API
- RecoveryServicesBackupClient
- RecoveryServicesClient
- RedisManagementClient
- Relay
- RemediationsClient
- ResourceHealthMetadata API Client
- ResourceManagementClient
- Run History APIs
- RunCommandsClient
- SchedulerManagementClient
- SeaBreezeManagementClient
- SearchIndexClient
- SearchManagementClient
- SearchServiceClient
- Security Center
- Security Insights
- ServerManagement
- Service Fabric Client APIs
- Mapa de Serviços
- ServiceBusManagementClient
- ServiceFabricManagementClient
- SharedImageGalleryServiceClient
- SignalRManagementClient
- SiteRecoveryManagementClient
- Software Plan RP
- SqlManagementClient
- SqlVirtualMachineManagementClient
- Cliente de gerenciamento de cache de armazenamento
- StorageImportExport
- StorageManagementClient
- StorSimple8000SeriesManagementClient
- StorSimpleManagementClient
- StreamAnalyticsManagementClient
- SubscriptionClient
- SubscriptionDefinitionsClient
- SubscriptionsManagementClient
- Text Analytics Client
- TimeSeriesInsightsClient
- TopLevelDomains API Client
- TrafficManagerManagementClient
- Update Management
- UpdateAdminClient
- UsageManagementClient
- VirtualMachineImageTemplate
- VirtualWANAsAServiceManagementClient
- Visual Studio Projects Resource Provider Client
- Visual Studio Resource Provider Client
- VM Insights Onboarding
- VMwareCloudSimple
- WebApplicationFirewallManagement
- WebApps API Client
- WebSite Management Client
- windowsesu
- WorkbookClient
- Workload Monitor

OpenAPI Microsoft | Tenant

Para construir apps que utilizam a Microsoft identity platform para gerenciamento de identidade e acesso, você precisa de acesso a um *tenant* do Azure Active Directory (Azure AD). É no tenant do Azure AD que você registra e gerencia seus apps, configura o acesso deles a dados no Microsoft 365 e outras APIs web, e habilita recursos como o Conditional Access.

Um tenant representa uma organização. É uma instância dedicada do Azure AD que uma organização ou desenvolvedor de app recebe no início de um relacionamento com a Microsoft. Esse relacionamento pode começar com a inscrição no Azure, Microsoft Intune ou Microsoft 365, por exemplo.

Cada tenant do Azure AD é distinto e separado de outros tenants do Azure AD. Ele tem sua própria representação de identidades de trabalho e escolares, identidades de consumidor (se for um tenant Azure AD B2C) e registros de aplicação. Um registro de aplicação dentro do seu tenant pode permitir autenticações apenas de contas dentro do seu tenant ou de todos os tenants.

Utilizar um tenant Azure AD existente

Para verificar o tenant:

1. Faça login no Azure Portal. Utilize a conta que você usará para gerenciar sua aplicação.
2. Verifique o canto superior direito. Se você tiver um tenant, será conectado automaticamente. Você verá o nome do tenant diretamente abaixo do nome da sua conta.

Se você não tiver um tenant associado à sua conta, verá um GUID sob o nome da sua conta. Você não poderá realizar ações como registrar apps até criar um tenant do Azure AD.

Criar um novo tenant do Azure AD

Você fornecerá as seguintes informações para criar seu novo tenant:

- **Organization name**
- **Initial domain** - O domínio inicial <domainname>.onmicrosoft.com não pode ser editado ou excluído. Você pode adicionar um nome de domínio personalizado posteriormente.
- **País ou região**

OpenAPI Microsoft | Register Application

Registrar sua aplicação estabelece uma relação de confiança entre seu app e a plataforma de identidade da Microsoft. A confiança é unidirecional: seu app confia na plataforma de identidade da Microsoft, e não o contrário.

Siga estes passos para criar o registro do app:

1. Faça login no Azure Portal.
2. Se você tem acesso a múltiplos tenants, use o filtro **Directories + subscriptions** no menu superior para alternar para o tenant no qual deseja registrar a aplicação.
3. Pesquise e selecione **Azure Active Directory**.
4. Em **Manage**, selecione **App registrations** > **New registration**.
5. Insira um **Name** de exibição para sua aplicação. Os usuários da sua aplicação podem ver o nome de exibição quando usam o app, por exemplo durante o sign-in. Você pode alterar o nome de exibição a qualquer momento e múltiplos registros de app podem compartilhar o mesmo nome. O Application (client) ID gerado automaticamente do registro do app, e não seu nome de exibição, identifica seu app de forma única dentro da plataforma de identidade.
6. Especifique quem pode utilizar a aplicação, às vezes chamado de seu *sign-in audience*.
7. Selecione **Register** para concluir o registro inicial do app

Register an application - Microsoft

https://portal.azure.com

Microsoft Azure Search resources, services, and docs (G+/)

meganb@contoso.com CONTOSO AD (DEV)

Home > Contoso AD (dev) >

Register an application

* Name

The user-facing display name for this application (this can be changed later).

Supported account types

Who can use this application or access this API?

☒ Accounts in this organizational directory only (Contoso AD (dev) only - Single tenant)

☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant)

☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)

☐ Personal Microsoft accounts only

[Help me choose...](#)

Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web e.g. https://myapp.com/auth

By proceeding, you agree to the [Microsoft Platform Policies](#)

Register

Adicionar uma URI de redirecionamento

Uma *redirect URI* é o local para onde a plataforma de identidade da Microsoft redireciona o cliente de um usuário e envia tokens de segurança após a autenticação.

Em uma aplicação web de produção, por exemplo, a URI de redirecionamento é frequentemente um endpoint público onde sua aplicação está em execução, como `https://contoso.com/auth-response`. Durante o desenvolvimento, é comum também adicionar o endpoint onde você executa sua aplicação localmente, como `https://127.0.0.1/auth-response` OU `http://localhost/auth-response`.

Esta **RedirectURI** será usada posteriormente para configurar o **sgcOpenAPI Microsoft Client**.

Adicionar credenciais

As credenciais são usadas por aplicativos clientes confidenciais que acessam uma API web. Exemplos de clientes confidenciais são aplicativos web, outras APIs web ou aplicativos do tipo serviço e daemon. As credenciais permitem que seu aplicativo se autentique como ele mesmo, sem exigir interação de um usuário em tempo de execução.

Você pode adicionar tanto certificados quanto client secrets (uma string) como credenciais ao registro do seu aplicativo de cliente confidencial.

Microsoft Azure Search resources, services, and docs (G+)

testadmin@fourthcoffe... FOURTH COFFEE (FOURTHCOFFE...)

Home > Fourth Coffee > Contoso App 1

Contoso App 1 | Certificates & secrets

Search (Ctrl+/) Got feedback?

- Overview
- Quickstart
- Integration assistant

Manage

- Branding
- Authentication
- Certificates & secrets**
- Token configuration
- API permissions
- Expose an API
- App roles
- Owners
- Roles and administrators | Preview
- Manifest

Support + Troubleshooting

- Troubleshooting
- New support request

Credentials enable confidential applications to identify themselves to the authentication service when receiving tokens at a web addressable location (using an HTTPS scheme). For a higher level of assurance, we recommend using a certificate (instead of a client secret) as a credential.

Application registration certificates, secrets and federated credentials can be found in the tabs below.

Certificates (0) **Client secrets (0)** Federated credentials (0)

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

+ New client secret

Description	Expires	Value	Secret ID
No client secrets have been created for this application.			

Adicione um client secret

Às vezes chamado de *senha de aplicativo*, um client secret é um valor de string que seu app pode usar no lugar de um certificado para se identificar.

Os client secrets são considerados menos seguros que as credenciais de certificado. Os desenvolvedores de aplicações às vezes usam client secrets durante o desenvolvimento local do app por sua facilidade de uso. No entanto, você deve usar credenciais de certificado para quaisquer aplicações que estejam em produção.

1. No portal do Azure, em **App registrations**, selecione sua aplicação.
2. Selecione **Certificates & secrets** > **Client secrets** > **New client secret**.
3. Adicione uma descrição para seu client secret.
4. Selecione uma expiração para o secret ou especifique um tempo de vida personalizado.
 - O tempo de vida do client secret é limitado a dois anos (24 meses) ou menos. Você não pode especificar um tempo de vida personalizado superior a 24 meses.
 - A Microsoft recomenda que você defina um valor de expiração inferior a 12 meses.
5. Selecione **Add**.
6. *Registre o valor do secret* para uso no código da sua aplicação cliente. Este valor de secret *nunca é exibido novamente* depois que você sai desta página.

OpenAPI Microsoft | OAuth2 Code

O uso do OAuth2 Code Grant Flow requer interação com o usuário para fazer login e obter os privilégios necessários.

Uma vez que você tenha o [Tenant Id](#) e as [Credenciais](#), pode configurar as propriedades de OAuth2 para o Code Grant.

Para configurar o OpenAPI Client para o OAuth2 Code Grant, configure a propriedade **MicrosoftOptions.Authentication** com o seguinte valor:

```
GetOpenAPIClient->MicrosoftOptions->Authentication = oamaOAuth2Code;
```

Em seguida, você pode configurar os parâmetros do OAuth2

```
GetOpenAPIClient->MicrosoftOptions->Authentication = oamaOAuth2Code;
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->AuthURL = "https://login.microsoftonline.cc
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->TokenURL = "https://login.microsoftonline.cc
GetOpenAPIClient->Authentication->OAuth2->OAuth2Options->ClientId = "90945b8d-f6b7-4b97-b2bd-21c3c90b5f3b";
GetOpenAPIClient->Authentication->OAuth2->OAuth2Options->ClientSecret = "client_secret";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->IP = "127.0.0.1";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->Port = 8080;
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSL = True;
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->Port = 8080;
```

```

GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->CertFile = "sgc.pem";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->KeyFile = "sgc.pem";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->Password = "";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->RedirectURL = "https://localhost:8080";
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->Scope->Text = "
openid
";

```

O Tenant ID deve ser configurado para as URLs de Authentication e Token, basta substituir o Tenant Id correto na url.

A Microsoft só permite a URL **localhost** se você estiver escutando no Local IP, portanto defina a **redirect URL** com localhost como nome de dns em vez de configurar com o endereço IP.

O valor do **scope** depende da API, consulte a documentação da Microsoft / Azure para cada API.

Na **primeira vez** que uma requisição é feita, é exibido o navegador web pedindo que o usuário faça login em sua conta Microsoft. Se o usuário já tiver feito login anteriormente, a requisição HTTP será feita diretamente.

