



sgcWebSockets 2026.6

June 2026

CBuilder 문서

Copyright © 2012-2026 eSeGeCe Software

info@esegece.com

www.esegece.com

목차

소개	23
개요	27
에디션	27
설치	29
설치	32
Install Setup	32
패키지 설치	43
Install Errors	50
Configure Install	54
sgcIndy 패키지 설치	56
Configure ZLib	62
빠른 시작	63
개요	63
QuickStart WebSockets	65
QuickStart HTTP	67
스레딩 흐름	69
Build	71
Build OSX Application	72
Build Android Application	74
iOS 애플리케이션 빌드	75
Fast Performance Server	77
Memory Manager	80
OpenSSL	83
OpenSSL Windows	86
OpenSSL OSX	88

OpenSSL Android	90
OpenSSL iOS	91
OpenSSL Own CA Certificates.....	93
OpenSSL P12 Certificates	95
OpenSSL Verify Certificate	96
OpenSSL Load Additional Functions.....	97
Indy	99
주제	101
WebSocket 이벤트	101
WebSocket Parameters Connection	102
DLL 내부에서 사용.....	103
Windows Service	104
Web Browser Test	105
사용자 지정 서브프로토콜	106
Authentication	107
Secure Connections	109
HeartBeat	111
WatchDog.....	112
Logs.....	113
HTTP.....	114
브로드캐스트와 채널	115
Bindings.....	116
대용량 파일 전송(POST)	117
압축.....	119
Flash.....	120
Custom Objects	121
Groups.....	122
IOCP	124
EPOLL.....	125
ALPN	126

HTTP 요청 전달	127
Quality Of Service	128
Queues	130
Transactions	132
TCP Connections	134
SubProtocol	135
Throttle	136
Server-Sent Events	137
LoadBalancing	139
Files	140
Proxy	141
Fragmented Messages	142
구성 요소.....	143
TsgcWebSocketClient.....	143
Connect WebSocket Server	145
Client Open Connection	146
클라이언트가 연결 종료.....	148
Client Keep Connection Open	149
끊긴 연결	150
Connect TCP Server	151
Connections TIME_WAIT	152
WebSocket Redirections.....	153
Connect Secure Server	154
Certificates OpenSSL.....	155
Certificates SChannel.....	156
SChannel Get Connection Info	157
Client Send Text Message	158
Client Send Binary Message.....	159
클라이언트가 텍스트 및 바이너리 메시지 전송	160
Receive Text Messages	161

Receive Binary Messages	162
클라이언트 인증	163
클라이언트 예외	166
클라이언트 WebSocket HandShake	167
클라이언트 Register Protocol.....	168
Client Proxies	169
TsgcTCPClient	170
TsgcTCPClient	171
TsgcWebSocketServer	172
서버 시작	174
Server Bindings.....	175
Server Startup Shutdown	176
Server Keep Active	177
Server SSL.....	178
Server SSL SChannel	180
Server Verify Certificate.....	183
Server Keep Connections Alive.....	184
Server Plain TCP	185
서버 연결 종료	186
클라이언트 연결.....	187
Server Authentication.....	188
Server Send Text Message	190
Server Send Binary Message	191
Server Receive Text Message.....	192
Server Receive Binary Message	193
Server Read Headers from Client.....	194
TsgcWebSocketHTTPServer	195
HTTP Server Requests	197
HTTP Dispatch Files.....	198
HTTP/2 Server	199
HTTP/2 Server Push	200

HTTP/2 Alternate Service.....	202
HTTP/2 Server Threads.....	203
응답 본문 없는 HTTP 404 오류	205
HTTP Server Sessions	206
HTTP Server Stream Video	208
Server SSL SChannel	209
TsgcWebSocketServer_HTTPAPI	212
HTTPAPI URL Reservation	214
HTTPAPI FineTune	216
HTTPAPI Server SSL.....	218
자체 서명 인증서	219
HTTPAPI Disable HTTP/2	220
HTTPAPI Custom Headers.....	221
HTTPAPI Send Text Response.....	222
HTTPAPI Send File Response	223
HTTPAPI OnDisconnect가 발생하지 않음	225
TsgcWebSocketClient_WinHTTP	226
TsgcWebSocketFirewall	227
Firewall: 블랙리스트 및 화이트리스트	234
Firewall: Brute Force Protection	236
Firewall: SQL Injection and XSS Detection.....	238
Firewall: Rate Limiting and Flood Protection	241
Firewall: Bot Detection	244
TsgcWSRateLimiter	247
TsgcWSCircuitBreaker.....	258
TsgcWSAPIKeyManager	268
TsgcWebSocketLoadBalancerServer.....	279
TsgcWebSocketProxyServer.....	280
TsgcIWWebSocketClient	281
TsgcWSConnection.....	283
프로토콜	285

Protocols Javascript.....	287
Protocol MQTT.....	290
TsgcWSPClient_MQTT	292
클라이언트 MQTT Connect.....	293
Connect Mosquitto MQTT Servers	294
Client MQTT Sessions	295
Client MQTT Version	296
MQTT Publish Subscribe	297
MQTT Topics	298
MQTT Subscribe	299
MQTT Publish Message	300
MQTT Receive Messages	301
MQTT Publish and Wait Response	302
MQTT Clear Retained Messages.....	303
Protocol AMQP	304
TsgcWSPClient_AMQP.....	305
Client AMQP Connect	308
Client AMQP Disconnect.....	309
AMQP 채널	310
AMQP Exchanges	312
AMQP Queues	314
AMQP Publish Messages.....	317
AMQP Consume Messages	318
AMQP Get Messages.....	320
AMQP QoS.....	321
AMQP Transactions.....	322
Protocol Kafka	323
TsgcWSPClient_Kafka.....	324
Client Kafka Connect.....	326
Client Kafka Disconnect.....	327
Kafka Produce Messages	328

Kafka Consume Messages	329
Kafka Consumer Groups	330
Kafka Topics	331
Kafka Offsets.....	332
Protocol AMQP1	333
TsgcWSPClient_AMQP1	334
클라이언트 AMQP1 연결.....	337
Client AMQP1 Disconnect	338
Client AMQP1 Idle Timeout Connection.....	339
클라이언트 AMQP1 연결 상태.....	340
Client AMQP1 Authentication	341
Client AMQP1 Azure MessageBus.....	342
AMQP1 Sessions.....	344
AMQP1 Links.....	346
AMQP1 Sender Links	347
AMQP1 Receiver Links	349
AMQP1 Send Message.....	351
AMQP1 Read Message.....	353
Protocol STOMP	354
TsgcWSPClient_STOMP	355
TsgcWSPClient_STOMP_RabbitMQ.....	356
TsgcWSPClient_STOMP_ActiveMQ	357
프로토콜 AppRTC	358
TsgcWSPServer_AppRTC.....	359
프로토콜 WebRTC	360
TsgcWSPServer_WebRTC.....	361
Protocol WebRTC Javascript.....	362
Protocol WAMP.....	363
TsgcWSPServer_WAMP	364
TsgcWSPClient_WAMP	365
Protocol WAMP Javascript.....	366

구독자	369
게시자	370
간단한 RPC	371
RPC 진행 결과	372
Protocol WAMP2	374
TsgcWSPClient_WAMP2	375
Protocol Default	377
TsgcWSPServer_sgc.....	378
TsgcWSPClient_sgc	379
TsgcIWWSPClient_sgc	380
Protocol Default Javascript.....	381
Protocol Dataset.....	385
TsgcWSPServer_Dataset	386
TsgcWSPClient_Dataset	387
TsgcIWWSPClient_Dataset.....	388
Protocol Dataset Javascript.....	389
Protocol Dataset Replicate Table	392
Protocol Dataset Notify Updates.....	393
Protocol Files	394
TsgcWSPServer_Files.....	395
TsgcWSPClient_Files.....	396
TsgcWSMessageFile	397
How Send Files To Server	398
파일을 클라이언트에 보내는 방법	399
큰 파일을 보내는 방법	400
Protocol Presence	401
TsgcWSPServer_Presence	402
TsgcWSPPresenceMessage	403
TsgcWSPClient_Presence.....	404
Protocol Presence Javascript	405
Protocol E2EE.....	408

TsgcWSPServer_E2EE	410
TsgcWSPClient_E2EE	411
APIs	412
API Binance	414
Binance Connect WebSocket API	421
Binance WebSocket 채널 구독	422
Binance Get Market Data	423
Binance Private REST API.....	424
Binance Trade Spot.....	425
Binance Private Requests Time	427
Binance Withdraw	428
API Binance Futures	429
API Binance Futures Trade	435
API SocketIO.....	436
API Coinbase	438
Coinbase WebSocket API 연결.....	442
Coinbase WebSocket 채널 구독.....	443
Coinbase 시장 데이터 가져오기	444
Coinbase 비공개 REST API	445
Coinbase 비공개 요청 시간.....	446
Coinbase 주문하기	447
Coinbase 샌드박스 계정	448
API SignalRCore	449
API SignalR	455
API Kraken.....	458
API Kraken WebSockets Public	460
API Kraken WebSockets Private.....	466
API Kraken REST Public.....	469
API Kraken REST Private	471
API Kraken Futures.....	474
API Kraken Futures WebSockets Public.....	476

API Kraken Futures WebSockets Private	483
API Kraken Futures REST Public	489
API Kraken Futures REST Private	491
API Pusher	496
API Bitmex	504
Bitmex Connect WebSocket API	508
Bitmex Subscribe WebSocket Channel	509
How to Place a Bitmex Order	510
API Bitfinex	512
API Kucoin	516
Kucoin Connect WebSocket API	522
Kucoin Subscribe WebSocket Channel	523
Kucoin Get Market Data	524
Kucoin Private REST API	525
Kucoin Trade Spot	526
Kucoin Private Requests Time	528
API Kucoin Futures	529
Kucoin Futures Connect WebSocket API	534
Kucoin Futures Subscribe WebSocket Channel	535
Kucoin Futures Get Market Data	536
Kucoin Futures Private REST API	537
Kucoin Futures Trade	538
Kucoin Futures Private Requests Time	541
API 3Commas	542
API OKX	546
API XTB	552
API Bybit	555
API Forex	559
Forex Connect REST API	561
Forex Trade Orders	562
Forex Account Queries	564

Forex 시장 데이터	565
Forex Connect WebSocket API	566
Forex Subscribe Streams	567
API Blockchain	569
API Cex.....	571
API Cex Plus	578
API Discord.....	582
API OpenAI	585
API MEXC	587
API MEXC Futures.....	591
API Bitget.....	594
API GateIO	596
API Deribit	598
API Crypto.com.....	601
API HTX	603
API Whatsapp	605
WhatsApp Create App	608
WhatsApp Phone Number Id.....	609
WhatsApp Token	610
WhatsApp Webhook	611
WhatsApp Security.....	612
WhatsApp Send Messages.....	613
WhatsApp 대화형 메시지 보내기	616
WhatsApp Send Template Messages.....	620
WhatsApp Receive Messages and Status Notifications.....	622
WhatsApp Send Files	624
WhatsApp Download Media	626
API Telegram.....	627
Send Telegram Message With Inline Buttons.....	635
Send Telegram Message With Buttons.....	636
Send Telegram Message Bold	637

Telegram Chat을 Bot으로 찾을 수 없음	638
Telegram Sponsored Messages	639
Send Telegram Invoice Message	640
Telegram Get SuperGroup Members	641
Telegram 프록시 추가	642
Register Telegram User	643
RCON	644
CryptoHopper	645
RTCMultiConnection	650
WebPush	652
TsgcWSAPIServer_WebPush	653
TsgcWebPush_Client.....	654
Extensions	655
PerMessage-Deflate	656
Deflate-Frame	657
MCP	658
TsgcWSAPIServer_MCP	659
MCP Server Sessions.....	661
MCP Server Tools	662
MCP Server Prompts.....	665
MCP Server Resources.....	668
MCP Server Roots.....	671
MCP Server Sampling	672
MCP Server Elicitation.....	673
TsgcWSAPIClient_MCP	674
MCP Client Tools	675
MCP Client Prompts.....	677
MCP 클라이언트 리소스	679
MCP Client Roots.....	681
MCP Client Sampling.....	682
MCP Client Elicitation.....	683

OpenAI.....	684
TsgcHTTP_API_OpenAI.....	691
OpenAI Completion	692
OpenAI Chat.....	693
OpenAI Edit.....	694
OpenAI Audio.....	695
OpenAI Moderation	696
OpenAI RealTime.....	697
OpenAI Responses	698
OpenAI Speech	699
OpenAI Fine-Tuning	700
OpenAI Batch.....	701
OpenAI Uploads	702
OpenAI Applications	703
TsgcAIOpenAIAssistant.....	704
TsgcAIOpenAIAssistant File Search	706
TsgcAIOpenAIAssistant Streaming.....	708
TsgcAIOpenAIAssistant Function Calling.....	710
OpenAI Audio.....	712
TsgcAudioRecorderMCI	713
TsgcAudioPlayerMCI	714
TsgcTextToSpeechSystem.....	715
TsgcTextToSpeechGoogle	716
TsgcTextToSpeechAmazon	717
TsgcAIChat.....	718
TsgcAIOpenAIChatBot	719
TsgcAIOpenAITranslator.....	720
TsgcAIOpenAIEmbeddings.....	721
TsgcAIDatabaseVectorFile.....	722
TsgcAIDatabaseVectorPinecone.....	723
Embeddings Create Vectors.....	724