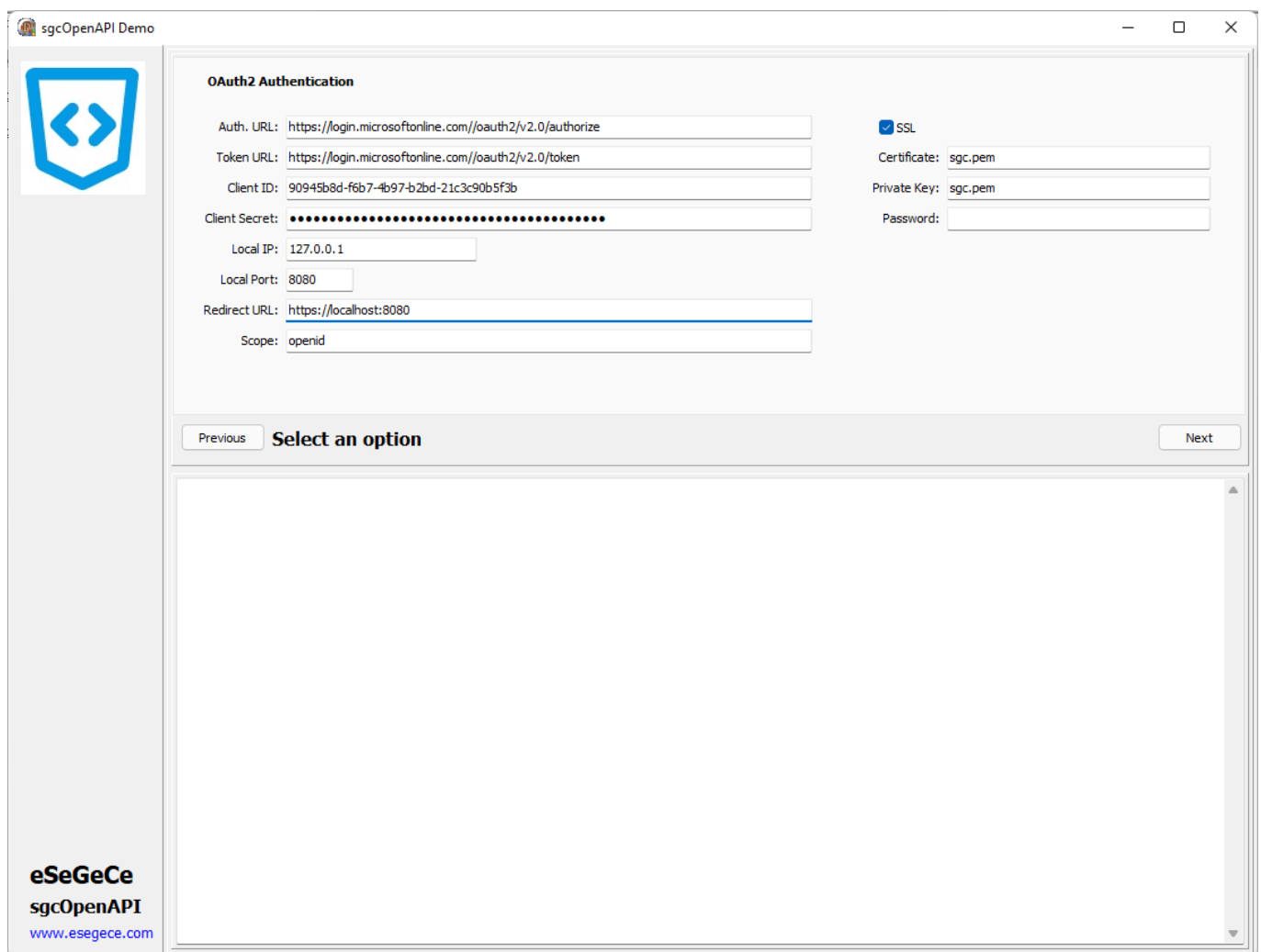
OpenAPI Microsoft | OAuth2 Credentials

Utilizar o OAuth2 Code Grant Flow não requer interação com o usuário, então é adequado para serviços, daemons... ou qualquer aplicação que deva ser executada sem interação do usuário.

Uma vez que você tenha o [Tenant Id](#) e as [Credenciais](#), pode configurar as propriedades de OAuth2 para o Code Grant.

Para configurar o OpenAPI Client para o OAuth2 Code Grant, configure a propriedade **MicrosoftOptions.Authentication** com o seguinte valor:

```
GetOpenAPIClient->MicrosoftOptions->Authentication = oamaOAuth2Credentials;
```



sgcOpenAPI Demo

OAuth2 Authentication

Auth. URL:

Token URL:

Client ID:

Client Secret:

Local IP:

Local Port:

Redirect URL:

Scope:

☒ SSL

Certificate:

Private Key:

Password:

Select an option

eSeGeCe
sgcOpenAPI
www.esegece.com

```
GetOpenAPIClient->MicrosoftOptions->Authentication = oamaOAuth2Credentials;
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->AuthURL = "https://login.microsoftonline.com/oauth2/v2.0/authorize";
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->TokenURL = "https://login.microsoftonline.com/oauth2/v2.0/token";
GetOpenAPIClient->Authentication->OAuth2->OAuth2Options->ClientId = "90945b8d-f6b7-4b97-b2bd-21c3c90b5f3b";
GetOpenAPIClient->Authentication->OAuth2->OAuth2Options->ClientSecret = "client_secret";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->IP = "127.0.0.1";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->Port = 8080;
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSL = True;
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->Port = 8080;
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->CertFile = "sgc.pem";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->KeyFile = "sgc.pem";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->Password = "";
```

```
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->RedirectURL = "https://localhost:8080";  
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->Scope->Text = "  
https://graph.microsoft.com/.default  
";
```

O Tenant ID deve ser configurado para as URLs de Authentication e Token, basta substituir o Tenant Id correto na url.

A Microsoft só permite a URL **localhost** se você estiver escutando no Local IP, portanto defina a **redirect URL** com localhost como nome de dns em vez de configurar com o endereço IP.

O valor do **scope** depende da API, consulte a documentação da Microsoft / Azure para cada API.

As credenciais OAuth2 não exigem nenhuma interação do usuário, portanto nenhum navegador será aberto na primeira chamada de requisição HTTP.

OpenAPI Microsoft | Graph

O Microsoft Graph expõe REST APIs e bibliotecas cliente para acessar dados nos seguintes serviços de nuvem da Microsoft:

- Serviços core do Microsoft 365: Bookings, Calendar, Delve, Excel, Microsoft 365 compliance eDiscovery, Microsoft Search, OneDrive, OneNote, Outlook/Exchange, People (contatos do Outlook), Planner, SharePoint, Teams, To Do, Workplace Analytics
- Serviços Enterprise Mobility + Security: Advanced Threat Analytics, Advanced Threat Protection, Azure Active Directory, Identity Manager e Intune
- Serviços do Windows: atividades, dispositivos, notificações, Universal Print
- Dynamics 365 Business Central

Get Current User

```

GetOpenAPIClient->MicrosoftOptions->Authentication = oamaOAuth2Code;
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->AuthURL = "https://login.microsoftonline.com/";
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->TokenURL = "https://login.microsoftonline.com/";
GetOpenAPIClient->Authentication->OAuth2->OAuth2Options->ClientId = "90945b8d-f6b7-4b97-b2bd-21c3c90b5f3b";
GetOpenAPIClient->Authentication->OAuth2->OAuth2Options->ClientSecret = "client_secret";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->IP = "127.0.0.1";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->Port = 8080;
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSL = True;
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->Port = 8080;
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->CertFile = "sgc.pem";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->KeyFile = "sgc.pem";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->Password = "";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->RedirectURL = "https://localhost:8080";
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->Scope->Text = "openid";

";
GetOpenAPIClient->meuserGetUser();

```


APIs

As APIs a seguir foram geradas utilizando o eSeGeCe sgcOpenAPI Generator e fornecidas gratuitamente a qualquer usuário. O código-fonte da interface é fornecido com a trial, para que você possa ver o que esperar caso adquira uma licença de qualquer uma das apis privadas como Google, Amazon ou Microsoft.

API	Descrição
Geolocation	A Abstract IP Geolocation API recebe um endereço IP e o traduz em uma localização detalhes, como endereço, fuso horário e mais.

OpenAPI | AbstractApi Geolocation

O que é a API IP Geolocation?

A Abstract IP Geolocation API recebe um endereço IP e o traduz em uma localização, bem como muitos outros detalhes, como um endereço, fuso horário e mais.

Quais são alguns casos de uso para a API de Geolocalização de IP?

Existem muitos casos de uso poderosos para APIs e dados de geolocalização por IP. Estes incluem, entre outros:

- Redireciona automaticamente os usuários para sites ou subsites relevantes com base em sua localização
- Detectar e exibir automaticamente a localização, o país ou o fuso horário de um usuário sem exigir que ele faça essa personalização explicitamente
- Personalize o conteúdo ou a experiência de um site ou app com base na localização do usuário. Por exemplo, exibir a previsão do tempo local do usuário, alíquotas de impostos e IVA, moeda, notícias, feriados, etc.
- Filtra os usuários com base em sua localização, por exemplo, se você não puder oferecer seus serviços a usuários em um país específico.
- Exigir que um usuário aceite certos termos conforme exigido por regulamentações locais, como banners de cookies do GDPR para cidadãos da União Europeia

Onde obter uma API Key?

Basta se registrar em [abstractapi.com](https://www.abstractapi.com) e você obterá uma api key gratuitamente

<https://www.abstractapi.com/api/ip-geolocation-api>

Código de Exemplo

O código a seguir retorna as informações sobre o endereço IP fornecido

```
ShowMessage(GetOpenAPIClient->Retrieve_the_location_of_an_IP_address("asdfkjlkj32i3j2liwj3es", "88.5.12.4"));
```

Demos | Server Chat

Este demo mostra como construir um Server Chat utilizando o [TsgcWebSocketHTTPServer](#) e WebSockets como protocolo de comunicação.

Toda vez que um novo peer envia uma mensagem, o servidor lê a mensagem e a transmite para todos os clientes conectados.

Start Server

Antes de iniciar o servidor, você deve configurá-lo para definir a porta de escuta, se usa uma conexão segura ou não...

- Primeiro eu crio uma nova instância de [TsgcWebSocketHTTPServer](#).
- Se o Servidor utilizar conexões seguras, ele precisa de um certificado PEM, basta definir onde está localizado este certificado e a porta de escuta para SSL. Você pode configurar a versão TLS e a API OpenSSL (se necessário)

```
// ... ssl
switch (cboOpenSSLAPI->ItemIndex)
{
    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->ItemIndex)
{
    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;
}
```

- Por padrão, se você iniciar o servidor, ele escutará em TODOS os IPs da porta de escuta, então é recomendado utilizar a propriedade binding para escutar em apenas 1 IP específico.

```
With WSServer->Bindings->Add do
{
    Port = StrToInt(txtDefaultPort->Text);
    IP = txtHost->Text;
}
```

- Uma vez configuradas todas as opções, chame `Server.Active = true` para iniciar o servidor.

Configuração de Eventos

- Utilize os eventos **OnConnect** e **OnDisconnect** para saber quando um cliente se conecta ao servidor.

- **Mensagens** enviadas do **cliente para o servidor** são recebidas no evento **OnMessage**, portanto use este manipulador de evento para retransmitir a mensagem recebida para todos os clientes

```
private void OnMessageEvent(TsgcWSConnection *Connection, string Text)
{
    server->Broadcast(Text);
}
```

Despachar Requisições HTTP

O WebSocket HTTP Server permite que você trate os protocolos WebSocket e HTTP na mesma porta de escuta, de modo que um navegador web possa requisitar uma página web para acessar seu servidor. O **OnCommandGet** é o evento utilizado para ler a HTTP Request e enviar as HTTP Responses.

Use o parâmetro **ARequestInfo** para ler a Requisição HTTP e **AResponseInfo** para escrever a Resposta HTTP.

Basicamente, use o **ARequestInfo.Document** para ler qual documento o cliente está solicitando e envie uma resposta usando as seguintes propriedades: **ResponseNo**, **ContentType** e **ContentText**.

Exemplo: um cliente solicita o documento '/jquery.js'

```
private void OnCommandGetEvent(TsgcWSConnection *Connection, TsgcWSHTTPRequestInfo *RequestInfo, ref TsgcWSHTTPResponseInfo *ResponseInfo)
{
    if (RequestInfo->Document == "/jquery.js")
    {
        ResponseInfo->ContentType = "text/javascript";
        ResponseInfo->ContentText = pageJQuery.Content;
        ResponseInfo->ResponseNo = 200;
    }
}
```

Client Chat

Esta demo mostra como construir um chat de cliente, usando o [TsgcWebSocketClient](#), que se conecta a um Servidor WebSocket, envia uma mensagem e essa mensagem é recebida por todos os clientes conectados.

Conectar ao Servidor

- Primeiro crie uma nova instância de [TsgcWebSocketClient](#).
- Em seguida, configure o **Host** e a **Port** do servidor.
- Se o cliente utiliza uma conexão segura, configure a propriedade **TLSOptions** do componente.

```
if (chkTLS->Checked)
{
    WSClient->Port = StrToInt(txtSSLPort->Text)
}
else
{
    WSClient->Port = StrToInt(txtDefaultPort->Text);
}
WSClient->Host = txtHost->Text;
case cboOpenSSLAPI->ItemIndex of
{
    0: WSClient->TLSOptions->OpenSSL_Options->APIVersion = sslAPI_1_0;
    1: WSClient->TLSOptions->OpenSSL_Options->APIVersion = sslAPI_1_1;
};
case cboTLSVersion->ItemIndex of
{
    0: WSClient->TLSOptions->Version = tlsUndefined;
    1: WSClient->TLSOptions->Version = tls1_0;
    2: WSClient->TLSOptions->Version = tls1_1;
    3: WSClient->TLSOptions->Version = tls1_2;
    4: WSClient->TLSOptions->Version = tls1_3;
}
WSClient->TLS = chkTLS->Checked;
```

- Uma vez que todas as opções podem ser configuradas, defina `Client.Active = true` para conectar ao servidor.

Enviar Mensagem para o Servidor

- Para enviar uma mensagem ao servidor, use o método **WriteData**, envie qualquer mensagem de texto e o servidor enviará como resposta a mesma mensagem.

```
WSClient->WriteData(txtName->Text + ": " + txtMessage->Text);
```

Receive Messages from Server

- Toda vez que uma nova mensagem de Texto é recebida pelo cliente, **OnMessage** o evento é disparado.

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

Demos | Client

Esta demo mostra como construir um cliente websocket, usando [TsgcWebSocketClient](#).

Conectar ao Servidor

- Primeiro crie uma nova instância de [TsgcWebSocketClient](#).
- Em seguida, configure o **Host** e a **Port** do servidor.
- Por padrão, o cliente conectará utilizando o **protocolo WebSocket**. Mas você pode configurar o cliente para conectar utilizando o **protocolo TCP simples**. Basta definir **Specifications.RFC6455 = false**, e o cliente utilizará o protocolo TCP simples em vez do protocolo WebSocket. Você pode ler mais sobre [conexões TCP](#).

```
WSClient->Host = txtHost->Text;
WSClient->Port = StrToInt(txtPort->Text);
WSClient->Options->Parameters = txtParameters->Text;
WSClient->TLS = chkTLS->Checked;
WSClient->Specifications->RFC6455 = chkOverWebSocket->Checked;
WSClient->Proxy->Enabled = chkProxy->Checked;
WSClient->Proxy->Username = txtProxyUsername->Text;
WSClient->Proxy->Password = txtProxyPassword->Text;
WSClient->Proxy->Host = txtProxyHost->Text;
if txtProxyPort->Text "" then
WSClient->Proxy->Port = StrToInt(txtProxyPort->Text);
WSClient->Extensions->PerMessage_Deflate->Enabled := chkCompressed->Checked;
// ... active
WSClient->Active = True;
```

Client Events

Utilize os eventos a seguir para controlar o fluxo do cliente: quando conecta, desconecta, recebe uma mensagem, um erro é detectado...

```
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 (" + IntToStr(CloseCode) + "): " + Connection->IP);
}
private void OnErrorEvent(TsgcWSConnection *Connection, string Error)
{
    DoLog("#error: " + Connection->IP + " - " + Error);
}
```

Demos | Cliente MQTT

Esta demo mostra como conectar a um servidor broker MQTT. Requer um [TsgcWebSocketClient](#) para tratar os protocolos WebSocket / TCP.

Configuração

- Primeiro crie uma nova instância de [TsgcWebSocketClient](#), verifique o [Demo do Cliente](#).
- Em seguida, crie uma nova instância de [TsgcWSPClient_MQTT](#).
- Depois disso, você deve atribuir o Protocolo MQTT ao cliente WebSocket e configurar as opções de conexão no cliente 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;
}
```

Eventos MQTT

O fluxo de conexão é controlado pelo componente Cliente MQTT, portanto você deve tratar os eventos MQTT para saber quando está conectado ao broker, quando uma nova mensagem é publicada, quando está desconectado...

```
private void OnMQTTConnectEvent(TsgcWSConnection *Connection, bool Session, int ReasonCode, string ReasonName, Ts
{
    DoLog("#MQTT Connect");
    chkTLS->Enabled = false;
    chkCompressed->Enabled = false;
    if (FMQTTSubscribeTopic != "")
    {
        mqtt->Subscribe(FMQTTSubscribeTopic);
        FMQTTSubscribeTopic = "";
    }
}

private void OnMQTTPublishEvent(TsgcWSConnection *Connection, string Topic, string Text, TsgcWSMQTTPublishPropert
{
    DoLog(Topic + ": " + Text);
}

private void OnMQTTSubscribeEvent(TsgcWSConnection *Connection, int PacketIdentifier, TsgcWSSUBACKS *Codes, TsgcW
{
    DoLog("#Subscribe: " + IntToStr(PacketIdentifier));
}

private void OnMQTTDisconnectEvent(TsgcWSConnection *Connection, int ReasonCode, string ReasonName, TsgcWSMQTTDIS
{
    DoLog("#disconnected");
    chkTLS->Enabled = true;
    chkCompressed->Enabled = true;
}
```


Demos | Client SocketIO

Esta demo mostra como conectar-se a um Servidor Socket.IO. Requer um [TsgcWebSocketClient](#) para tratar os protocolos WebSocket / TCP.

Configuração

- Primeiro crie uma nova instância de [TsgcWebSocketClient](#) , verifique o [Demo do Cliente](#).
- Em seguida, crie uma nova instância de [TsgcWSAPI_SocketIO](#).
- Depois disso, você deve atribuir a API Socket.IO ao cliente WebSocket e configurar as opções de conexão no cliente WebSocket.

```
if (socketio == null)
{
    socketio = new TsgcWSAPI_SocketIO();
    socketio->Client = client;
}
socketio->Client = client;
txtParameters->Text = "/";
chkTLS->Checked = true;
chkOverWebSocket->Checked = true;
```

Enviar mensagens

O Socket.IO usa **TsgcWebSocketClient** para enviar mensagens ao servidor, portanto basta chamar **WriteData** e passar como parâmetro a mensagem JSON para o servidor socket.io

```
client->WriteData("42[\"new message\", \"\" + txtSocketIOMessage.Text + "\"]");
```

Receive Messages

As mensagens recebidas conforme o fluxo da conexão são tratadas pelo TsgcWebSocketClient, então use este componente para ler as mensagens enviadas pelo servidor e para saber se a conexão está ativa ou não.

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

Demos | Server Monitor

Esta demo mostra como atualizar 3 Monitores HTML usando WebSocket como protocolo. O servidor tem um timer interno que atualiza aleatoriamente os valores dos medidores e atualiza o valor usando uma mensagem websocket. Essa mensagem é lida pelo cliente javascript e atualiza o valor do Medidor.

Configuração

- Primeiro crie uma nova instância de [TsgcWebSocketServer](#), confira a [Demo de Chat do Servidor](#).
- Em seguida, crie um novo Timer e, a cada 500 milissegundos, atualize os valores de: memória, rede ou cpu. Envie a atualização para todos os clientes conectados.
- No cliente Javascript, leia a mensagem enviada pelo servidor e atualize o valor do medidor.

```
<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>

```


Demos | Server Snapshots

Este demo mostra como enviar imagens do servidor para o cliente e como todos os clientes recebem a mesma imagem utilizando o método broadcast do componente de servidor.

Configuração

- Primeiro, crie uma nova instância de [TsgcWebSocketHTTPServer](#), consulte a [Demo de Chat do Servidor](#).
- Habilite a compressão para enviar menos bytes quando a mensagem é transmitida aos clientes

```
Server.Extensions.PerMessage_Deflate.Enabled = true
```

- Então, a cada 5 segundos, o servidor faz broadcast de um stream de imagem para todos os clientes conectados

```
void DoBroadcastStream()
{
    TBitmap *oBitmap = new TBitmap();
    try
    {
        // load bitmap
        if (WSServer->Active)
        {
            TMemoryStream *oStream = new TMemoryStream();
            Try
            {
                oBitmap->SaveToStream(oStream);
                oStream->Seek(0, soFromBeginning);
                WSServer->Broadcast(oStream);
            }
            __finally
            {
                FreeAndNil(oStream);
            }
        }
    }
    __finally
    {
        oBitmap->FreeImage;
        FreeAndNil(oBitmap);
    }
}
```

Demos | Client Snapshots

Esta demo mostra como ler mensagens websocket binárias, usando o [TsgcWebSocketClient](#), que se conecta a um Servidor WebSocket e recebe um stream que é uma imagem exibida ao usuário.

Conectar ao Servidor

- Primeiro crie uma nova instância de [TsgcWebSocketClient](#).
- Em seguida, configure o **Host** e a **Port** do servidor.
- Habilite a compressão para receber menos bytes quando a mensagem for transmitida do servidor.

```
Client.Extensions.PerMessage_Deflate.Enabled = true
```

- A imagem enviada pelo servidor chega como um fluxo, portanto use o evento **OnBinary** para ler as imagens.

```
void OnClientBinary(TsgcWSConnection *Connection, TMemoryStream *Data)
{
    TBitmap *oBitmap = new TBitmap();
    try
    {
        oBitmap->LoadFromStream(Data);
        Image1->Picture->Assign(oBitmap);
        memoLog->Lines->Add(
            "#image uncompressed size: " + IntToStr(Data->Size) +
            ". Total received: " + IntToStr(Connection->RecBytes));
    }
    finally
    {
        FreeAndNil(oBitmap);
    }
}
```

Demos | Upload de Arquivo

Esta demo mostra como fazer upload de um arquivo do navegador web para um servidor utilizando o protocolo websocket.

Configuração

- Primeiro crie uma nova instância de [TsgcWebSocketServer](#), confira a [Demo de Chat do Servidor](#).
- O arquivo chegará ao servidor como um stream binário, portanto você deve tratar o evento **OnBinary** para ler o arquivo.