Embeddings ChatBot	725
Pinecone	726
Anthropic	729
TsgcHTTP_API_Anthropic	733
Anthropic Messages	734
Anthropic Vision	736
Anthropic Tool Use	738
Anthropic Models	740
Anthropic Batches	741
Anthropic Extended Thinking	742
Anthropic Documents	744
Anthropic Prompt Caching	746
Anthropic Citations	748
Anthropic Web Search	749
Anthropic Structured Outputs	751
Anthropic Files	753
Anthropic MCP Connector	755
Gemini	757
TsgcHTTP_API_Gemini	759
DeepSeek	760
TsgcHTTP_API_DeepSeek	761
DeepSeek Messages	762
DeepSeek Vision	763
DeepSeek Models	764
Gemini	765
Gemini Messages	767
Gemini Vision	769
Gemini Models	770
Gemini Structured Outputs	771
Gemini Token Counting	772
Gemini Embeddings	773

Gemini Tool Use	774
Ollama	776
TsgcHTTP_API_Ollama	778
Ollama Messages	779
Ollama Models.....	780
Ollama Embeddings.....	781
Grok	782
TsgcHTTP_API_Grok	783
Grok Messages	784
Grok Vision.....	785
Grok Models.....	786
Mistral AI	787
TsgcHTTP_API_Mistral.....	789
Mistral Messages.....	790
Mistral Vision	791
Mistral Models	792
Mistral Embeddings	793
IoT	794
IoT Amazon MQTT Client.....	795
IoT Azure MQTT Client	802
HTTP.....	807
HTTP2	808
TsgcHTTP2Client.....	809
Request HTTP/2 Method	811
HTTP/2 Server Push	812
HTTP/2 Download File	813
HTTP/2 부분 응답	814
HTTP/2 Headers	815
클라이언트가 연결 종료.....	816
클라이언트 연결 활성 유지	817
HTTP/2 Reason Disconnection	818

Client Pending Requests.....	819
클라이언트 인증.....	820
HTTP/2 and OAuth2	821
TsgcHTTP2ConnectionClient.....	822
TsgcHTTP2RequestProperty	823
TsgcHTTP2ResponseProperty.....	824
Apple Push Notifications	825
APNs 원격 알림 생성	826
APNs에 Notification Requests 보내기	827
Token-Based Connection to APNs	828
APNs에 대한 Certificate-Based Connection.....	829
HTTP1	831
OAuth2	835
TsgcHTTP_OAuth2_Client	836
OAuth2 Client for Web Applications	840
OAuth2 Client for Desktop Applications.....	841
TsgcHTTP_OAuth2_Client_Google	842
TsgcHTTP_OAuth2_Client_Microsoft.....	843
Authorization Code Grant (RFC 6749).....	844
Authorization Code with PKCE (RFC 7636)	846
Client Credentials Grant (RFC 6749).....	848
Resource Owner Password Credentials Grant (RFC 6749).....	850
Device Authorization Grant (RFC 8628)	852
DPoP - Demonstrating Proof of Possession (RFC 9449).....	854
TsgcHTTP_OAuth2_Server	858
OAuth2 Server 예제	861
OAuth2 Customize Sign-In HTML	865
OAuth2 Server Endpoints.....	866
OAuth2 Register Apps	867
OAuth2 Recover Access Tokens	868
OAuth2 Server Authentication.....	869

OAuth2 None Authenticate URL.....	870
Authorization Code Grant	871
Client Credentials Grant	873
Resource Owner Password Credentials Grant	875
Refresh Token Grant.....	877
Device Authorization Grant (RFC 8628)	879
Token Revocation (RFC 7009)	881
Token Introspection (RFC 7662)	883
DPoP Validation (RFC 9449)	885
TsgcHTTP_OAuth2_Server_Provider	887
OAuth2 Provider Azure AD	889
OAuth2 Provider Private Endpoints	890
OAuth2 Provider Authentication	891
OAuth2 Provider Requests.....	893
JWT	894
TsgcHTTP_JWT_Client	896
TsgcHTTP_JWT_Server.....	899
WebAuthn	900
TsgcWSAPIServer_WebAuthn	901
WebAuthn Registration	904
WebAuthn Registration Request	907
WebAuthn Registration Response	908
WebAuthn 등록 결과.....	910
WebAuthn Authentication.....	911
WebAuthn Authentication Request	913
WebAuthn Authentication Response.....	914
WebAuthn 인증 결과.....	915
WebAuthn MDS	916
WebAuthn Authorization.....	917
WebAuthn Authorization HTTP	918
WebAuthn Authorization WebSocket	919

Webauthn Javascript Client.....	920
Amazon SQS	924
Google OAuth2 Keys	929
Google Service Accounts	935
Google Cloud Pub/Sub	939
Google Calendar	947
Google Calendar Sync Calendars	953
Google Calendar Sync Events	954
Google Calendar RefreshToken.....	955
Google Calendar Service Account	956
Google Cloud FCM.....	957
TsgcWebView2	960
WebView2 Navigation.....	962
WebView2 JavaScript.....	964
WebView2 Cookies	966
WebView2 Downloads	968
WebView2 Settings.....	970
WebView2 고급 기능	972
TsgcUDPCClient	975
TsgcUDPServer	976
STUN	977
TsgcSTUNClient	978
STUN Client UDP Retransmissions.....	980
STUN Client Long Term Credentials.....	981
STUN Client Attributes	982
TsgcSTUNServer	983
STUN 서버 장기 자격 증명	984
STUN Server Alternate Server.....	985
TURN.....	986
TsgcTURNClient	987
TURN Client Allocate IP Address.....	989

TURN Client Create Permissions	990
TURN Client Send Indication.....	991
TURN Client Channels.....	992
TsgcTURNServer	993
TURN Server Long Term Credentials	994
TURN Server Allocations.....	995
ICE	996
TsgcICEClient.....	997
ICE Gather Candidates.....	998
ICE Pair Candidates.....	999
TsgcRTCPeerConnection	1000
RTCPeerConnection WebSocket Server	1002
RTCPeerConnection WebSocket Client.....	1003
RTCPeerConnection STUN TURN	1004
RTCPeerConnection Signaling	1005
RTCPeerConnection ICE.....	1006
RTCPeerConnection DTLS	1007
RTCPeerConnection Data.....	1008
Datasnap	1009
TsgcWSHTTPWebBrokerBridgeServer	1010
TsgcWSHTTP2WebBrokerBridgeServer	1012
TsgcWSServer_HTTPAPI_WebBrokerBridge.....	1013
OpenAPI	1014
OpenAPI	1014
OpenAPI Parser Pascal	1016
OpenAPI Additional Properties.....	1021
OpenAPI Command Line	1023
OpenAPI Library	1025
OpenAPI API.....	1027
OpenAPI Client	1028

OpenAPI 서버.....	1031
OpenAPI Server Spec-First	1033
OpenAPI Server Code-First.....	1034
OpenAPI Amazon AWS	1035
OpenAPI Amazon AWS Credentials.....	1041
OpenAPI Amazon AWS S3	1043
OpenAPI Google Cloud	1044
OpenAPI Google Cloud OAuth2.....	1049
OpenAPI Google Cloud Service Accounts.....	1052
OpenAPI Google Cloud PubSub	1056
OpenAPI Google Cloud Calendar	1057
OpenAPI Google Drive	1058
OpenAPI Microsoft	1060
OpenAPI Microsoft Tenant.....	1065
OpenAPI Microsoft Register Application	1066
OpenAPI Microsoft OAuth2 Code.....	1069
OpenAPI Microsoft OAuth2 자격 증명	1071
OpenAPI Microsoft Graph	1073
APIs	1074
AbstractApi Geolocation.....	1075
데모	1076
Server Chat.....	1076
Client Chat.....	1078
클라이언트.....	1079
클라이언트 MQTT	1080
클라이언트 SocketIO	1082
Server Monitor.....	1083
서버 스냅샷.....	1086
Client Snapshots.....	1087
파일 업로드.....	1088

Server Authentication.....	1090
KendoUI_Grid.....	1091
ServerSentEvents	1093
Server WebRTC	1094
Server AppRTC	1095
Telegram Client	1097
서드파티.....	1099
Coturn.....	1099
라이선스.....	1101
License	1101
색인	1103
PDF-Back-Cover	1112

소개

sgcWebSockets는 실시간 통신, 메시징 및 AI 통합을 위한 전문가급 구성 요소 라이브러리입니다. WebSocket, HTTP/2, MQTT, AMQP, WebRTC, Server-Sent Events 및 30개 이상의 서드파티 API 커넥터의 프로덕션 준비 완료된 구현을 제공하며, 모두 일관되고 이벤트 기반의 구성 요소 아키텍처를 통해 액세스할 수 있습니다. 거래 플랫폼, IoT 게이트웨이, AI 기반 애플리케이션 또는 엔터프라이즈 메시징 시스템을 구축하든, sgcWebSockets는 필요한 네트워킹 기반을 제공합니다.

이 라이브러리는 **Delphi 7부터 Delphi 13까지, C++Builder, Lazarus/FreePascal** 및 **.NET(Framework 2.0+, .NET Core 1.0+, .NET 5-9, .NET Standard)**을 지원합니다. 애플리케이션은 VCL과 FireMonkey를 사용하여 단일 코드베이스에서 Windows, macOS, Linux, iOS 및 Android를 대상으로 할 수 있습니다.

Delphi / C++Builder / FreePascal Features

Core WebSocket

- **RFC 6455**를 구현하는 완전히 기능적인 **멀티스레드 WebSocket 서버**.
- **Text** 및 **Binary** 메시지 지원.
- PerMessage Deflate 확장(**RFC 7692**)을 사용한 **메시지 압축**입니다.
- 자동 재연결 및 연결 모니터링을 위한 내장 **WatchDog** 및 **HeartBeat**입니다.
- 여러 백엔드에 걸쳐 연결을 분산하기 위한 **로드 밸런싱** 서버입니다.
- 웹 브라우저가 모든 TCP 서버에 연결할 수 있게 하는 **Proxy Server** 구성 요소.
- **HTTP Proxy** 및 **SOCKS Proxy** 서버를 통한 클라이언트 연결.
- 이벤트: OnConnect, OnDisconnect, OnMessage, OnError, OnHandshake.

고성능 서버

- 고성능 Windows 서버를 위한 **IOCP**가 포함된 **Microsoft HTTP Server API(HTTP.SYS)**입니다.
- **IOCP**(Windows), **EPOLL**(Linux) 또는 기본 연결당 단일 스레드 모델이 있는 Indy 서버.
- 동일한 포트를 통한 **WebSocket** 및 **HTTP/2** 연결.
- 단일 포트에서 **DataSnap**, **HTTP/2** 및 **WebSocket**을 지원하는 **WebBroker Server**.
- **WinHTTP API** 기반의 클라이언트 WebSocket입니다.

Security and Authentication

- 서버 및 클라이언트 구성 요소를 위한 **SSL/TLS**. **OpenSSL 1.1.1** 및 **3.0**이 지원됩니다. 클라이언트는 Windows에서 **SChannel**을 지원합니다.
- **OAuth 2.0** 및 **JWT** (JSON Web Token) 클라이언트 및 서버 구성 요소입니다.
- passkey 기반 인증을 위한 **WebAuthn Server Protocol**입니다.
- **E2EE**(End-To-End Encryption) 서브프로토콜입니다.

AI 및 머신러닝

- **MCP Server** 및 **MCP Client** 구성 요소와 함께하는 **OpenAI** 및 **Claude** 통합입니다.
- 음성 명령 지원이 있는 **ChatBot** 및 **Translator** 애플리케이션 구성 요소.
- **Embeddings** 및 **Vector Database** 지원(파일 기반 및 Pinecone).
- **Text-To-Speech**는 시스템 기본값, Google Cloud, Amazon AWS를 통해 제공됩니다.

HTTP/2 Protocol

- 전체 **HTTP/2** 클라이언트 및 서버 구현입니다.
- HTTP를 통한 **Server-Sent Events**(푸시 알림)입니다.

Messaging Protocols

- WebSocket 및 일반 TCP를 통한 **MQTT 3.1.1** 및 **5.0**.
- RabbitMQ 브로커 지원과 함께 **AMQP 0.9.1** 및 **1.0**을 지원합니다.
- WebSocket 및 TCP를 통한 **STOMP**(ActiveMQ, RabbitMQ 브로커).
- **WAMP 1.0** 및 **2.0** 클라이언트 및 서버 프로토콜.
- **JSON-RPC 2.0** 서브프로토콜.

P2P 및 실시간 통신

- **WebRTC** 서버 프로토콜입니다.
- **STUN** 및 **TURN** 클라이언트 및 서버 구성 요소.
- NAT 순회를 위한 **ICE** 클라이언트.
- **UDP** 클라이언트 및 서버입니다.

API 통합

- **30개 이상의 내장 API 커넥터**: Binance, Coinbase, Kraken, Bitfinex, Discord, WhatsApp, Telegram 등.
- **SignalR** 및 **SignalR Core**, **Socket.IO** 및 **Pusher** 클라이언트입니다.
- 클라우드 서비스: **Amazon SQS**, **Google Cloud Pub/Sub**, **Google Calendar**.