```
void OnServerBinary(TsgcWSConnection *Connection, const TMemoryStream *Data)
{
    if (FFileName == "")
        FFileName = FormatDateTime("yyyymmddhhnnsszz", Now) + ".dat";
    TFileStream *oFile = new TFileStream(FFileName, fmCreate);
    try
    {
        oFile->CopyFrom(Data, Data->Size);
    }
    __finally
    {
        delete oFile;
    }
    memoLog->Lines->Add("Received File: " + FFileName);
}
```

- Se você quiser saber o nome do arquivo, pode enviar uma mensagem de texto com o nome do arquivo antes de o arquivo ser enviado

```
void OnServerMessage(TsgcWSConnection *Connection, const String Text)
{
    if (LeftStr(Text, Length(CS_UPLOAD_FILE)) == CS_UPLOAD_FILE)
        FFileName = MidStr(Text, Length(CS_UPLOAD_FILE) + 1, Length(Text));
    else
        memoLog->Lines->Add("Message Received (" + Connection->Guid + "): " + Text);
}
```

O código **javascript** para enviar um arquivo utilizando **websockets** é mostrado abaixo:

```
<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>
```


Demos | Autenticação de Servidor

Este demo mostra como utilizar a Autenticação de Servidor; se você quiser saber mais sobre os diferentes tipos de autenticação, leia o seguinte artigo sobre [Autenticação](#).

Autenticação

- Primeiro, crie uma nova instância de [TsgcWebSocketServer](#). Habilite a propriedade Authentication, `server.Authentication.Enabled = true;`
- Em seguida, verifique no handler do evento **OnAuthentication** se o nome de usuário e a senha estão corretos. Se estiverem corretos, defina a propriedade `Authenticated` como `true`, caso contrário defina como `false`.

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

Demos | KendoUI_Grid

Este demo mostra como o KendoUI Grid funciona usando WebSockets como protocolo e um Navegador Web como cliente. Basicamente, é um grid javascript que é atualizado quando qualquer um dos clientes faz alguma alteração; essas alterações são atualizadas usando o protocolo websocket para todos os clientes conectados, então todos os clientes podem ver em tempo real os mesmos dados, incluindo todas as alterações feitas pelos clientes.

Configuração

- Primeiro, crie uma nova instância de [TsgcWebSocketHTTPServer](#), consulte a [Demo de Chat do Servidor](#).
- Em seguida, você deve tratar OnCommandGet para enviar os arquivos necessários solicitados pelos clientes de navegador web.

```
void OnCommandGet(TidContext *AContext, TidHTTPRequestInfo *ARequestInfo, TidHTTPResponseInfo *AResponseInfo)
{
    if (ARequestInfo->Document == "/jquery.mobile.css")
    {
        AResponseInfo->ContentText = pagejQueryMobileCSS->Content;
        AResponseInfo->ContentType = "text/css";
        AResponseInfo->ResponseNo = 200;
    }
    else if (ARequestInfo->Document == "/jquery.js")
    {
        AResponseInfo->ContentText = pagejQuery->Content;
        AResponseInfo->ContentType = "text/javascript";
        AResponseInfo->ResponseNo = 200;
    }
    else if (ARequestInfo->Document == "/jquery.mobile.js")
    {
        AResponseInfo->ContentText = pagejQueryMobile->Content;
        AResponseInfo->ContentType = "text/javascript";
        AResponseInfo->ResponseNo = 200;
    }
    else
    {
        if (AContext->Connection->Socket->Binding->Port == WSServer->SSLOptions->Port)
            FRequestSSL = true;
        else
            FRequestSSL = false;
        AResponseInfo->ContentText = pageKendoUI_Grid->Content;
        AResponseInfo->ContentType = "text/html";
        AResponseInfo->ResponseNo = 200;
    }
}
```

Atualizações de WebSockets

Quando um cliente atualiza um registro de grid, esta mudança é transmitida a todos os clientes conectados utilizando o protocolo websocket. Utilize o evento OnMessage para ser notificado sobre as mudanças no grid. As mensagens estão em formato JSON, portanto você só precisa ler o texto JSON, decodificá-lo e enviar uma resposta ao outro peer.

```
void OnServerMessage(TsgcWSCConnection *Connection, const String Text)
{
    TsgcJSON *oJSON = new TsgcJSON();
    try
    {
        oJSON->Read(Text);
        // ... read
        if (oJSON->Node["type"]->Value == "read")
        {
            TsgcObjectJSON *oArray = oJSON->AddArray("data");
            for (int i = 0; i < 20; i++)
            {
                TsgcJSON *oObj = oArray->JSONObject->AddObject(IntToStr(i))->JSONObject;
```

```

        oObj->AddPair("ContactID", i);
        oObj->AddPair("ContactName", ContactName[i]);
        oObj->AddPair("ContactTitle", ContactTitle[i]);
        oObj->AddPair("CompanyName", CompanyName[i]);
        oObj->AddPair("Country", Country[i]);
    }
    Connection->WriteData(oJSON->Text);
}
// ... update
else if (oJSON->Node["type"]->Value == "update")
{
    WSServer->Broadcast(StringReplace(Text, "\"type\": \"update\"", "\"type\": \"push-update\"", []), "", "",
        Connection->Guid);
    Connection->WriteData(Text);
}
// ... destroy
else if (oJSON->Node["type"]->Value == "destroy")
{
    WSServer->Broadcast(StringReplace(Text, "\"type\": \"destroy\"", "\"type\": \"push-destroy\"", []), "", "",
        Connection->Guid);
    Connection->WriteData(Text);
}
// ... create
else if (oJSON->Node["type"]->Value == "create")
{
    String vText = StringReplace(Text, "null", FormatDateTime("yyyymmddhhnnsszzz", Now), []);
    WSServer->Broadcast(StringReplace(vText, "\"type\": \"create\"", "\"type\": \"push-create\"", []), "", "",
        Connection->Guid);
    Connection->WriteData(vText);
}
}
finally
{
    oJSON->Free();
}
}

```

Demos | ServerSentEvents

Este demo mostra como os Server-Sent Events funcionam no WebSocket Server. O `sgcWebSockets` permite que o servidor trate mais de um protocolo na mesma porta de escuta.

Você pode ler mais sobre [Server Sent Events](#).

Este demo mostra como o servidor enviará a cada segundo a hora para todos os clientes conectados utilizando Server Sent Events.

Uma vez que o servidor é iniciado, ele faz **broadcast** para todos os clientes conectados de uma mensagem com o **Server Time**, de modo que, toda vez que o cliente recebe esta mensagem, ele a exibe ao usuário.

```
private void Timer1Timer(TOobject *sender)
{
    server->Broadcast("data: " + "Server Time: " + FormatDateTime("hh:nn:ss", Now));
}
```

O código javascript para tratar a conexão websocket é mostrado abaixo:

```
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";
});
```

Exemplos | Servidor WebRTC

Esta demo mostra como construir um Servidor de Videoconferência usando o [TsgcWebSocketHTTPServer](#) e WebRTC como biblioteca javascript.

O demo usa o protocolo WebSocket para sinalizar o WebRTC e usa servidores STUN/TURN públicos; para sites de produção, você precisa usar seus próprios servidores STUN/TURN. Usuários registrados podem baixar o Co-urn para windows, que já vem compilado e com todas as bibliotecas necessárias para executar nos seus servidores.

O cliente deve ser um navegador web com suporte para conexões WebRTC.

Configuração

- Primeiro, crie uma nova instância de [TsgcWebSocketHTTPServer](#), consulte a [Demo de Chat do Servidor](#).
- Em seguida, crie uma nova instância de [TsgcWSPServer_WebRTC](#).
- Depois disso, você deve atribuir o Protocolo WebRTC ao Servidor WebSocket e configurar o host e a porta do servidor.

```
WSServer->Port = StrToInt(txtDefaultPort->Text);  
// ... bindings  
With WSServer->Bindings->Add do  
{  
  Port = StrToInt(txtDefaultPort->Text);  
  IP = txtHost>Text;  
}  
// ... active  
WSServer->Active = true;
```

A demo requer uma página HTML de índice que é utilizada para despachar a página inicial do WebRTC, esta página é fornecida com a demo.

Executar no WebBrowser

Uma vez configurado o servidor, inicie-o e selecione um dos navegadores web disponíveis. Ele abrirá uma nova sessão de navegador solicitando o início de uma nova sessão. Se for bem-sucedido, você verá o seu vídeo e, se abrir a mesma url em outro navegador, verá ambos os peers conectados.

A demo é executada por padrão sem SSL, o que é válido apenas para conexões localhost. Para sites de produção, use conexões SSL. Confira a [Server Chat Demo](#) para configurar SSL no lado do servidor.

Demos | Server AppRTC

Este demo mostra como construir um Servidor de Videoconferência utilizando [TsgcWebSocketHTTPServer](#) e o AppRTC como biblioteca javascript.

O demo usa o protocolo WebSocket para sinalizar o WebRTC e usa servidores STUN/TURN públicos; para sites de produção, você precisa usar seus próprios servidores STUN/TURN. Usuários registrados podem baixar o CoTurn para windows, que já vem compilado e com todas as bibliotecas necessárias para executar nos seus servidores.

O cliente deve ser um navegador web com suporte para conexões WebRTC.

Configuração

- Primeiro, crie uma nova instância de [TsgcWebSocketHTTPServer](#), consulte a [Demo de Chat do Servidor](#).
- Em seguida, crie uma nova instância de [TsgcWSPServer_AppRTC](#).
- Depois disso, você deve atribuir o Protocolo AppRTC ao Servidor WebSocket e configurar o host e a porta do servidor. O WebRTC requer conexões seguras, portanto você precisará usar um certificado PEM e configurar a propriedade SSLOptions do componente.

```
WSServer->Port = StrToInt(txtDefaultPort->Text);
// ... bindings
With WSServer->Bindings->Add do
{
    Port = StrToInt(txtDefaultPort->Text);
    IP = txtHost->Text;
}
// ... properties
WSPAppRTC->AppRTC->RoomLink = "https://" + txtHost->Text + ":" + txtDefaultPort->Text + "/r/";
WSPAppRTC->AppRTC->WebSocketURL = "wss://" + txtHost->Text + ":" + txtDefaultPort->Text;
// ... active
WSServer->Active = true;
```

- AppRTC.RoomLink é a url para a qual o navegador web será redirecionado para fazer login em uma sala
- AppRTC.WebSocketURL é a url da conexão websocket
- Os IceServers podem ser configurados no protocolo AppRTC Server.

A demo requer uma página HTML de índice que é utilizada para despachar a página inicial do AppRTC, esta página é fornecida com a demo.

Executar no WebBrowser

Uma vez configurado o servidor, inicie-o e selecione um dos navegadores web disponíveis. Ele abrirá uma nova sessão de navegador web pedindo para entrar em uma nova sala. Entre nesta sala e, se bem-sucedido, você verá um link que deve ser utilizado a partir de outro navegador web para iniciar uma nova videoconferência.

AppRTC

Please enter a room name.

959454873

JOIN

RANDOM

Recently used rooms:

Demos | Telegram Client

Esta demo mostra como conectar ao Telegram, receber todos os contatos, enviar mensagens de Texto, enviar Imagens... e muito mais

Configuração

- Primeiro crie uma nova instância de [TsgcTDLib_Telegram](#).
- Em seguida, antes de tentar conectar ao telegram, você deve passar alguns parâmetros para o componente cliente, como API Hash, API Id... Uma vez definidos todos os parâmetros obrigatórios, defina a propriedade `Active = true` para iniciar uma conexão.