크로스 플랫폼

- **VCL** 및 **FireMonkey** (Windows, macOS, iOS, Android, Linux).
- **C++Builder** 지원.
- **Lazarus / FreePascal** 지원.
- 브라우저 클라이언트를 위한 내장 **JavaScript 라이브러리**.

sgcWebSockets 라이브러리에는 다음 구성 요소가 포함되어 있습니다:

1 sgcWebSockets

- **TsgcWebSocketClient**: Indy 라이브러리 기반 WebSocket 클라이언트.
- **TsgcWebSocketServer**: Indy 라이브러리 기반 WebSocket 서버
- **TsgcWebSocketHTTPServer**: Indy 라이브러리 기반 WebSocket + HTTP 서버.
- **TsgcWebSocketServer_HTTPAPI**: HTTP.SYS Microsoft HTTP API 기반의 고성능 WebSocket + HTTP 서버.
- **TsgcWebSocketClient_WinHTTP**: WinHTTP 라이브러리 기반 WebSocket 클라이언트.
- **TsgcWebSocketFirewall**: IP 필터링, 무차별 대입 보호, rate limiting 및 위협 탐지 기능을 갖춘 WebSocket Server Firewall.
- **TsgcWebSocketLoadBalancerServer**: WebSocket 및 HTTP 프로토콜용 로드 밸런서 서버.
- **TsgcWebSocketProxyServer**: 웹 브라우저가 TCP 서버에 연결할 수 있도록 하는 WebSocket 프록시 서버.

2 sgcWebSocket APIs

- **TsgcWSAPI_Binance**: Binance Spot Client, WebSocket + REST API를 지원합니다.
- **TsgcWSAPI_Binance_Futures**: Binance Futures Client, WebSocket + REST API를 지원합니다.
- **TsgcWSAPI_SocketIO**: Socket.IO Client.
- **TsgcWSAPI_Coinbase**: Coinbase Pro Client, WebSocket + REST API를 지원합니다.
- **TsgcWSAPI_Bitmex**: Bitmex Client, WebSocket + REST API를 지원합니다.
- **TsgcWSAPI_SignalR**: SignalR WebSocket Client.
- **TsgcWSAPI_SignalRCore**: SignalRCore WebSocket Client.
- **TsgcWSAPI_Pusher**: Pusher WebSocket Client.
- **TsgcWSAPI_Kraken**: Kraken Client API, WebSocket 및 REST Api를 지원합니다.
- **TsgcWSAPI_Kraken_Futures**: Kraken Futures Client API, WebSocket 및 REST Api를 지원합니다.
- **TsgcWSAPI_Bitstamp**: Bitstamp WebSocket Client.
- **TsgcWSAPI_Cex**: Cex WebSocket Client.
- **TsgcWSAPI_Huobi**: Huobi WebSocket Client.
- **TsgcWSAPI_ThreeCommas**: ThreeCommas Client API.
- **TsgcWSAPI_Bitfinex**: Bitfinex WebSocket API.
- **TsgcWSAPI_Discord**: Discord WebSocket Client.
- **TsgcWSAPI_BlockChain**: BlockChain WebSocket Client.
- **TsgcWSAPI_Bybit**: Bybit WebSocket Client.
- **TsgcWSAPI_OKX**: OKX WebSocket Client.
- **TsgcWSAPI_Kucoin**: KuCoin Spot WebSocket Client.
- **TsgcWSAPI_Kucoin_Futures**: KuCoin Futures WebSocket Client.
- **TsgcWSAPI_Deribit**: Deribit WebSocket Client.
- **TsgcWSAPI_CryptoCom**: Crypto.com WebSocket Client.
- **TsgcWSAPI_HTX**: HTX WebSocket Client.
- **TsgcWSAPI_MEXC**: MEXC WebSocket Client.
- **TsgcWSAPI_Bitget**: Bitget WebSocket Client.
- **TsgcWSAPI_GateIO**: Gate.io WebSocket Client.
- **TsgcWSAPI_XTB**: XTB WebSocket Client.

- **TsgcWSAPI_CexPlus**: CexPlus WebSocket Client.
- **TsgcWSAPI_OpenAI**: OpenAI Realtime WebSocket API Client.

3 sgcWebSocket Libs

- **TsgcTDLib_Telegram**: Telegram API Client.
- **TsgcWhatsApp_Client**: WhatsApp Business Cloud Client.
- **TsgcHTTP_Cryptohopper**: Cryptohopper Client API.
- **TsgcLib_RCON**: RCON Client.

4 sgcWebSocket 프로토콜

- **TsgcWSPClient_MQTT**: MQTT (3.1.1 및 5.0) 클라이언트. WebSocket 및 일반 TCP 연결을 지원합니다.
- **TsgcWSPClient_AMQP1**: AMQP 1.0.0 클라이언트. RabbitMQ 브로커를 지원합니다.
- **TsgcWSPClient_AMQP**: AMQP 0.9.1 클라이언트. RabbitMQ 브로커를 지원합니다.
- **TsgcWSPClient_STOMP**: STOMP 클라이언트, WebSocket 및 일반 TCP 연결을 지원합니다.
 - **TsgcWSPClient_STOMP_ActiveMQ**: ActiveMQ 브로커용 STOMP 클라이언트.
 - **TsgcWSPClient_STOMP_RabbitMQ**: RabbitMQ 브로커용 STOMP 클라이언트.
- **TsgcWSPClient_WAMP**: WAMP 1.0 클라이언트 프로토콜.
- **TsgcWSPServer_WAMP**: WAMP 1.0 서버 프로토콜.
- **TsgcWSPClient_WAMP2**: WAMP 2.0 클라이언트 프로토콜.
- **TsgcWSPServer_AppRTC**: AppRTC Google 프로젝트 기반 WebRTC 서버.
- **TsgcWSPServer_WebRTC**: WebRTC 서버 프로토콜.
- **TsgcWSPClient_sgc**: JSON RPC 기반 WebSocket 클라이언트 SGC 프로토콜.
- **TsgcWSPServer_sgc**: JSON RPC 기반 WebSocket 서버 SGC 프로토콜.
- **TsgcWSPClient_Files**: WebSocket 파일 전송 클라이언트 프로토콜.
- **TsgcWSPServer_Files**: WebSocket 파일 전송 서버 프로토콜.
- **TsgcWSPClient_Dataset**: WebSocket 클라이언트 Dataset 동기화 프로토콜.
- **TsgcWSPServer_Dataset**: WebSocket 서버 Dataset 동기화 프로토콜.
- **TsgcWSPClient_Presence**: WebSocket 클라이언트 Presence 프로토콜.
- **TsgcWSPServer_Presence**: WebSocket 서버 Presence 프로토콜.
- **TsgcWSPClient_E2EE**: 종단간 암호화 클라이언트 프로토콜.
- **TsgcWSPServer_E2EE**: 종단간 암호화 서버 프로토콜.

5 sgcWebSockets HTTP

- **TsgcHTTP1Client**: Indy TIdHTTP 기반 HTTP 1.0 Client.
- **TsgcHTTP2Client**: HTTP 2.0 Client.
- **TsgcHTTP_JWT_Client**: JWT (JSON WEB TOKEN) Client.
- **TsgcHTTP_JWT_Server**: JWT (JSON WEB TOKEN) Server.
- **TsgcHTTP_OAuth2_Client**: OAuth 2.0 Client.
- **TsgcHTTP_OAuth2_Server**: OAuth 2.0 Server.
- **TsgcHTTPAWS_SQS_Client**: Amazon AWS SQS Client.
- **TsgcHTTPGoogleCloud_PubSub_Client**: Google Cloud Pub/Sub Client.
- **TsgcHTTPGoogleCloud_Calendar_Client**: Google Calendar Client.
- **TsgcHTTP_OAuth2_Client_Google**: Google용 OAuth 2.0 Client.

- **TsgcHTTP_OAuth2_Client_Microsoft**: Microsoft용 OAuth 2.0 Client.
- **TsgcHTTP_OAuth2_Server_Provider**: OAuth 2.0 Server Provider.
- **TsgcWSAPIServer_WebAuthn**: passkey 기반 인증을 위한 WebAuthn Server.
- **TsgcHTTPGoogleCloud_FCM_Client**: Google Cloud FCM (Firebase Cloud Messaging) Client.

6 sgcWebSockets IoT

- **TsgcloAmazon_MQTT_Client**: Amazon MQTT IoT Core Client입니다.
- **TsgcloAzure_MQTT_Client**: Azure IoT MQTT Client입니다.

7 sgcWebSockets P2P

- **TsgcUDPCliant**: UDP Client입니다.
- **TsgcUDPServer**: UDP Server입니다.
- **TsgcSTUNClient**: STUN Client입니다.
- **TsgcSTUNServer**: STUN Server입니다.
- **TsgcTURNClient**: STUN / TURN Client입니다.
- **TsgcTURNServer**: STUN / TURN Server입니다.
- **TsgcICEClient**: ICE Client입니다.
- **TsgcRTCPeerConnection**: WebRTC Peer Connection Client입니다.

8 sgcWebSockets DataSnap

- **TsgcWSHTTPWebBrokerBridgeServer**: HTTP + WebSockets 지원이 있는 DataSnap 서버 교체품.
- **TsgcWSHTTP2WebBrokerBridgeServer**: HTTP + HTTP/2 + WebSockets 지원이 있는 DataSnap 서버 교체품.
- **TsgcWSServer_HTTPAPI_WebBrokerBridge**: HTTP.SYS Microsoft Server 기반 DataSnap 서버 교체품.

9 sgcWebSockets AI

- **TsgcAIOpenAIChatBot**: 음성 명령으로 ChatBot을 구축합니다.
- **TsgcAIOpenAITranslator**: 실시간 번역.
- **TsgcAudioRecorderMCI**: MCI를 사용하여 오디오를 녹음합니다.
- **TsgcAudioPlayerMCI**: MCI를 사용하여 오디오를 재생합니다.
- **TsgcTextToSpeechSystem**: 운영 체제 기본값을 사용하는 Text-To-Speech.
- **TsgcTextToSpeechGoogle**: Google Cloud를 사용하는 Text-To-Speech.
- **TsgcTextToSpeechAmazon**: Amazon AWS를 사용하는 Text-To-Speech.
- **TsgcAIOpenAIEmbeddings**: 사용자 지정 데이터를 사용하여 AI 애플리케이션을 구축할 수 있습니다.
- **TsgcAIDatabaseVectorFile**: vector를 일반 텍스트 파일에 저장합니다.
- **TsgcAIDatabaseVectorPinecone**: Pinecone vector 데이터베이스를 지원합니다.
- **TsgcAIOpenAIAssistant**: OpenAI Assistants API Client.
- **TsgcWSAPIServer_MCP**: MCP (Model Context Protocol) Server.
- **TsgcWSAPIClient_MCP**: MCP (Model Context Protocol) Client.

Versions Support

Delphi 지원 IDE

- Delphi 7 (* Indy 10으로 업그레이드한 경우에만 지원되며, Intraweb는 지원되지 않습니다)
- 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

CBuilder 지원 IDE

- 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

FreePascal 지원 IDE

- Lazarus

체험판 버전

무료 버전과 함께 제공되는 컴파일된 *.dcu 파일은 기본 Indy 및 Intraweb 버전을 사용합니다. 이러한 패키지 중 하나라도 업그레이드한 경우 아마도 작동하지 않을 것이며 전체 소스 코드 버전을 구입해야 합니다.

Indy Package

일부 구성 요소(예: TsgcWebSocketClient 또는 TsgcWebSocketServer)는 Indy를 TCP/IP 라이브러리로 사용하므로, sgcWebSockets 패키지를 설치하려면 Indy가 필요합니다. 기본적으로 sgcWebSockets는 RAD Studio에 내장된 Indy 라이브러리를 사용하지만, 더 많은 기능을 가진 사용자 정의 Indy 버전을 제공합니다: OpenSSL API 1.1, OpenSSL 3.0, ALPN 프로토콜 지원...