```
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 o cliente tenta conectar-se ao Telegram, normalmente um código é necessário, então você deve tratar **OnTelegramAuthenticationCode** e retornar o parâmetro `Code` com o valor fornecido pela sua conta Telegram.

```
private void OnAuthenticationCodeEvent(TObject *Sender, ref string Code)
{
    Code = InputBox("Telegram", "Introduce Telegram Code");
}
```

Send Telegram Messages

Para enviar uma mensagem do Telegram (texto, arquivos, imagens...) sempre é necessário primeiro definir o ChatId para onde você quer enviar a mensagem e então o parâmetro que pode ser uma mensagem de texto, um nome de arquivo...

```
// send text message
sgcTelegram->SendTextMessage("456413", "Hello From sgcWebSockets!!!");

// send file message
sgcTelegram->SendDocumentMessage("383784", "c:\\yourfile.txt");
```

Receive Telegram Messages

As mensagens recebidas pelo cliente Telegram são tratadas em handlers de eventos específicos. Há um evento para quando uma próxima Mensagem de Texto é recebida, quando um novo Documento é recebido, foto...


```
private void OnMessageTextEvent(TObject *Sender, TsgcTelegramMessageText *MessageText)
{
    DoLogMessage(MessageText->ChatId, MessageText->SenderId->ToString(), MessageText->Text);
}

private void OnMessageDocumentEvent(TsgcTDLib_Telegram *Sender, TsgcTelegramMessageDocument *MessageDocument)
{
    DoLogMessage(MessageDocument->ChatId, MessageDocument->SenderId->ToString(), MessageDocument->FileName);
}
```

Coturn

Coturn

A partir do sgcWebSockets 4.5.2 ENTERPRISE Edition, você pode construir seu próprio servidor STUN/TURN utilizando Delphi/CBuilder.

É uma implementação gratuita e de código aberto de servidores TURN e STUN.

O TURN Server é um servidor e gateway de travessia de NAT para tráfego de mídia VoIP. Também pode ser usado como um servidor e gateway TURN de tráfego de rede de propósito geral.

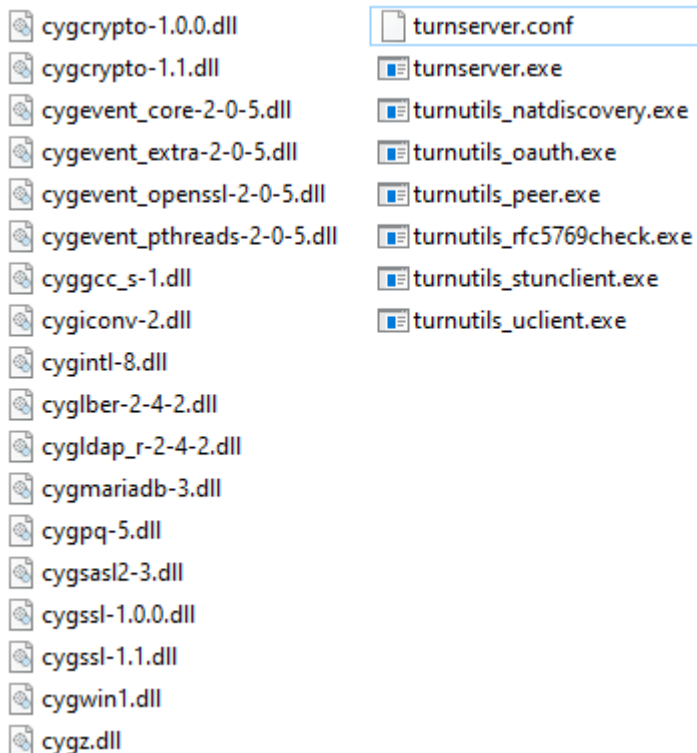
As plataformas-alvo de projeto suportadas são:

- **Linux**
- **Mac OS X**
- **Windows** (Cygwin): binários compilados estão disponíveis para usuários registrados.

Windows Configuration

Primeiro você deve baixar os binários compilados da sua conta; há 2 versões disponíveis: win32 e win64. Selecione a plataforma desejada e descompacte os binários em uma pasta. Os seguintes arquivos serão criados:

1. Algumas bibliotecas cygwin necessárias para executar a aplicação; você deve implantar essas bibliotecas com o servidor coturn.
2. Algumas aplicações de console:
 - 2.1 turnserver.exe: é a principal aplicação de console para executar um servidor TURN/STUN
 - 2.2 Outras aplicações: são utilizadas para fins de configuração ou de teste.
3. Turnserver.conf: é o arquivo de configuração do servidor coturn.



turnserver.conf

Este é o arquivo de configuração do servidor coturn; se você abri-lo, verá uma configuração padrão.

Configuração Simples

Seu servidor tem o seguinte IP público 80.15.44.123 e escuta na porta 80. As credenciais para conexão são: username = demo, password = secret
Defina a seguinte configuração:

```
listening-ip=80.15.44.123
listening-port=80
realm=yourrealm.com
user=demo:secret
```

Configuração com TLS habilitado

O servidor tem o seguinte IP público 80.15.44.123 e escuta nas portas 80 e 443 (conexões TLS). As credenciais para conexão são: username = demo, password = secret. O nome do seu certificado (deve estar em formato PEM) é certificate.crt e a chave privada é private.key.
Defina a seguinte configuração:

```
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
```

Há mais configurações disponíveis, basta abrir o turnserver.conf e ler as seções documentadas.