설치

Delphi / CBuilder / Lazarus

1. 포함된 파일을 디렉터리 {\$DIR}에 압축 해제하십시오

2. Delphi\CBuilder에서:

파일이 압축 해제되는 디렉터리 {\$DIR}를 Tools, Environment options, Directories에서 Delphi\CBuilder library path에 추가하십시오

모든 Delphi\CBuilder 버전

디렉터리 {\$DIR}\source를 라이브러리 경로에 추가하십시오

특정 Delphi 버전의 경우

Delphi 7 : {\$DIR}\libD7 디렉터리를 라이브러리 경로에 추가하십시오.
 Delphi 2007 : library path에 {\$DIR}\libD2007 디렉터리를 추가하십시오
 Delphi 2009 : {\$DIR}\libD2009 디렉터리를 라이브러리 경로에 추가하십시오
 Delphi 2010 : 디렉터리 {\$DIR}\libD2010을 library path에 추가하십시오
 Delphi XE : {\$DIR}\libDXE 디렉터리를 라이브러리 경로에 추가하십시오.
 Delphi XE2 : 디렉터리 {\$DIR}\libDXE2\\$(Platform)를 라이브러리 경로에 추가하십시오
 Delphi XE3 : {\$DIR}\libDXE3\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오.
 Delphi XE4 : library path에 {\$DIR}\libDXE4\\$(Platform) 디렉터리를 추가하십시오
 Delphi XE5 : {\$DIR}\libDXE5\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오
 Delphi XE6 : {\$DIR}\libDXE6\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오
 Delphi XE7 : 라이브러리 경로에 디렉터리 {\$DIR}\libDXE7\\$(Platform) 을 추가하십시오
 Delphi XE8 : {\$DIR}\libDXE8\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오.
 Delphi 10 : {\$DIR}\libD10\\$(Platform) 디렉토리를 라이브러리 경로에 추가합니다
 Delphi 10.1 : {\$DIR}\libD10_1\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오
 Delphi 10.2 : {\$DIR}\libD10_2\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오.
 Delphi 10.3 : {\$DIR}\libD10_3\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오
 Delphi 10.4 : {\$DIR}\libD10_4\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오
 Delphi 11 : {\$DIR}\libD11\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오
 Delphi 12 : {\$DIR}\libD12\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오
 Delphi 13 : {\$DIR}\libD13\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오

특정 CBuilder 버전용

C++ Builder 2010 : {\$DIR}\libD2010 디렉터리를 라이브러리 경로에 추가하십시오
 C++ Builder XE : 라이브러리 경로에 {\$DIR}\libDXE 디렉터리를 추가하십시오.
 C++ Builder XE2 : 디렉터리 {\$DIR}\libDXE2\\$(Platform)을 library path에 추가하십시오
 C++ Builder XE3 : 디렉터리 {\$DIR}\libDXE3\\$(Platform)을(를) 라이브러리 경로에 추가하십시오
 C++ Builder XE4 : 디렉터리 {\$DIR}\libDXE4\\$(Platform)를 라이브러리 경로에 추가하십시오
 C++ Builder XE5 : {\$DIR}\libDXE5\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오.
 C++ Builder XE6 : 디렉터리 {\$DIR}\libDXE6\\$(Platform)를 라이브러리 경로에 추가하십시오
 C++ Builder XE7 : {\$DIR}\libDXE7\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오
 C++ Builder XE8 : {\$DIR}\libDXE8\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오
 C++ Builder 10 : 디렉터리 {\$DIR}\libD10\\$(Platform)을(를) 라이브러리 경로에 추가하십시오
 C++ Builder 10.1 : {\$DIR}\libD10_1\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오
 C++ Builder 10.2 : 라이브러리 경로에 디렉터리 {\$DIR}\libD10_2\\$(Platform) 을 추가하십시오
 C++ Builder 10.3 : {\$DIR}\libD10_3\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오
 C++ Builder 10.4 : {\$DIR}\libD10_4\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오
 C++ Builder 11 : 디렉터리 {\$DIR}\libD11\\$(Platform)를 라이브러리 경로에 추가하십시오
 C++ Builder 12 : {\$DIR}\libD12\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오

C++ Builder 13 : {\$DIR}\libD13\\$(Platform) 디렉터리를 라이브러리 경로에 추가하십시오

모든 CBuilder 버전에서 dcp\\$(Platform)을 라이브러리 경로에 추가하십시오(.bpi 파일 포함).

3. Delphi에서

선택

File, Open에서 올바른 Packages\sgcWebSockets.groupproj를 탐색하십시오(먼저 sgcWebSocketsX.dpk를 컴파일한 다음 dclsgcWebSocketsX.dpk를 설치하십시오)

Delphi용 패키지 파일

```
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. CBuilder에서

선택

File, Open에서 올바른 Packages\sgcWebSockets.groupproj를 탐색하십시오(먼저 sgcWebSocketsX.dpk를 컴파일한 다음 dclsgcWebSocketsX.dpk를 설치하십시오)

CBuilder용 패키지 파일

```
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. Lazarus에서

File, Open을 선택하고 Packages\sgcWebSocketsLazarus.lpk를 탐색하십시오(먼저 컴파일한 다음 설치)

컴파일된 파일은 Lazarus 디렉터리에 있습니다. 그 안에는 최신 Indy 소스 버전이 있는 Indy 디렉터리가 있습니다.

Lazarus 2.0.6 및 Indy 10.5.9.4930으로 테스트됨

6. 데모

모든 데모는 Demos 하위 디렉터리에서 사용할 수 있습니다. 프로젝트를 열고 실행하기만 하면 됩니다. Intraweb 데모는 Intraweb 버전이 다르기 때문에 일부 유닛을 수정해야 할 수 있습니다.

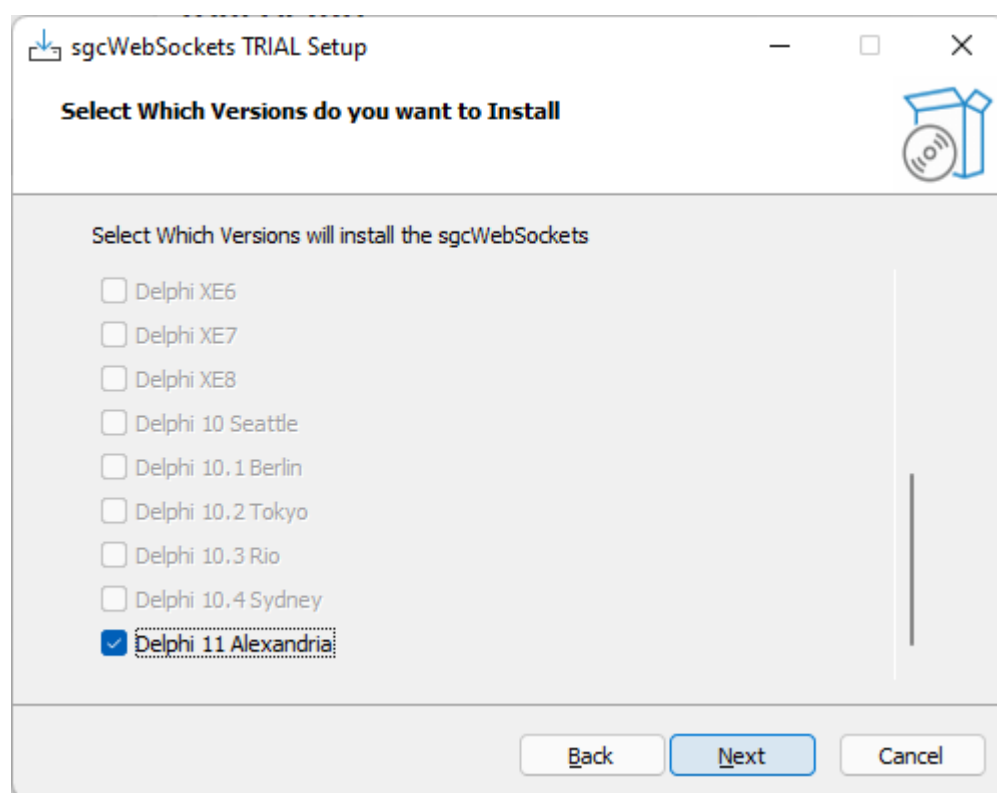
Install Setup

*최소 Windows Vista가 필요합니다(Windows 2000, XP 및 Server 2003은 지원되지 않음).

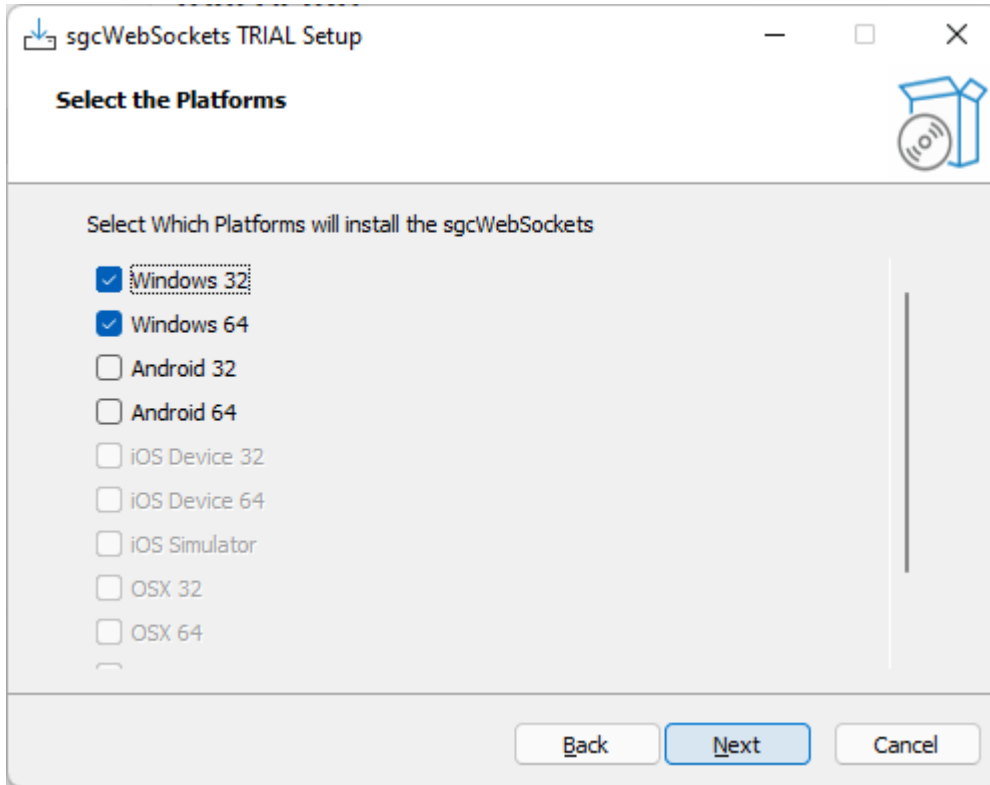
Windows Setup을 사용하여 sgcWebSockets 라이브러리를 설치하면 설치가 안내되며 매우 간단합니다. 설치 중에 오류가 있으면 [Install Errors](#) 페이지를 참조하고 [패키지를 수동으로 설치](#)해 볼 수 있습니다.

Trial Setup

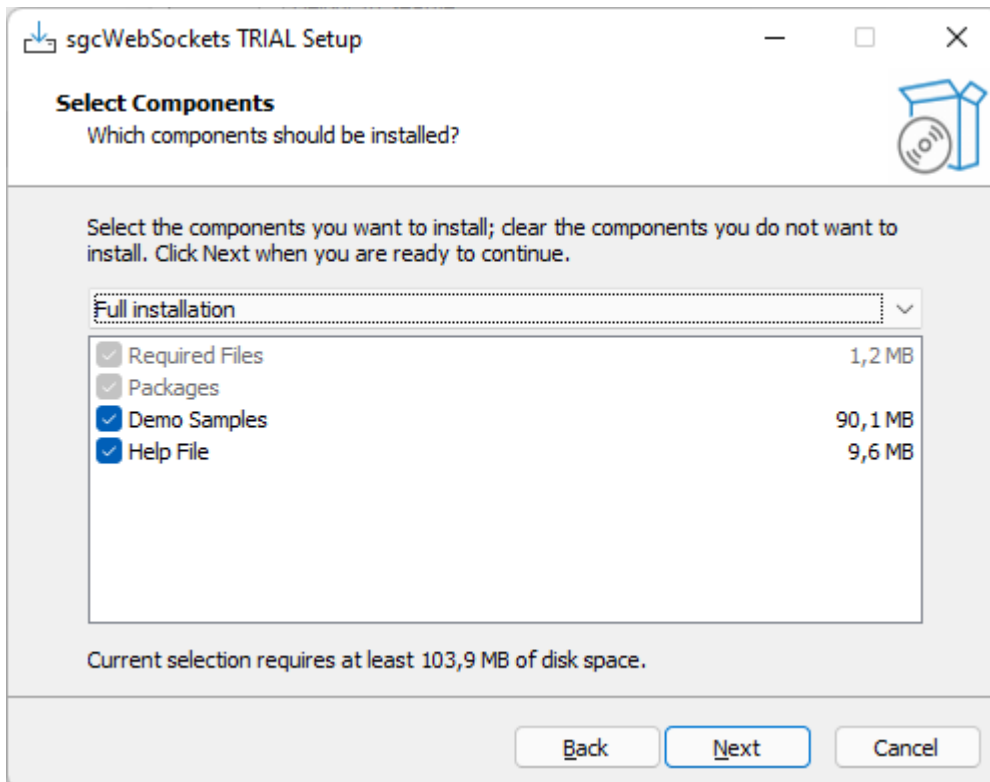
- Trial Installer를 실행하십시오.
- Trial 설치에 Admin 권한이 필요합니다.
- 설치 프로그램은 Delphi / CBuilder / Rad Studio 버전 목록을 표시하며 기본적으로 다운로드된 버전이 활성화됩니다. 이 버전이 설치 프로그램에 의해 감지되지 않으면, 설치 프로그램은 파일을 추출하지만 컴파일을 시도하지 않습니다. [패키지를 수동으로 설치](#)를 참조하십시오.



- 다음 페이지는 설치할 수 있는 플랫폼을 보여줍니다. 설치 프로그램이 감지한 플랫폼만 활성화됩니다.



- 다음 페이지에는 체험판을 설치하려면 수락해야 하는 라이선스 계약이 표시됩니다.
- 라이선스 계약에 동의한 후 설치될 구성 요소가 표시됩니다. 기본적으로 패키지, 컴파일된 DCU, 데모 및 도움말 파일이 모두 설치됩니다. 도움말 파일과 데모의 설치 여부를 사용자 정의할 수 있습니다.



- 마지막으로, 파일을 추출하고, 패키지를 컴파일 및 설치하고, IDE에 필요한 경로를 등록합니다.

Customers Setup

라이선스를 구매한 사용자는 설치 프로그램을 사용하여 sgcWebSockets 라이브러리를 설치할 수 있습니다. 패키지 설치를 위한 단계별 지침은 아래에서 확인하십시오.

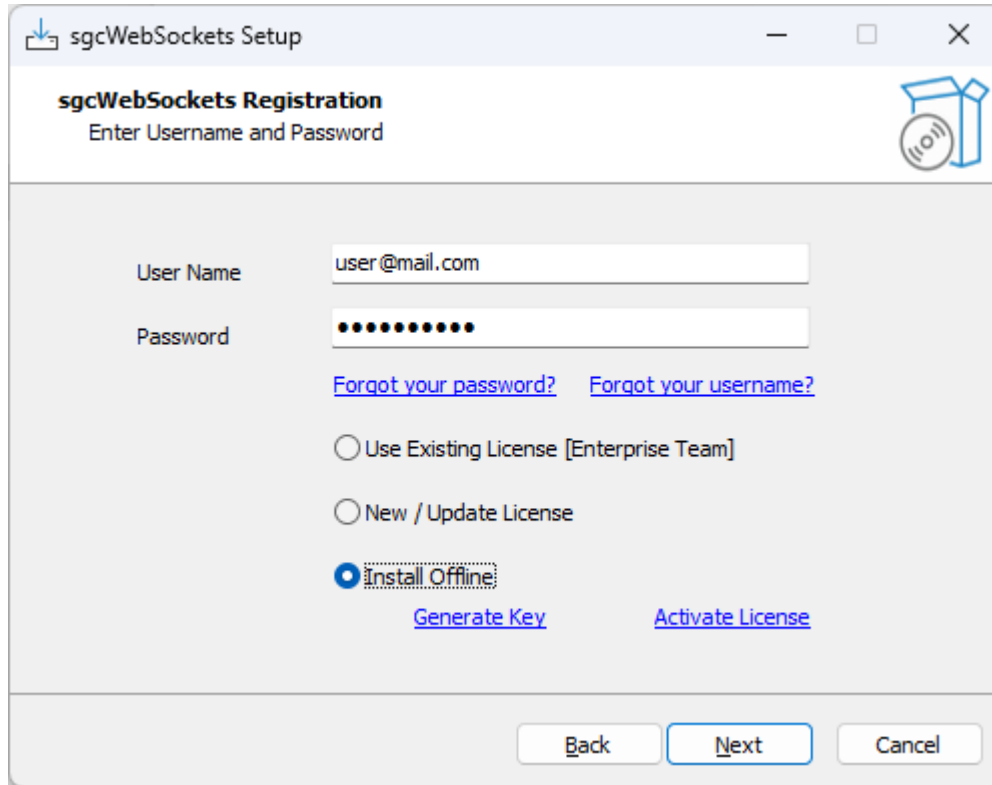
- 설치 프로그램을 실행하십시오.
- 설치 프로그램은 최저 권한으로 실행됩니다(관리자로 실행되면 네트워크 드라이브에 설치할 수 없습니다). 대상에 관리자 권한이 필요한 경우 설정을 관리자로 실행하십시오.
- 먼저 프라이빗 eSeGeCe 계정의 사용자 이름/비밀번호를 설정해야 합니다. 이는 한 번만 입력하면 됩니다. 다음에 설정을 사용할 때 설치 프로그램이 이전에 저장된 값을 읽습니다.

The image shows the 'sgcWebSockets Setup' window. The title bar says 'sgcWebSockets Setup'. The main title is 'sgcWebSockets Registration' with the subtitle 'Enter Username and Password'. There is a small icon of a box with a coin in the top right corner. The form has 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]' (which is selected), 'New / Update License', and 'Install Offline'. At the bottom, there are three buttons: 'Back', 'Next' (which is highlighted with a blue border), and 'Cancel'.

- 3가지 옵션이 있습니다:
 - **Use Existing License:** 이 버전이 이미 설치된 경우 기본적으로 이 옵션이 선택됩니다. 이 버전에 대한 최신 구성을 사용합니다.
 - **New / Update License:** 이 버전이 이전에 설치되지 않은 경우 기본적으로 이 옵션이 선택됩니다. Server License에 연결하여 라이선스 정보를 가져옵니다. 최근에 라이선스를 업그레이드한 경우 이 옵션을 선택하여 라이선스를 업데이트할 수 있습니다.
 - **Install Offline:** 컴퓨터에 인터넷 액세스가 없는 경우 이 옵션을 선택하여 라이선스를 활성화하십시오.

키 생성

이 옵션은 라이선스를 활성화하는 데 사용될 키를 생성합니다.



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](#)

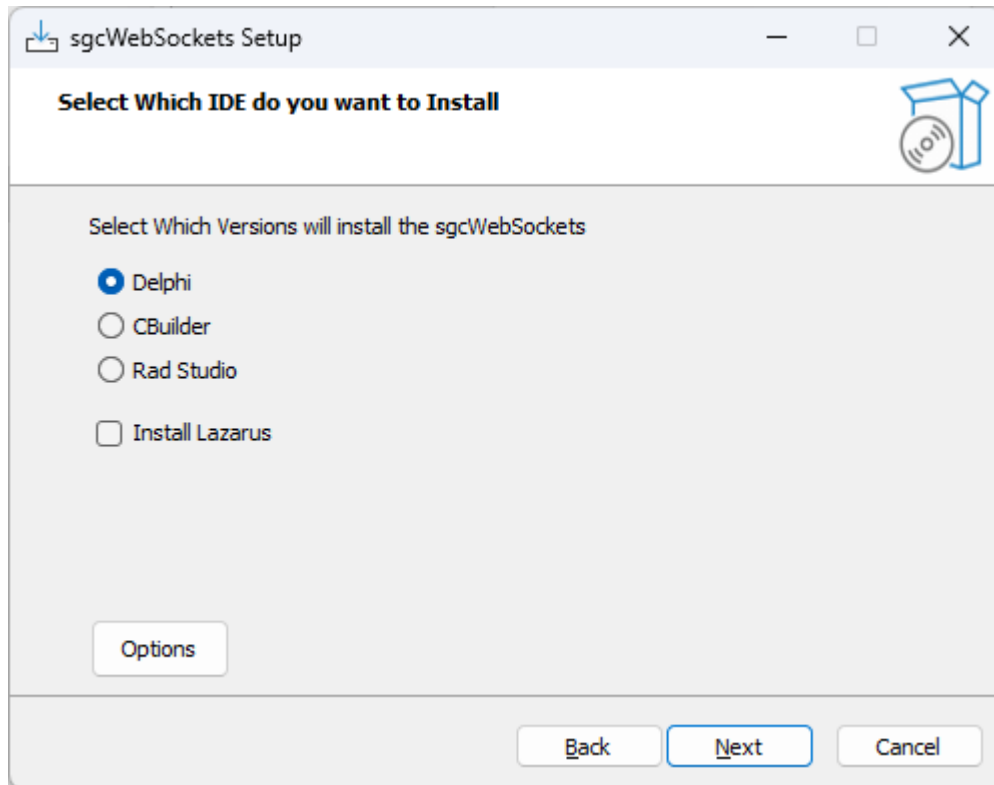
키를 복사하여 비공개 온라인 계정에 액세스하십시오: www.esegece.com/my-account/subscriptions.



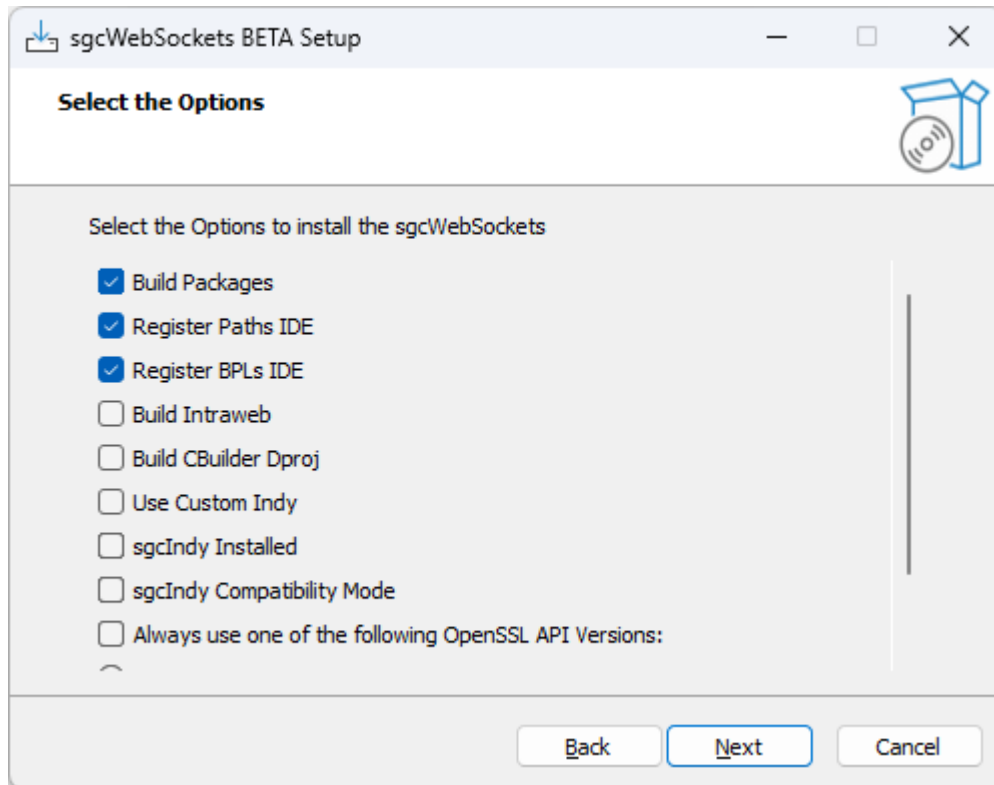
Generate Key - eSeGeCe Software (2023.8.0)

VjJ4b2MxTXlVWghpUm1oVltdHdZVlpXU2xOak1VNvIZMGhhYUZZd01UVlhhMIzVmpBeGMxZHVRbGRXUldzeFdWY3hSbVZXV25GUmJYQnNWbGQ0TmxVeFZrOVRNREZ6WTBac1ZHSnVRbTlXym5CelRWWk9XR05JV21GTlIzaDRWVEZvYTFSDfJuSk5WRVpWVFVvd2QxcEVRbk5qUjFGNVdrVndhVl6YUhsVk1XUjNaR3h2ZDJKSvJ3sUlHSM2hvVld0YWQyTnNaRmhOVmtwT1RvAG9XVll4VWt0WlZscEdVbXBDVldFeFNsQlphMVV4VmtVeFdHskZOVmRTVlhCM1ZrVmFVMU5yTVZaTlNHeFRWa1phVUZsWE1VOUSNVkoxWTBoT2FWWlVSVEZWTVdoUfUyeEZkMkd6YUZwbGEzQlIXbFprUzJSR1NuVlZiVpXVZzhMlF3VFRGU2NsVnJTazlTVkZaV1ZteG9hMVZHV2taU2FsSlZWbFpLZFZSc1ZYaFNWa1pWVld4Q1YxSXpVWHBxUkVaVFVXMVdjazFWVm1GbGEwcFBXVmQwUmsxR1VYaFZiSEJyVfVSc1ZsVldVbGRVYkVwSVpFuk9WVTFHU2tOVWExVTFVbFpTV1ZwR1FsTlNNRFF4VmtaV1UySnJNWEpQVmxalU1ZrWktVRmxYTVU5Tk1WRjRwV3RhYTAXRVJrWlpWRTV2VkrKR2NsSlVSbFZXyKvweFdrUkJOVlpXU2xWVGF6VIRVak5STUZaSE1YZFJheIZXVFZWV1dGZEhVbEZWYTFwV1RWWlNWbFZ0TlU1aGVsWldWVlpTVTFReVJuUmtSRVWVFVaS1lWUnNWWGhTVm5BMLUyeENVMUpWVlhkV1J6rJNzbTFSZDA5VlZsaFdSWEJQVld0a2FtVldVWGRWYTbwUFVsUlZnBfZyYUd0VlJrcElWR3BPVkJaNVVqTlZSa1U1VUZFOVBRPT0=

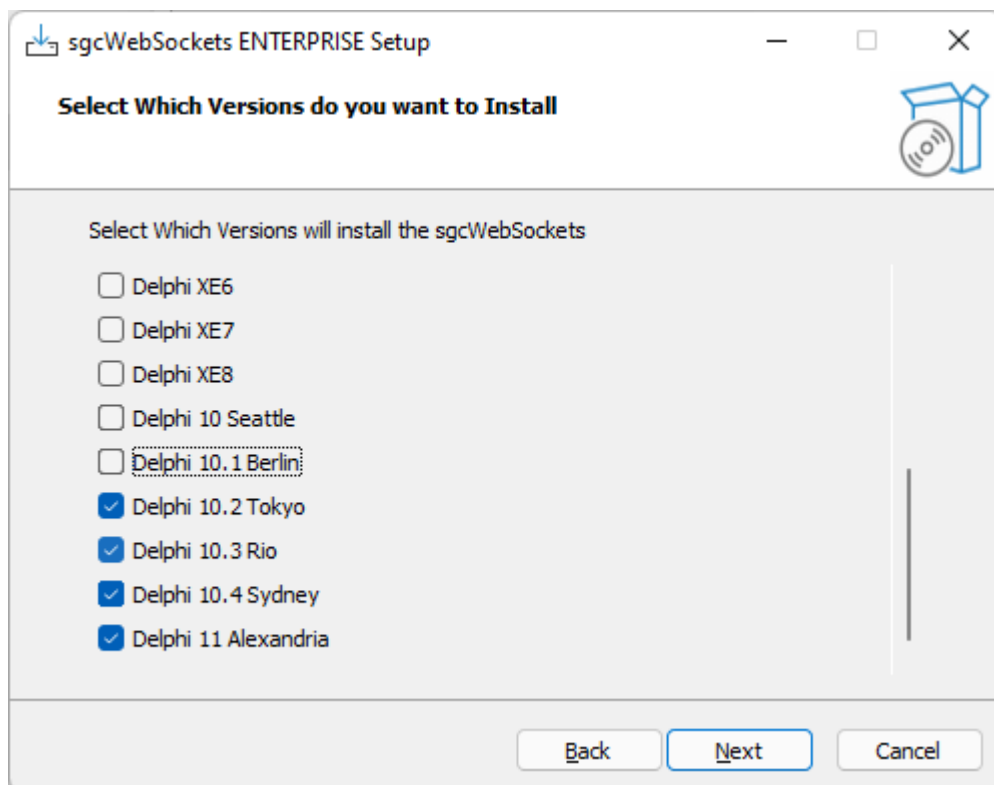
활성화할 구독을 선택하고 키를 붙여넣으십시오.