Executar coturn

Uma vez configurado, você pode executar o servidor apenas executando turnserver.exe; uma nova aplicação de console será aberta e um arquivo de log será criado. Você pode aumentar a verbosidade da aplicação de console (obter mensagens mais detalhadas) se habilitar "verbose" no arquivo 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
```

License

Contrato de Licença de Usuário Final dos Componentes eSeGeCe

O End-User License Agreement ("EULA") dos eSeGeCe Components ("eSeGeCe") é um acordo legal entre você (seja um indivíduo ou uma única entidade) e o Autor do eSeGeCe para todos os eSeGeCe components que podem incluir componentes de software associados, mídia, materiais impressos, e documentação "online" ou eletrônica ("eSeGeCe components"). Ao instalar, copiar ou de outra forma utilizar os eSeGeCe components, você concorda em ficar vinculado aos termos deste EULA. Este acordo de licença representa o acordo integral relativo ao programa entre você e o Autor do eSeGeCe, (referido como "LICENSER"), e ele substitui qualquer proposta, representação ou entendimento anterior entre as partes. Se você não concordar com os termos deste EULA, não instale nem utilize os eSeGeCe components.

Os componentes eSeGeCe são protegidos por leis de direitos autorais e tratados internacionais de direitos autorais, bem como por outras leis e tratados de propriedade intelectual. Os componentes eSeGeCe são licenciados, não vendidos.

Se você deseja o CÓDIGO-FONTE, precisa pagar a taxa de registro. Você NÃO deve fornecer as chaves de licença e/ou as edições completas do eSeGeCe (incluindo as edições DCU e edições Source) a quaisquer indivíduos e/ou entidades terceiras. E você também NÃO deve utilizar as chaves de licença e/ou as edições completas do eSeGeCe de quaisquer indivíduos e/ou entidades terceiras.

1. CONCESSÃO DE LICENÇA

Os componentes eSeGeCe são licenciados da seguinte forma:

(a) Instalação e Uso.

O LICENSER concede a você o direito de instalar e utilizar cópias dos componentes eSeGeCe no seu computador executando uma cópia validamente licenciada do sistema operacional para o qual os componentes eSeGeCe foram projetados [por exemplo, Windows 2000, Windows 2003, Windows XP, Windows ME, Windows Vista, Windows 7, Windows 8, Windows 10].

(b) Royalty Free.

Você pode criar aplicações comerciais baseadas nos componentes eSeGeCe e distribuí-las com seus executáveis, sem necessidade de royalties.

(c) Modificações (somente edições com Source).

Você pode fazer modificações, melhorias, trabalhos derivados e/ou extensões ao SOURCE CODE licenciado fornecido a você sob os termos estabelecidos neste contrato de licença.

(d) Cópias de Backup.

Você também pode fazer cópias dos componentes eSeGeCe conforme necessário para fins de backup e arquivamento.

2. DESCRIÇÃO DE OUTROS DIREITOS E LIMITAÇÕES

(a) Manutenção dos Avisos de Direitos Autorais.

Você não deve remover nem alterar quaisquer avisos de copyright em toda e qualquer cópia dos componentes eSeGeCe.

(b) Distribuição.

Você não pode distribuir cópias registradas dos componentes eSeGeCe a terceiros. As edições de avaliação disponíveis para download nos sites oficiais da eSeGeCe podem ser distribuídas livremente.

Você pode criar componentes/controles ActiveX/bibliotecas que incluam os componentes eSeGeCe para suas aplicações, mas você NÃO deve distribuí-los nem publicá-los a terceiros.

(c) Proibição de distribuição do CÓDIGO-FONTE (somente edições Source).

Você NÃO deve distribuir nem publicar o CÓDIGO-FONTE, nem qualquer modificação, aprimoramento, trabalho derivado e/ou extensões, em forma de CÓDIGO-FONTE para terceiros.

Você NÃO deve fazer com que qualquer parte do CÓDIGO-FONTE seja distribuída, publicada, divulgada ou de outra forma disponibilizada a terceiros.

(d) Proibição de Engenharia Reversa, Descompilação e Desmontagem.

Você não pode fazer engenharia reversa, descompilar ou desmontar os componentes eSeGeCe, exceto e somente na medida em que tal atividade seja expressamente permitida pela lei aplicável, não obstante esta limitação.

(e) Aluguel.

Você não pode alugar, arrendar ou emprestar os componentes eSeGeCe.

(f) Serviços de Suporte.

O LICENSER pode fornecer serviços de suporte relacionados aos componentes eSeGeCe ("Serviços de Suporte"). Qualquer código de software suplementar fornecido a você como parte dos Serviços de Suporte será considerado parte dos componentes eSeGeCe e estará sujeito aos termos e condições deste EULA.

eSeGeCe é licenciado para ser usado por apenas um desenvolvedor por vez. E o suporte técnico será fornecido a apenas um determinado desenvolvedor.

(g) Conformidade com as Leis Aplicáveis.

Você deve cumprir todas as leis aplicáveis relativas ao uso dos componentes eSeGeCe.

3. RESCISÃO

Sem prejuízo de quaisquer outros direitos, o LICENCIANTE pode rescindir este EULA se você deixar de cumprir os termos e condições deste EULA. Nesse caso, você deve destruir todas as cópias dos componentes da eSeGeCe em sua posse.

4. COPYRIGHT

Todos os direitos de titularidade, incluindo, sem limitação, direitos autorais, sobre os componentes eSeGeCe e quaisquer cópias deles pertencem ao LICENCIANTE ou a seus fornecedores. Todos os direitos de titularidade e de propriedade intelectual sobre o conteúdo que pode ser acessado por meio do uso dos componentes eSeGeCe são propriedade do respectivo proprietário do conteúdo e podem ser protegidos por leis e tratados de direitos autorais ou outros direitos de propriedade intelectual aplicáveis. Este EULA não lhe concede nenhum direito de utilizar tal conteúdo. Todos os direitos não expressamente concedidos são reservados pelo LICENCIANTE.

5. SEM GARANTIAS

O LICENSER expressamente se isenta de qualquer garantia para os componentes eSeGeCe. Os componentes eSeGeCe são fornecidos "no estado em que se encontram", sem qualquer garantia expressa ou implícita de qualquer tipo, incluindo, mas não se limitando a, quaisquer garantias de comercialização, não violação ou adequação a um propósito específico. O LICENSER não garante nem assume responsabilidade pela exatidão ou integridade de qualquer informação, texto, gráficos, links ou outros itens contidos nos componentes eSeGeCe. O LICENSER não oferece garantias com relação a qualquer dano que possa ser causado pela transmissão de um vírus de computador, worm, time bomb, logic bomb ou outro programa de computador semelhante. O LICENSER ainda se isenta expressamente de qualquer garantia ou declaração aos Usuários Autorizados ou a qualquer terceiro.

6. LIMITAÇÃO DE RESPONSABILIDADE

Em nenhuma hipótese o LICENCIADOR será responsável por quaisquer danos (incluindo, sem limitação, lucros cessantes, interrupção de negócios ou perda de informações) decorrentes do uso ou da incapacidade de uso pelos "Usuários Autorizados" dos componentes eSeGeCe, mesmo que o LICENCIADOR tenha sido avisado da possibilidade de tais danos. Em nenhuma hipótese o LICENCIADOR será responsável por perda de dados ou por danos indiretos, especiais, incidentais, consequenciais (incluindo lucros cessantes), ou outros danos baseados em contrato, ato ilícito ou de outra forma. O LICENCIADOR não terá responsabilidade alguma com relação ao conteúdo dos componentes eSeGeCe ou qualquer parte deles, incluindo, mas não se limitando a, erros ou omissões neles contidos, difamação, violações de direitos de imagem, privacidade, direitos de marca registrada, interrupção de negócios, danos pessoais e perda de privacidade, direitos morais ou a divulgação de informações confidenciais.

Índice

- AbstractApi Geolocation [1118](#)
- Add Telegram Proxy [674](#)
- ALPN [128](#)
- Amazon SQS [961](#)
- AMQP Channels [319](#)
- AMQP Consume Messages [327](#)
- AMQP Exchanges [321](#)
- AMQP Get Messages [329](#)
- AMQP Publish Messages [326](#)
- AMQP QoS [330](#)
- AMQP Queues [323](#)
- AMQP Transactions [331](#)
- AMQP1 Read Message [365](#)
- AMQP1 Receiver Links [361](#)
- AMQP1 Send Message [363](#)
- AMQP1 Sender Links [358](#)
- Anthropic [762](#)
- Anthropic Batches [774](#)
- Anthropic Citations [781](#)
- Anthropic Extended Thinking [775](#)
- Anthropic Files [786](#)
- Anthropic MCP Connector [788](#)
- Anthropic Messages [767](#)
- Anthropic Models [773](#)
- Anthropic Prompt Caching [779](#)
- Anthropic Structured Outputs [784](#)
- Anthropic Tool Use [771](#)
- Anthropic Vision [769](#)
- Anthropic Web Search [782](#)
- API 3Commas [564](#)
- API Binance [429](#), [445](#), [452](#)
- API Binance Futures [445](#), [452](#)
- API Binance Futures Trade [452](#)
- API Bitfinex [532](#)
- API Bitget [622](#)
- API Bitmex [523](#)
- API Blockchain [594](#)
- API Bybit [579](#)
- API Cex [596](#)
- API Cex Plus [604](#)
- API Coinbase [455](#)
- API Coinbase Pro [455](#)
- API Crypto.com [629](#)
- API Deribit [626](#)
- API Discord [609](#)
- API Forex [584](#)
- API GateIO [624](#)
- API HTX [631](#)
- API Kraken [476](#), [478](#), [484](#), [487](#), [489](#), [492](#), [494](#), [501](#), [507](#), [509](#)
- API Kraken Futures [492](#), [494](#), [501](#), [507](#), [509](#)
- API Kraken Futures REST Privada [509](#)
- API Kraken Futures REST Public [507](#)
- API Kraken REST Private [489](#)
- API Kraken REST Public [487](#)
- API Kucoin [536](#)
- API Kucoin Futures [550](#)
- API MEXC [614](#)
- API MEXC Futures [618](#)
- API OKX [569](#)
- API OpenAI [612](#)
- API Pusher [514](#)
- API SignalR [473](#)
- API SignalRCore [467](#)
- API SocketIO [453](#)
- API Telegram [658](#)
- API Whatsapp [633](#)
- API XTB [575](#)
- APIs [427](#), [429](#), [437](#), [440](#), [445](#), [452](#), [453](#), [455](#), [460](#), [463](#), [467](#), [473](#), [476](#), [478](#), [484](#), [487](#), [489](#), [492](#), [494](#), [501](#), [507](#), [509](#), [514](#), [523](#), [532](#), [564](#), [584](#), [591](#), [594](#), [596](#), [609](#), [614](#), [618](#), [658](#), [1117](#)
- APNs [863](#), [864](#), [865](#)
 - Conexão Baseada em Certificado [865](#)
 - Token-Based Connection [864](#)
- Apple Push Notifications [861](#)
- Arquivos [119](#)
- Autenticação [109](#), [166](#), [191](#)
- Autenticação do cliente [166](#)
- Autenticação do servidor [191](#), [905](#)
- Authorization Code com PKCE (RFC 7636) [882](#)

Authorization Code Grant [907](#)
 Authorization Code Grant (RFC 6749) [880](#)
 Auto Ban [239](#)
 Binance Connect [437](#)
 Binance Get Market Data [439](#)