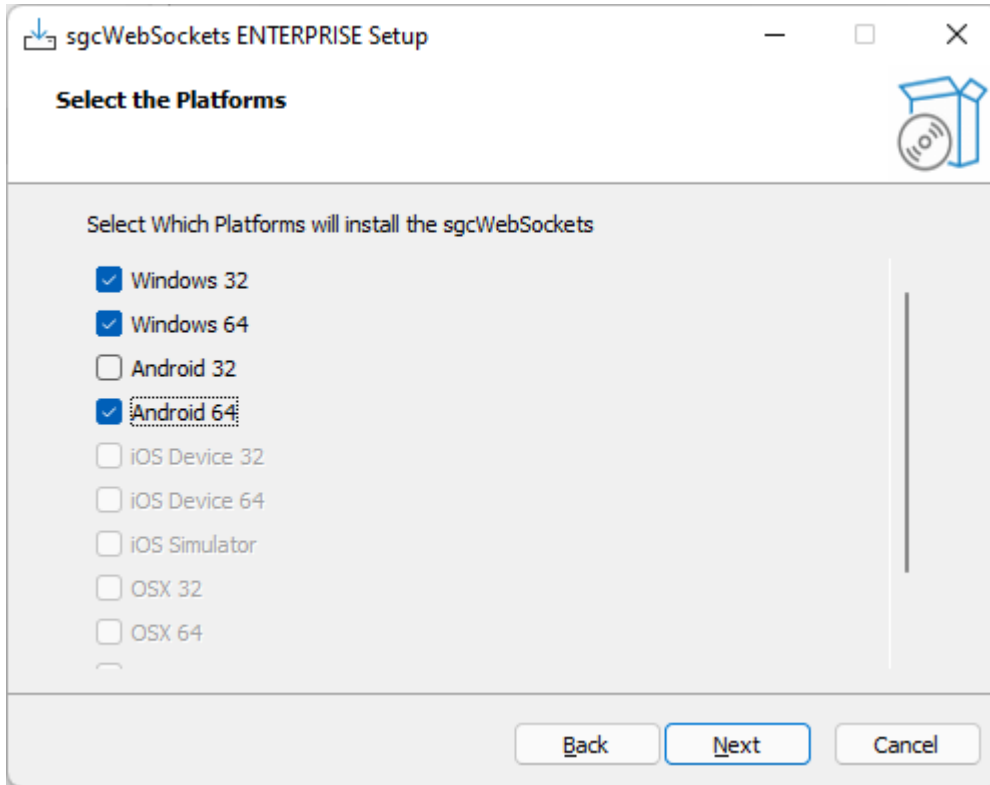
- 설치 프로그램을 사용할 때마다 사용자 정의할 수 있는 일부 옵션이 있습니다. 이러한 속성에 접근하려면 **Options** 버튼을 누르십시오.
 - Build Packages:** 선택하면 설치 프로그램이 패키지를 빌드하려고 시도합니다.
 - Register Paths IDE:** 선택하면 설치 프로그램이 IDE에 필요한 라이브러리 경로를 등록합니다.
 - Register BPLs IDE:** 선택하고 설치 프로그램이 패키지를 성공적으로 빌드한 경우, 설치 프로그램이 IDE에 디자인 타임 패키지를 등록합니다.
- 다음 옵션은 소스 코드가 있는 라이선스에서만 사용할 수 있습니다:
 - Build Intraweb:** 선택하면 설치 프로그램이 필요한 Intraweb 파일을 설치합니다(기본적으로 비활성화됨). 현재 Intraweb XVI 버전을 설치합니다.
 - Build CBuilder Dproj:** 선택하면 설치 프로그램이 sgcWebSockets Delphi 패키지를 사용하여 CBuilder 패키지를 빌드하고 필요한 모든 CBuilder 파일을 생성합니다.
 - Use Custom Indy:** (Enterprise 전용) 선택하면 sgcWebSockets가 Custom Indy 버전(openSSL 1.1 및 3.0, TLS 1.3, ALPN 등 지원)을 사용합니다.
 - sgcIndy Already Installed:** sgcIndy 패키지가 설치되어 있고 이 패키지를 사용하여 sgcWebSockets 패키지를 컴파일하려면 이 옵션을 선택하십시오.
 - sgcIndy Compatibility Mode:** sgcIndy 패키지가 Compatibility Mode로 설치된 경우(다른 패키지가 Indy를 사용하기 때문에, 예: DevExpress) 이 옵션을 선택하십시오.
 - Always use of the following OpenSSL API Versions:** OpenSSL 1.1.1 또는 OpenSSL 3.0.0 API의 사용을 강제하려면 이 옵션을 선택하십시오.
 - Debug Mode:** 디버그 메시지를 로그 파일에 저장합니다. 프로덕션 환경에서는 이 모드를 사용하지 마십시오.
 - Build Rad Studio IDE Win64:** Rad Studio IDE 12+ 64bit용 디자인 타임 패키지를 빌드합니다.
 - Include Resources:** 기본적으로 활성화됩니다. JavaScript 클라이언트 번들(sgcResources.RES)을 런타임 패키지에 포함합니다. 포함된 리소스는 서버가 sgcWebSockets 브라우저 클라이언트를 HTTP를 통해 원격 브라우저에 제공하는 경우에만 필요합니다. 패키지가 컴파일되기 전에 sgcVer.inc에서 SGC_RESOURCES를 정의 해제하려면 이 옵션을 선택 해제하여 소스에 링크하는 모든 실행 파일의 크기를 줄이십시오. Delphi / C++Builder 클라이언트 전용 애플리케이션과 번들 JS 클라이언트를 제공하지 않는 서버의 경우 비활성화해도 안전합니다. 기본 컴파일러 지시문은 [Configure Install](#)을 참조하십시오.



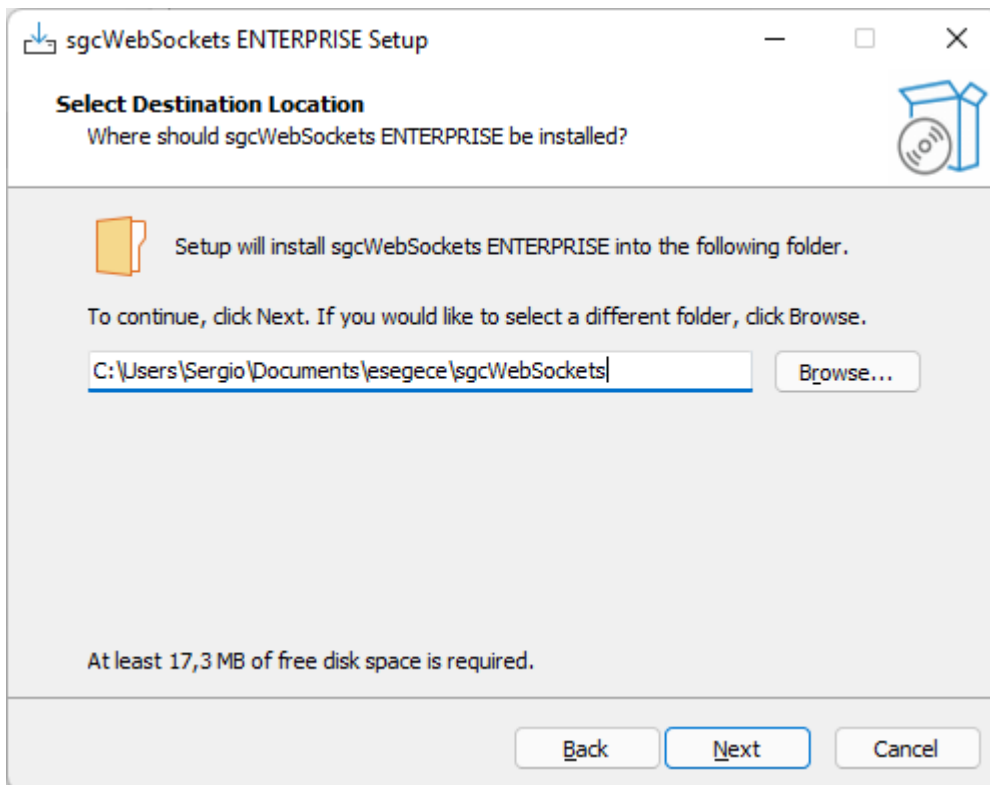
- 이제 설치할 IDE 버전을 선택할 수 있습니다. 설치 프로그램이 설치된 것으로 감지하는 IDE 버전만 사용할 수 있습니다.



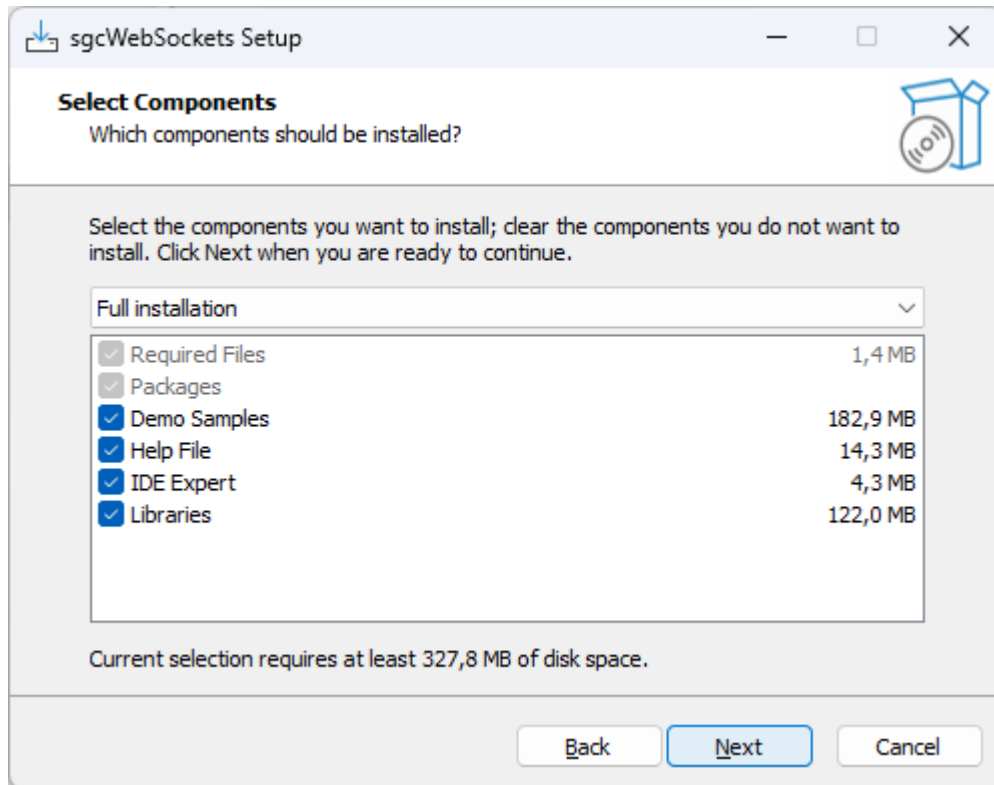
- 다음 단계는 플랫폼을 선택하는 것입니다.



- 패키지가 설치될 폴더를 선택하십시오. 패키지를 다시 설치하면 설치 프로그램이 이전 설치에서 선택한 동일한 폴더를 기본적으로 선택합니다.



- 설치할 구성 요소를 선택하십시오. 등록된 고객은 IDE에서 eSeGeCe 계정에 연결하고, 사용 가능한 업데이트가 있는지 알고, 헬프데스크에 직접 액세스하는... 등의 **IDE expert**를 가지고 있습니다.



- 마지막으로, 파일을 추출하고, 패키지를 컴파일 및 설치하고, IDE에 필요한 경로를 등록합니다.

Install Errors

- MsBuild는 **Library Path의 길이가 너무 길면** 오류를 발생시킵니다. 이 문제를 해결하려면 library path에서 사용하지 않는 경로를 삭제해 보십시오. MsBuild에는 32K 문자 제한이 있습니다.
- 오류 **F2110: 'tchar.h' file not found**. 이 오류는 CBuilder용으로 컴파일할 때 나타날 수 있습니다. 이를 수정하려면 CBuilder를 열고 닫은 다음 다시 설치를 시도하십시오.

설치 명령줄 매개변수

설치 관리자는 다음 명령을 지원합니다.

/SILENT

마법사와 배경 창은 표시되지 않지만 설치 진행 상황은 표시됩니다.

/VERYSILENT

설정이 매우 조용할 때는 이 설치 진행 창이 표시되지 않습니다.

/EXTRACT

패키지는 설치되지 않고 추출만 됩니다. 설치되는 경로는 /EXTRACT=path-to-folder를 사용하여 사용자 정의할 수 있습니다. 사용자 상호 작용 없이 파일만 추출하려면 이 매개변수와 /SILENT를 사용하십시오.

/IDE

이 매개변수를 사용하면 설치할 IDE를 설정할 수 있습니다. 다음 중 하나를 설정하십시오:

- delphi
- cbuilder
- radstudio

추가로 Lazarus를 추가할 수 있습니다.

예제: delphi와 lazarus를 설치합니다.

/ide=delphi-lazarus.

/VERSIONS

이 매개변수를 사용하여 설치할 RAD Studio 버전을 설정할 수 있습니다. 여러 옵션이 허용됩니다:

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

가능한 모든 버전을 설치하려면 "All" 값을 사용하십시오.

예: Delphi 10과 Delphi 13을 설치합니다.

/versions=D10-D13

/PLATFORMS

이 매개변수를 사용하여 설치하려는 RAD Studio 플랫폼을 설정할 수 있습니다. 여러 옵션이 허용됩니다:

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

가능한 모든 플랫폼을 설치하려면 "All" 값을 사용하십시오.

예: Win32 및 Win64 설치.

/platforms=Win32-Win64

/USERNAME

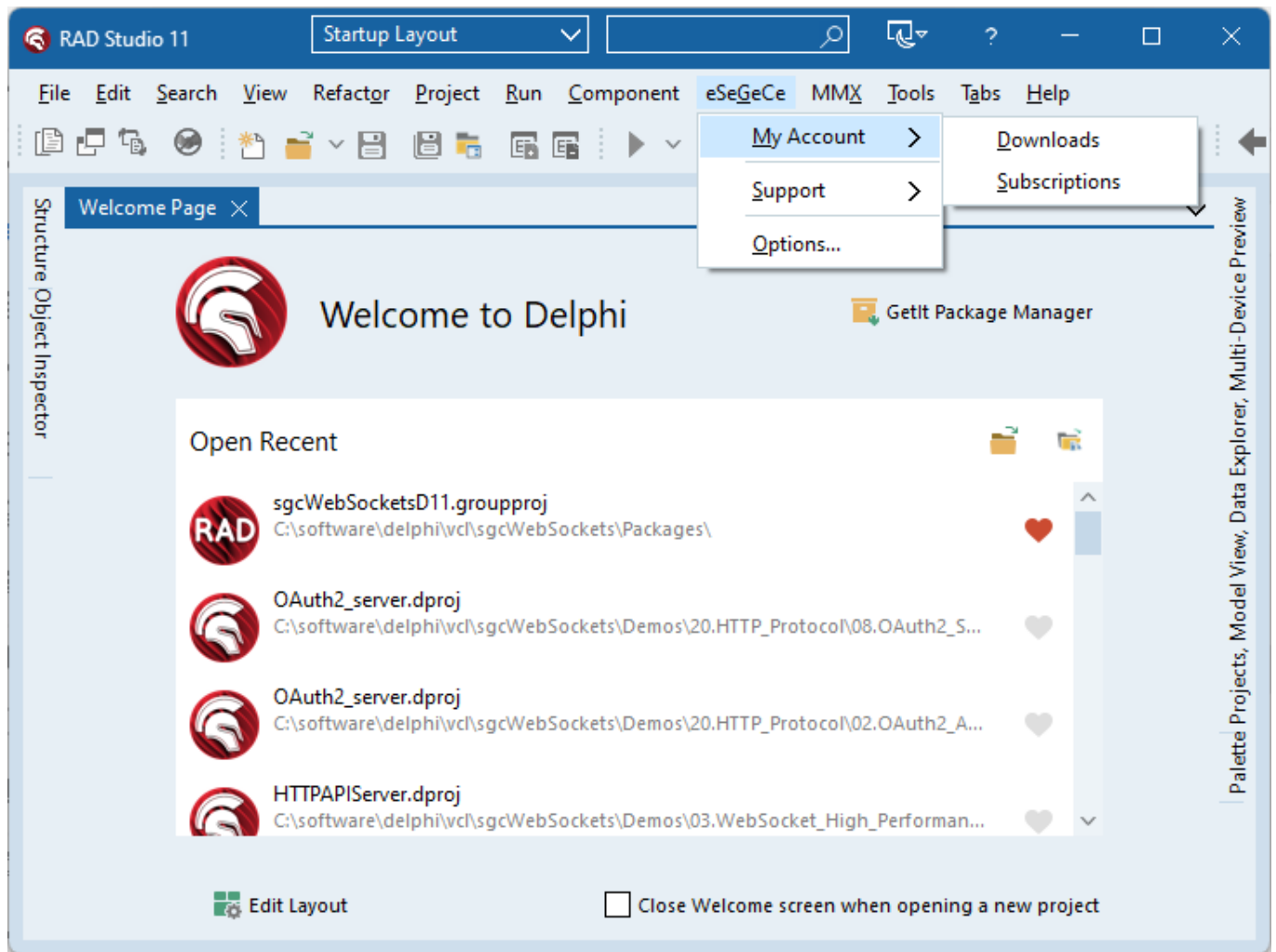
구독의 사용자 이름을 설정합니다.

/PASSWORD

구독의 비밀번호를 설정합니다.

IDE Expert

IDE Expert가 설치되어 있으면 다음 메뉴 옵션을 찾을 수 있습니다:

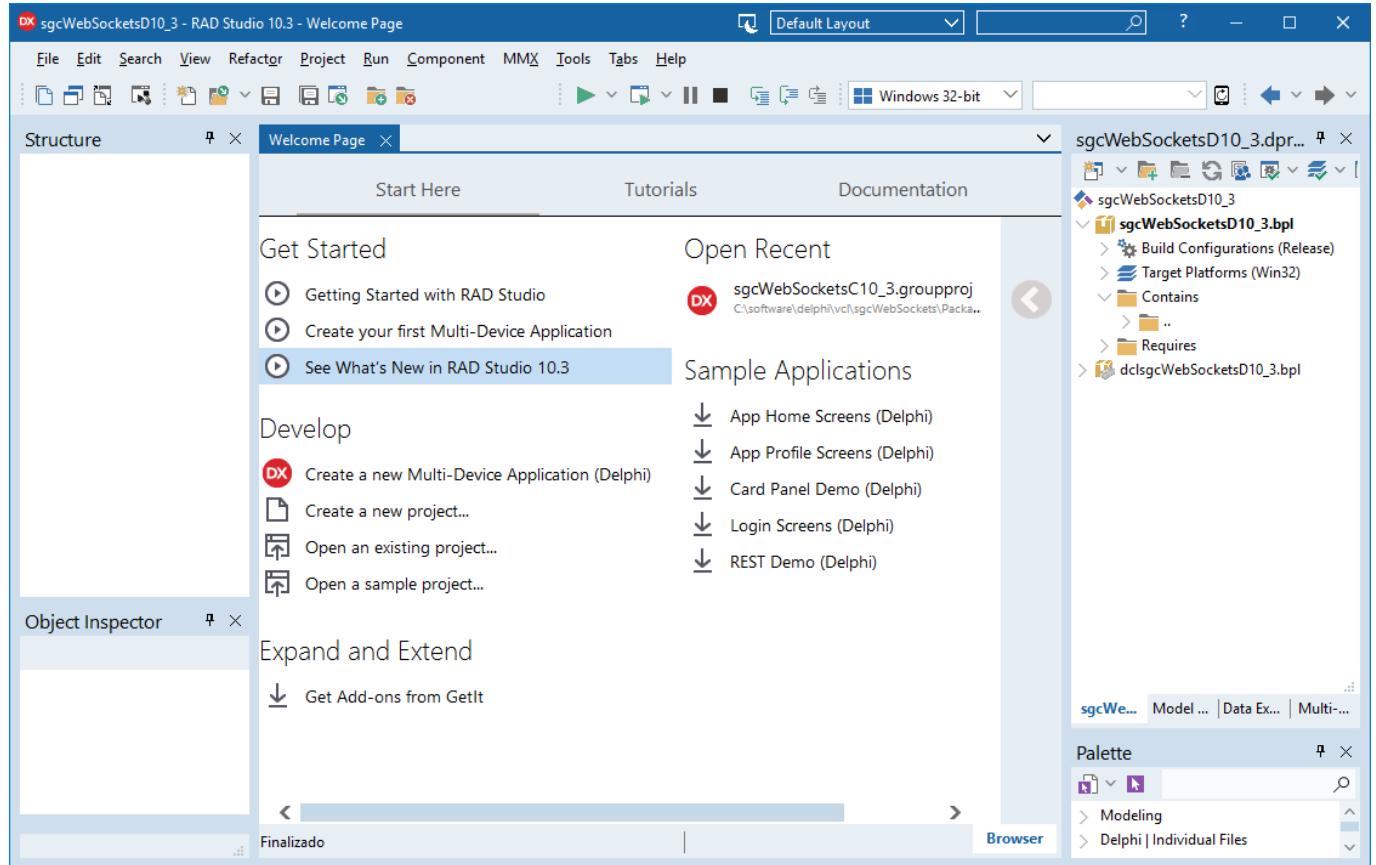


- **MyAccount:** Downloads 메뉴(베타 최신 버전을 다운로드할 수 있는 곳)와 Subscriptions에 직접 액세스하여 라이선스를 관리하거나 만료된 라이선스를 갱신할 수 있습니다.
- **Support:** 자동 로그인으로 HelpDesk 또는 Forum에 직접 액세스할 수 있습니다. Documentation 및 Contact Us 양식도 사용할 수 있습니다.
- **Options:** 이 메뉴에서 계정의 사용자 이름/비밀번호를 구성할 수 있습니다. 기본 브라우저를 선택하고 사용 가능한 업데이트가 있는지 확인하십시오.

패키지 수동 설치

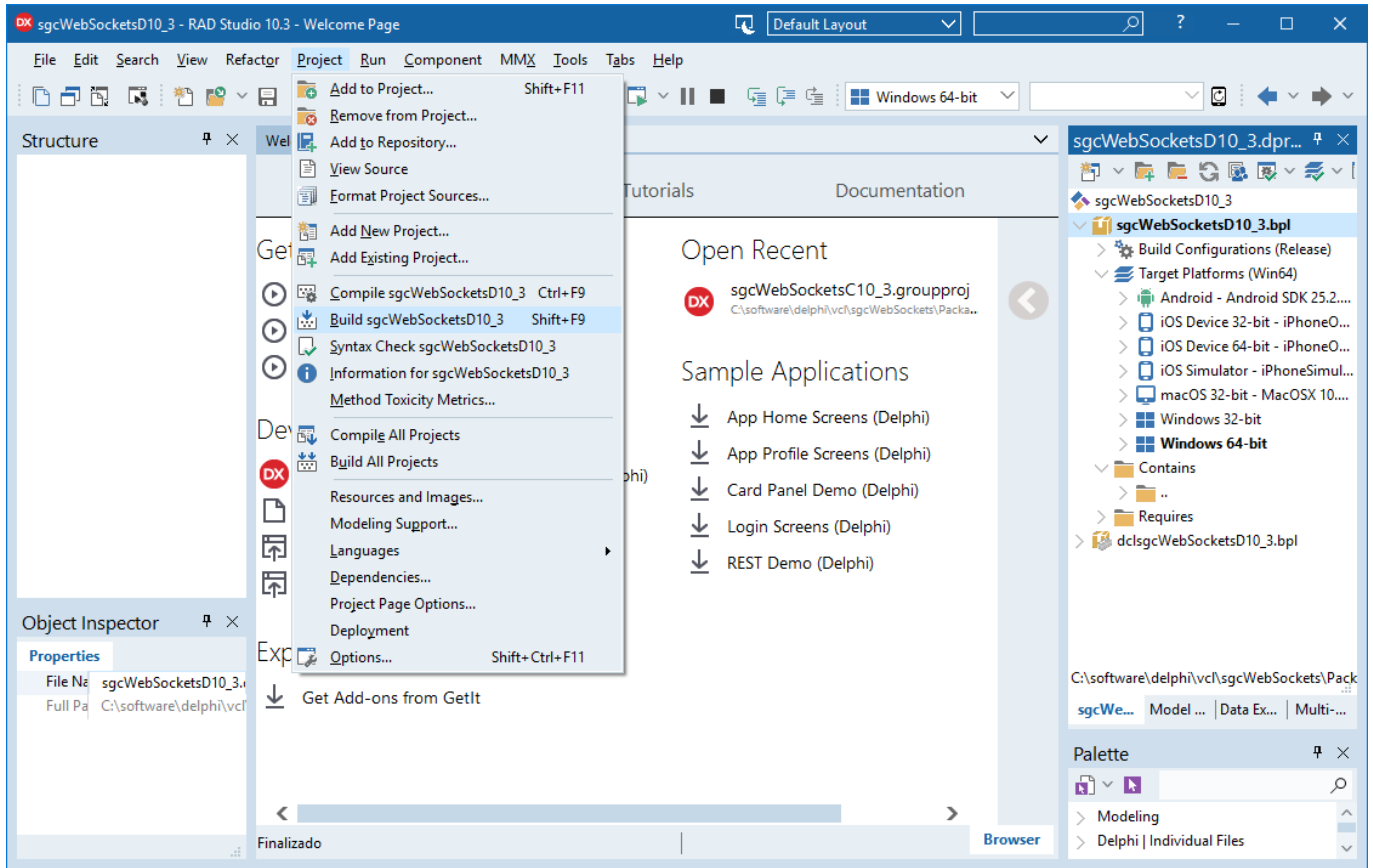
sgcWebSockets 패키지를 설치하려면 다음 단계를 따르십시오. 스크린샷은 Delphi 10.3 버전을 사용합니다.