 Binance Private Requests Time [443](#)
 Binance Private REST API [440](#)
 Binance Subscribe [438](#)
 Binance Trade Spot [441](#)
 Binance Withdraw [444](#)
 Binary Message [163](#)
 Bindings [118](#), [178](#)
 Bitmex Connect WebSocket API [527](#)
 Bitmex Subscribe WebSocket Channel [528](#)
 Bot [670](#)
 Bot Detection [247](#)
 Broadcast [117](#)
 Brute Force [239](#)
 Build [72](#), [73](#), [75](#), [76](#)
 Build Aplicativo Android [75](#)
 Build iOS Application [76](#)
 Build OSX Application [73](#)
 Canais [117](#)
 Certificados Autoassinados [222](#)
 Certificates OpenSSL [157](#)
 Certificates SChannel [158](#)
 Channels [319](#), [438](#), [461](#), [592](#), [1033](#)
 Chat do Cliente [1121](#)
 CIDR [237](#)
 Client [148](#), [150](#), [151](#), [161](#), [162](#), [163](#), [169](#), [171](#),
[172](#), [197](#), [301](#), [303](#), [304](#), [317](#), [318](#), [336](#), [852](#),
[853](#), [855](#), [856](#), [1021](#), [1022](#), [1023](#), [1030](#), [1032](#),
[1033](#), [1122](#), [1123](#), [1125](#), [1130](#), [1140](#)
 Client AMQP Connect [317](#)
 Client AMQP Disconnect [318](#)
 Client AMQP1 Authentication [352](#)
 Client AMQP1 Azure MessageBus [353](#)
 Client AMQP1 Connect [348](#)
 Client AMQP1 Disconnect [349](#)
 Client AMQP1 Idle Timeout Connection [350](#)
 Client Authentication [856](#)
 Client Close Connection [150](#), [852](#)
 Client Connections [190](#)
 Client Credentials Grant [909](#)
 Client Credentials Grant (RFC 6749) [884](#)
 Client Exceptions [169](#)
 Client Kafka Connect [336](#)
 Client Keep Connection Active [853](#)
 Client Keep Connection Open [151](#)
 Client MQTT Connect [301](#)
 Client MQTT Sessions [303](#)
 Client MQTT Version [304](#)
 Client Open Connection [148](#)
 Client Pending Requests [855](#)
 Client Proxies [172](#)
 Client Register Protocol [171](#)
 Client Send Binary Message [162](#)
 Client Send Text [161](#), [163](#)
 Client Send Text Message [161](#)
 Client Snapshots [1130](#)
 Client SocketIO [1125](#)
 Cliente [166](#), [170](#), [337](#), [414](#), [1031](#), [1121](#)
 Clientes [337](#), [414](#), [1121](#)
 Enviar Arquivos [414](#)
 Clients [303](#), [317](#), [336](#), [853](#), [855](#), [1123](#)
 Coinbase Pro Connect [460](#)
 Coinbase Pro Hora das Requisições Privadas
[464](#)
 Coinbase Pro Obter Dados de Mercado [462](#)
 Coinbase Pro Place Orders [465](#)
 Coinbase Pro Private REST API [463](#)
 Coinbase Pro SandBox Account [466](#)
 Coinbase Pro Subscribe [461](#)
 CommitOffset [342](#)
 CommitSync [340](#)
 Como registrar uma ordem na Bitmex [529](#)
 Compressão [121](#)
 Conexão Baseada em Certificado [865](#)
 APNs [865](#)
 Conexões Seguras [111](#)
 Conexões TCP [136](#)
 Configurações do WebView2 [1010](#)
 Configure Install [55](#)
 Configure ZLib [63](#)
 Connect Mosquitto [302](#)
 Connect Secure Server [156](#)
 Connect TCP Server [153](#)
 Connect WebSocket Server [147](#)
 Connections TIME_WAIT [154](#)
 Consume [339](#)
 Consumer [339](#)
 Consumer Group [340](#)

Conta Forex [589](#)
 Coturn [1142](#)
 Crawler Detection [247](#)
 CreateTopic [341](#)
 Credenciais Amazon AWS do OpenAPI [1083](#)
 Criar Permissões do Cliente TURN [1031](#)
 Cross-Site Scripting [241](#)
 CryptoHopper [677](#)
 Custom Objects [123](#)
 Custom Sub [108](#)
 Datacenter Detection [247](#)
 Datasnap [1051](#)
 DeepSeek [793](#)
 DeepSeek Messages [796](#)
 DeepSeek Models [798](#)
 DeepSeek Vision [797](#)
 Deflate-Frame [689](#)
 DeleteTopic [341](#)
 Desconexão do Cliente Kafka [337](#)
 Device Authorization Grant (RFC 8628) [888](#),
[915](#)
 DNSBL [247](#)
 Documentos Anthropic [777](#)
 DPoP - Demonstrating Proof of Possession
 (RFC 9449) [890](#)
 DPoP Validation (RFC 9449) [921](#)
 Dropped Disconnections [152](#)
 Edições [28](#)
 Embeddings ChatBot [758](#)
 Embeddings Criar Vetores [757](#)
 Embeddings do Mistral [828](#)
 Encaminhar Requisições HTTP [129](#)
 Endpoints do Servidor [902](#)
 Enviar Arquivos [414](#)
 Clientes [414](#)
 Enviar Arquivos Grandes por POST [119](#)
 Enviar Arquivos para Clientes [414](#)
 EPOLL [127](#)
 Error [208](#)
 Estado da Conexão AMQP1 do Cliente [351](#)
 Extensions [687](#)
 FCrDNS [247](#)
 Ferramentas do servidor MCP [694](#)
 Files [142](#), [201](#), [413](#), [414](#), [415](#), [849](#), [1131](#)
 Fired [228](#)
 Firewall Blacklist [237](#)
 Firewall Whitelist [237](#)
 Flash [122](#)
 Flow [70](#)
 Threading [70](#)
 Forex Connect [586](#), [591](#)
 Forex Market Data [590](#)
 Forex Subscribe [592](#)
 Forex Trade [587](#)
 Found [670](#)
 Gemini [790](#)
 Gemini CountTokens [806](#)
 Gemini EmbedContent [807](#)
 Gemini Embeddings [807](#)
 Gemini Function Calling [808](#)
 Gemini Models [804](#)
 Gemini Structured Outputs [805](#)
 Gemini Token Counting [806](#)
 Gemini Tool Use [808](#)
 Gemini Vision [803](#)
 Generate [862](#)
 Remote Notification APNs [862](#)
 Gerenciador de Memória [82](#)
 GetCommittedOffset [342](#)
 GetEarliestOffset [342](#)
 GetLatestOffset [342](#)
 GetMetadata [341](#)
 Google Calendar [985](#), [992](#), [993](#), [994](#)
 Google Calendar RefreshToken [994](#)
 Google Calendar Service Account [995](#)
 Google Calendar Sync Calendars [992](#)
 Google Calendar Sync Events [993](#)
 Google Cloud FCM [996](#)
 Google Cloud Pub [976](#)
 Google OAuth2 Keys [966](#)
 Google Service Accounts [972](#)
 Grok [816](#)
 Grok Messages [819](#)
 Grok Models [821](#)
 Grok Vision [820](#)
 GroupId [340](#)
 Groups [124](#)
 HandShake do Cliente WebSocket [170](#)
 HeartBeat [113](#)
 HighPerfAcceptsPerWorker [219](#)
 Histórico [589](#)

- Horário das Requisições Privadas Kucoin [549](#)
- HTTP [68](#), [116](#), [129](#), [200](#), [201](#), [208](#), [209](#), [223](#), [843](#), [847](#)
- HTTP Dispatch Files [201](#)
- HTTP Server Stream Video [211](#)
- HTTP/2 [202](#), [203](#), [205](#), [848](#), [849](#), [850](#), [851](#), [854](#), [857](#)
- HTTP/2 Alternate Service [205](#)
- HTTP/2 Download File [849](#)
- HTTP/2 Headers [851](#)
- HTTP/2 Partial Responses [850](#)
- HTTP/2 Reason Disconnection [854](#)
- HTTP/2 Server Push [203](#), [848](#)
- HTTP1 [867](#)
- HTTP2 [844](#)
- HTTPAPI [221](#), [223](#), [224](#), [228](#)
- HTTPAPI Custom Headers [224](#)
- HTTPAPI Disable HTTP [223](#)
- HTTPAPI FineTune [219](#)
- HTTPAPI OnDisconnect [228](#)
- HTTPAPI Send File Response [226](#)
- HTTPAPI Send Text Response [225](#)
- HTTPAPI URL Reservation [217](#)
- ICE [1038](#)
- ICE Gather Candidates [1040](#)
- ICE Pair Candidates [1041](#)
- In HTML [901](#)
- Indy [101](#)
- Indy SChannel [183](#), [212](#)
- Inicialização e Encerramento do Servidor [179](#)
- Instalação [30](#)
- Instalar o Pacote sgclndy [57](#)
- Instalar pacote [44](#)
- Install Errors [51](#)
- Install Setup [33](#)
- Introdução [23](#)
- IOCP [126](#)
- IoT [829](#), [830](#), [838](#)
- IoT Amazon MQTT Client [830](#)
- IoT Azure MQTT Client [838](#)
- IP Ban [239](#)
- IP Filtering [237](#)
- JWT [930](#)
- Kafka [336](#), [337](#), [338](#), [339](#), [340](#), [341](#), [342](#)
- Kafka Consume Messages [339](#)
- Kafka Consumer Groups [340](#)
- Kafka Produce Messages [338](#)
- Kafka Topics [341](#)
- KendoUI_Grid [1134](#)
- Kucoin Connect WebSocket API [543](#)
- Kucoin Futures Conectar à WebSocket API [556](#)
- Kucoin Futures Get Market Data [558](#)
- Kucoin Futures Private Requests Time [563](#)
- Kucoin Futures Private REST API [559](#)
- Kucoin Futures Subscribe WebSocket Channel [557](#)
- Kucoin Futures Trade [560](#)
- Kucoin Get Market Data [545](#)
- Kucoin Private REST API [546](#)
- Kucoin Subscribe WebSocket Channel [544](#)
- Kucoin Trade Spot [547](#)
- License [1144](#)
- Lightstreamer [591](#), [592](#)
- Links AMQP1 [357](#)
- LoadBalancing [141](#)
- Logs [115](#)
- MaxConnectionsPerIP [244](#)
- MaxMessagesPerSec [244](#)
- MCP [690](#)
- MCP Client Elicitation [716](#)
- MCP Client Prompts [710](#)
- MCP Client Roots [714](#)
- MCP Client Sampling [715](#)
- MCP Client Tools [708](#)
- MCP Server Elicitation [705](#)
- MCP Server Prompts [697](#)
- MCP Server Resources [700](#)
- MCP Server Roots [703](#)
- MCP Server Sampling [704](#)
- MCP Server Sessions [693](#)
- Mensagens do Gemini [801](#)
- Mensagens Fragmentadas [144](#)
- Mensagens Patrocinadas do Telegram [671](#)
- Message Filtering [241](#)
- Method [847](#)
- Mistral AI [822](#)
- Mistral Messages [825](#)
- Mistral Models [827](#)
- Mistral Vision [826](#)

[MQTT 301, 302, 303, 304, 305, 306, 308, 309, 310, 311, 312, 830, 838, 1123](#)
[MQTT Clear Retained Messages 312](#)
[MQTT Publish 305, 309, 311](#)
[MQTT Publish Message 309](#)
[MQTT Publish Subscribe 305](#)
[MQTT Receive Messages 310](#)
[MQTT Subscribe 308](#)
[MQTT Topics 306](#)
[Notification Requests 863](#)
 [Sending 863](#)
[OAuth2 857, 871, 897, 901, 902, 903, 904, 905, 906, 966](#)
[OAuth2 Client for Desktop Applications 877](#)
[OAuth2 Client for Web Applications 876](#)
[OAuth2 Customize Sign 901](#)
[OAuth2 None Authenticate URLs 906](#)
[OAuth2 Provider Authentication 927](#)
[OAuth2 Provider Azure AD 925](#)
[OAuth2 Provider Private Endpoints 926](#)
[OAuth2 Provider Requests 929](#)
[OAuth2 Recover Access Tokens 904](#)
[OAuth2 Register Apps 903](#)
[Offsets do Kafka 342](#)
[Ollama 810](#)
[Ollama Embeddings 815](#)
[Ollama Messages 813](#)
[Ollama Models 814](#)
[ompClassic 219](#)
[ompHighPerf 219](#)
[OpenAI 717](#)
[OpenAI Applications 736](#)
[OpenAI Audio 728, 745](#)
[OpenAI Batch 734](#)
[OpenAI Chat 726](#)
[OpenAI Completion 725](#)
[OpenAI Edit 727](#)
[OpenAI Fine-Tuning 733](#)
[OpenAI Moderation 729](#)
[OpenAI RealTime 730](#)
[OpenAI Responses 731](#)
[OpenAI Speech 732](#)
[OpenAI Uploads 735](#)
[OpenAPI 1056](#)
[OpenAPI Additional Properties 1063](#)
[OpenAPI Amazon AWS 1077](#)
[OpenAPI Amazon AWS S3 1085](#)
[OpenAPI API 1069](#)
[OpenAPI Client 1070](#)
[OpenAPI Command Line 1065](#)
[OpenAPI Google Cloud 1086](#)
[OpenAPI Google Cloud Calendar 1100](#)
[OpenAPI Google Cloud OAuth2 1091](#)
[OpenAPI Google Cloud PubSub 1099](#)
[OpenAPI Google Cloud Service Accounts 1094](#)
[OpenAPI Google Drive 1101](#)
[OpenAPI Library 1067](#)
[OpenAPI Microsoft 1103](#)
[OpenAPI Microsoft Graph 1116](#)
[OpenAPI Microsoft OAuth2 Code 1112](#)
[OpenAPI Microsoft OAuth2 Credentials 1114](#)
[OpenAPI Microsoft Register Application 1109](#)
[OpenAPI Microsoft Tenant 1108](#)
[OpenAPI Parser Pascal 1058](#)
[OpenAPI Server 1073](#)
[OpenAPI Server Code-First 1076](#)
[OpenAPI Server Spec-First 1075](#)
[OpenSSL 85, 88, 90, 92, 93, 157](#)
[OpenSSL Android 92](#)
[OpenSSL iOS 93](#)
[OpenSSL Load Additional Functions 99](#)
[OpenSSL OSX 90](#)
[OpenSSL Own CA Certificates 95](#)
[OpenSSL P12 Certificates 97](#)
[OpenSSL Verify Certificate 98](#)
[OpenSSL Windows 88](#)
[OperatingMode 219](#)
[Ordens 589](#)
[Orders 587](#)
[PDF-Back-Cover 1155](#)
[PDF-Front-Cover 1](#)
[PerMessage-Deflate 688](#)
[Pinecone 759](#)
[Poll 339](#)
[Posições 589](#)
[Price Bars 590](#)
[Price Ticks 590](#)
[Produce 338](#)
[Producer 338](#)
[Proteção contra Flood 244](#)

- Protocol AMQP [313](#)
- Protocol Dataset [400](#), [404](#), [407](#), [408](#)
- Protocol Dataset Javascript [404](#)
- Protocol Dataset Notify Updates [408](#)
- Protocol Dataset Replicate Table [407](#)
- Protocol Default [391](#), [396](#)
- Protocol Default Javascript [396](#)
- Protocol E2EE [423](#)
- Protocol MQTT [298](#)
- Protocol Presence [416](#), [420](#)
- Protocol Presence Javascript [420](#)
- Protocol STOMP [366](#)
- Protocol WAMP [375](#), [379](#)
- Protocol WAMP Javascript [379](#)
- Protocol WAMP2 [387](#)
- Protocol WebRTC [372](#)
- Protocolo AMQP1 [343](#)
- Protocolo AppRTC [370](#)
- Protocolo Files [409](#)
- Protocolo Kafka [333](#)
- Protocolo WebRTC [374](#)
- Protocolo WebRTC Javascript [374](#)
- Protocolos [293](#), [333](#), [370](#), [374](#), [409](#)
- Protocols [108](#), [171](#), [295](#), [298](#), [313](#), [366](#), [372](#), [375](#), [379](#), [387](#), [391](#), [396](#), [400](#), [404](#), [407](#), [408](#), [416](#), [420](#)
- Protocols Javascript [295](#)
- Proxy [143](#), [172](#), [674](#)
- Quality [130](#)
 - Service [130](#)
- Quality Of Service [130](#)
- QueueLength [219](#)
- Queues [132](#), [323](#)
- QuickStart HTTP [68](#)
- QuickStart WebSockets [66](#)
- Rate Limiting [244](#)
- RCON [676](#)
- Receber mensagens binárias [196](#)
- Receber Mensagens Binárias [165](#)
- Receber mensagens de texto [164](#)
- Receive Text Messages [195](#)
- Recursos Avançados do WebView2 [1012](#)
- Recursos do Cliente MCP [712](#)
- Refresh Token Grant [913](#)
- Register [675](#)
- Register Telegram User [675](#)
- Remote Notification APNs [862](#)
 - Generate [862](#)
- Request HTTP [847](#)
- Requisição de Autenticação WebAuthn [950](#)
- Resource Owner Password Credentials Grant [911](#)
- Resource Owner Password Credentials Grant (RFC 6749) [886](#)
- Response Body [208](#)
- REST API [586](#)
- Resultado da autenticação WebAuthn [952](#)
- Resultados Progressivos de RPC WAMP [385](#)
- Reverse DNS [247](#)
- RTCMultiConnection [682](#)
- RTCPeerConnection Data [1050](#)
- RTCPeerConnection DTLS [1049](#)
- RTCPeerConnection ICE [1048](#)
- RTCPeerConnection Signaling [1047](#)
- RTCPeerConnection STUN TURN [1046](#)
- RTCPeerConnection WebSocket Client [1045](#)
- RTCPeerConnection WebSocket Server [1044](#)
- SChannel [183](#), [212](#)
- SChannel Get Connection Info [160](#)
- SChannel Server [183](#), [212](#)
- Send Big Files [415](#)
- Send Files [413](#)
 - Server [413](#)
- Send Files To Server [413](#)
- Send Telegram Invoice Message [672](#)
- Send Telegram Message Bold [669](#)
- Send Telegram Message With Buttons [667](#), [668](#)
- Send Telegram Message With Inline Buttons [667](#)
- Sending [863](#)
 - Notification Requests [863](#)
- Server [177](#), [178](#), [180](#), [181](#), [189](#), [193](#), [194](#), [195](#), [197](#), [200](#), [221](#), [302](#), [413](#), [897](#), [1025](#), [1036](#), [1037](#), [1119](#), [1126](#), [1129](#), [1133](#), [1137](#)
 - Send Files [413](#)
- Server Authentication [1133](#)
- Server Bindings [178](#)
- Server Chat [1119](#)
- Server Close Connection [189](#)
- Server Example [897](#)
- Server Keep Active [180](#)

- Server Monitor [1126](#)
- Server Read Headers [197](#)
- Server Requests [200](#)
- Server Send Binary Message [194](#)
- Server Send Text Message [193](#)
- Server Snapshots [1129](#)
- Server SSL [181](#), [221](#)
- Server SSL SChannel [183](#), [212](#)
- Server Start [177](#)
- Server Verify Certificate [186](#)
- Server-Sent Events [139](#)
- ServerSentEvents [1136](#)
- Service [130](#), [205](#), [972](#)
 - Quality [130](#)
- Servidor [179](#), [187](#), [188](#), [191](#), [196](#), [202](#), [209](#), [848](#), [902](#), [905](#), [1026](#), [1138](#)
- Servidor AppRTC [1138](#)
- Servidor de Alto Desempenho [78](#)
- Servidor manter conexões ativas [187](#)
- Servidor TCP Simples [188](#)
- Sessões AMQP1 [355](#)
- Sessões do Servidor [209](#)
- SkipIOCPOnSuccess [219](#)
- SQL Injection [241](#)
- Stop Limit [587](#)
- STUN [1018](#), [1021](#), [1022](#), [1023](#), [1025](#), [1026](#)
- STUN Client Attributes [1023](#)
- STUN Client Long Term Credentials [1022](#)
- STUN Client UDP Retransmissions [1021](#)
- STUN Server Long Term Credentials [1025](#)
- STUN Servidor Alternate Server [1026](#)
- Sub [976](#)
- SubProtocol [137](#)
- Subscribe [339](#)
- Telegram Chat [670](#)
- Telegram Client [1140](#)
- Telegram Get SuperGroup Members [673](#)
- Teste no navegador web [107](#)
- Threading [70](#)
 - Flow [70](#)
- Threads do Servidor HTTP/2 [206](#)
- Throttle [138](#)
- Token Introspection (RFC 7662) [919](#)
- Token Revocation (RFC 7009) [917](#)
- Token-Based Connection [864](#)
 - APNs [864](#)
- Transactions [134](#)
- TsgcAIChat [751](#)
- TsgcAIDatabaseVectorFile [755](#)
- TsgcAIDatabaseVectorPinecone [756](#)
- TsgcAIOpenAIAssistant [737](#)
- TsgcAIOpenAIAssistant File Search [739](#)
- TsgcAIOpenAIAssistant Function Calling [743](#)
- TsgcAIOpenAIAssistant Streaming [741](#)
- TsgcAIOpenAIChatBot [752](#)
- TsgcAIOpenAIEmbeddings [754](#)
- TsgcAIOpenAITranslator [753](#)
- TsgcAudioPlayerMCI [747](#)
- TsgcAudioRecorderMCI [746](#)
- TsgcBotClassification [247](#)
- TsgcHTTP_API_Anthropic [766](#)
- TsgcHTTP_API_DeepSeek [795](#)
- TsgcHTTP_API_Gemini [792](#)
- TsgcHTTP_API_Grok [818](#)
- TsgcHTTP_API_Mistral [824](#)
- TsgcHTTP_API_Ollama [812](#)
- TsgcHTTP_API_OpenAI [724](#)
- TsgcHTTP_JWT_Client [932](#)
- TsgcHTTP_JWT_Server [935](#)
- TsgcHTTP_OAuth2_Client [872](#)
- TsgcHTTP_OAuth2_Client_Google [878](#)
- TsgcHTTP_OAuth2_Client_Microsoft [879](#)
- TsgcHTTP_OAuth2_Server [894](#)
- TsgcHTTP_OAuth2_Server_Provider [923](#)
- TsgcHTTP2Client [845](#)
- TsgcHTTP2ConnectionClient [858](#)
- TsgcHTTP2RequestProperty [859](#)
- TsgcHTTP2ResponseProperty [860](#)
- TsgcICEClient [1039](#)
- TsgcIWWebSocketClient [289](#)
- TsgcIWWSPClient_Dataset [403](#)
- TsgcIWWSPClient_sgc [395](#)
- TsgcRTCPeerConnection [1042](#)
- TsgcServerHTTPAPI_FineTune [219](#)
- TsgcSTUNClient [1019](#)
- TsgcSTUNServer [1024](#)
- TsgcTCPClient [173](#)
- TsgcTextToSpeechAmazon [750](#)
- TsgcTextToSpeechGoogle [749](#)
- TsgcTextToSpeechSystem [748](#)
- TsgcTURNClient [1028](#)
- TsgcTURNServer [1034](#)

TsgcUDPClient [1016](#)
 TsgcUDPServer [1017](#)
 TsgcWebPush_Client [686](#)
 TsgcWebSocketClient [145](#)
 TsgcWebSocketClient_WinHTTP [229](#)
 TsgcWebSocketFirewall [230](#), [237](#), [239](#), [241](#),
[244](#), [247](#)
 TsgcWebSocketHTTPServer [198](#)
 TsgcWebSocketHTTPServer_Sessions [209](#)
 TsgcWebSocketLoadBalancerServer [286](#)
 TsgcWebSocketProxyServer [288](#)
 TsgcWebSocketServer [175](#)
 TsgcWebSocketServer_HTTPAPI [215](#)
 TsgcWebView2 [999](#)
 TsgcWSAPIClient_MCP [707](#)
 TsgcWSAPIKeyManager [273](#)
 TsgcWSAPIServer_MCP [691](#)
 TsgcWSAPIServer_WebAuthn [937](#)
 TsgcWSAPIServer_WebPush [685](#)
 TsgcWSCircuitBreaker [262](#)
 TsgcWSConnection [291](#)
 TsgcWSHTTP2WebBrokerBridgeServer [1054](#)
 TsgcWSHTTPWebBrokerBridgeServer [1052](#)
 TsgcWSMessageFile [412](#)
 TsgcWSPClient_AMQP [314](#)
 TsgcWSPClient_AMQP1 [345](#)
 TsgcWSPClient_Dataset [402](#)
 TsgcWSPClient_E2EE [426](#)
 TsgcWSPClient_Files [411](#)
 TsgcWSPClient_Kafka [334](#)
 TsgcWSPClient_MQTT [300](#)
 TsgcWSPClient_Presence [419](#)
 TsgcWSPClient_sgc [394](#)
 TsgcWSPClient_STOMP [367](#)
 TsgcWSPClient_STOMP_ActiveMQ [369](#)
 TsgcWSPClient_STOMP_RabbitMQ [368](#)
 TsgcWSPClient_WAMP [378](#)
 TsgcWSPClient_WAMP2 [388](#)
 TsgcWSPPresenceMessage [418](#)
 TsgcWSPServer_AppRTC [371](#)
 TsgcWSPServer_Dataset [401](#)
 TsgcWSPServer_E2EE [425](#)
 TsgcWSPServer_Files [410](#)
 TsgcWSPServer_Presence [417](#)
 TsgcWSPServer_sgc [393](#)
 TsgcWSPServer_WAMP [377](#)
 TsgcWSPServer_WebRTC [373](#)
 TsgcWSRateLimiter [250](#)
 TsgcWSServer_HTTPAPI_WebBrokerBridge
[1055](#)
 TURN [1027](#), [1030](#), [1031](#), [1032](#), [1033](#), [1036](#),
[1037](#)
 TURN Client Allocate IP Address [1030](#)
 TURN Client Send Indication [1032](#)
 TURN Server Allocations [1037](#)
 TURN Server Long Term Credentials [1036](#)
 Upload File [1131](#)
 Usando DLL [105](#)
 Visão geral [64](#)
 Wait Response [311](#)
 WAMP [382](#), [383](#), [384](#), [385](#)
 WAMP Publishers [383](#)
 WAMP Simple RPC [384](#)
 WAMP Subscribers [382](#)
 WatchDog [114](#)
 WebAuthn [936](#)
 WebAuthn Authentication [948](#)
 WebAuthn Authentication Response [951](#)
 WebAuthn Authorization [954](#)
 WebAuthn Authorization HTTP [955](#)
 WebAuthn Authorization WebSocket [956](#)
 Webauthn Javascript Client [957](#)
 WebAuthn MDS [953](#)
 WebAuthn Registration [941](#)
 WebAuthn Registration Request [944](#)
 WebAuthn Registration Response [945](#)
 WebAuthn Registration Result [947](#)
 WebPush [684](#)
 WebRTC [1137](#)
 WebSocket Events [103](#)
 WebSocket Parameters Connection [104](#)
 WebSocket Redirections [155](#)
 WebSockets [66](#), [103](#), [104](#), [147](#), [155](#), [170](#), [437](#),
[438](#), [460](#), [461](#), [478](#), [484](#), [494](#), [501](#), [591](#), [592](#)
 WebSockets Private [484](#), [501](#)
 WebSockets Public [478](#), [494](#)
 WebView2 Cookies [1006](#)
 WebView2 Downloads [1008](#)
 WebView2 JavaScript [1004](#)
 WebView2 Navigation [1002](#)
 WhatsApp Create App [637](#)
 WhatsApp Download Media [657](#)

WhatsApp Enviar mensagens interativas [646](#)

WhatsApp Phone Number Id [639](#)

WhatsApp Receive Messages and Status Notifications [653](#)

WhatsApp Security [642](#)

WhatsApp Send Files [655](#)

WhatsApp Send Messages [643](#)

WhatsApp Send Template Messages [651](#)

WhatsApp Token [640](#)

WhatsApp Webhook [641](#)

Windows Service [106](#)

XSS [241](#)