1. sgcWebSocketsD10_3 그룹 프로젝트를 여십시오.

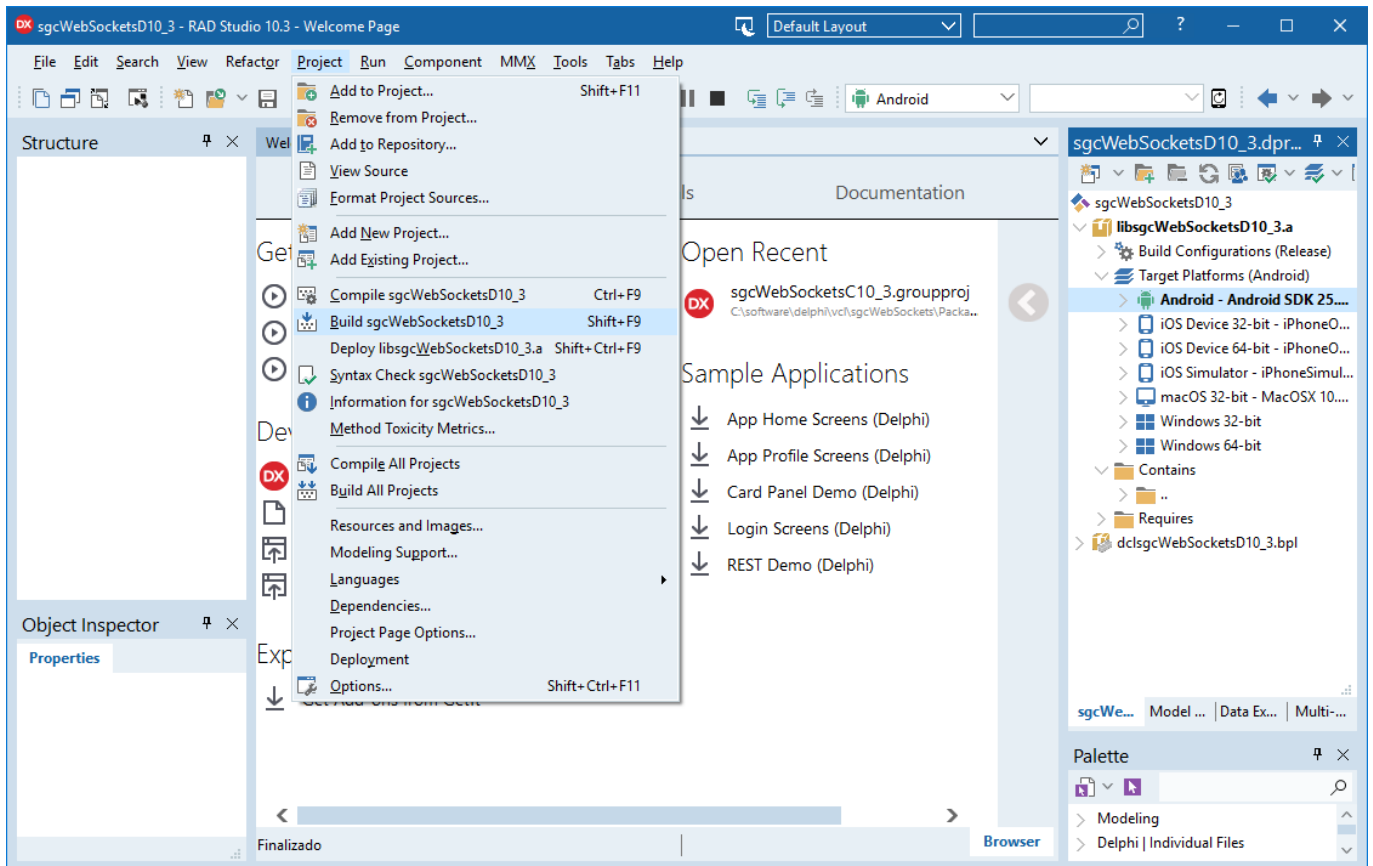


2. 이제 먼저 런타임 패키지를 컴파일해야 합니다(이름이 sgcWebSockets로 시작). 모든 대상 플랫폼마다 하나의 패키지가 있으며 이는 Delphi 버전에 따라 다르므로, 대상 플랫폼을 하나씩 선택하고 빌드하여 모든 패키지를 빌드하십시오.

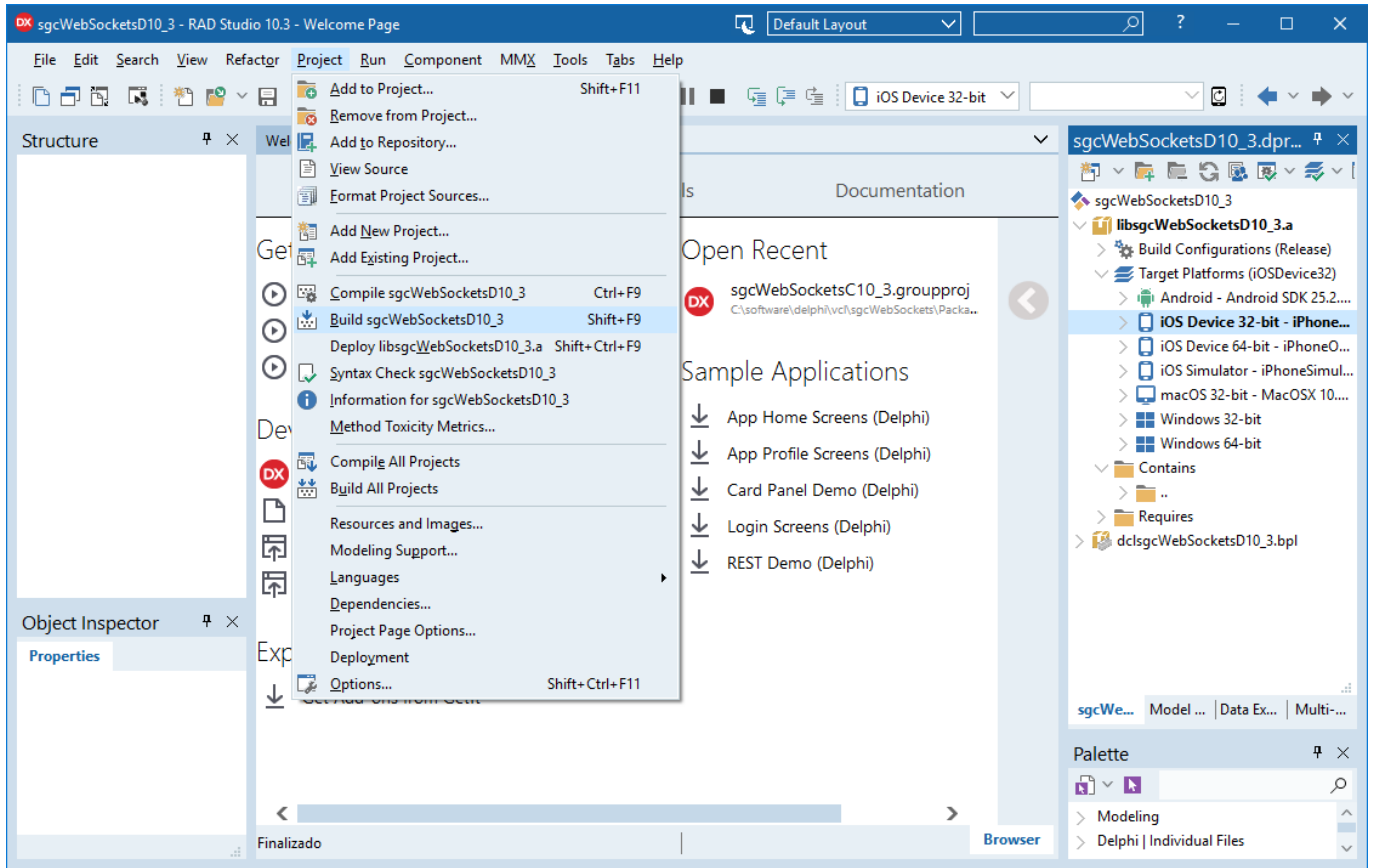
3. Target platform으로 win64를 선택하고 패키지를 빌드하십시오.



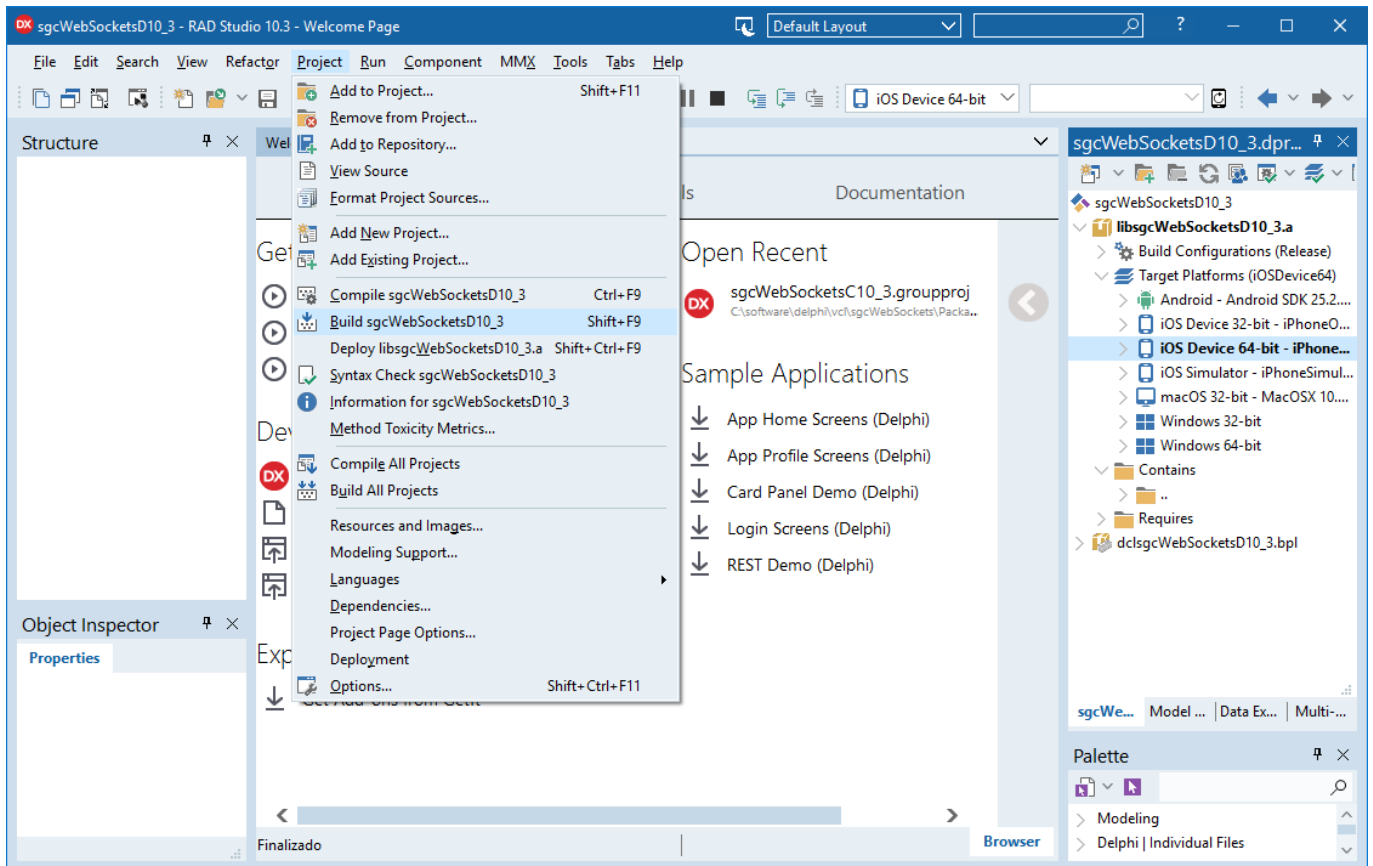
4. Target Platform으로 Android를 선택하고 패키지를 빌드하십시오.



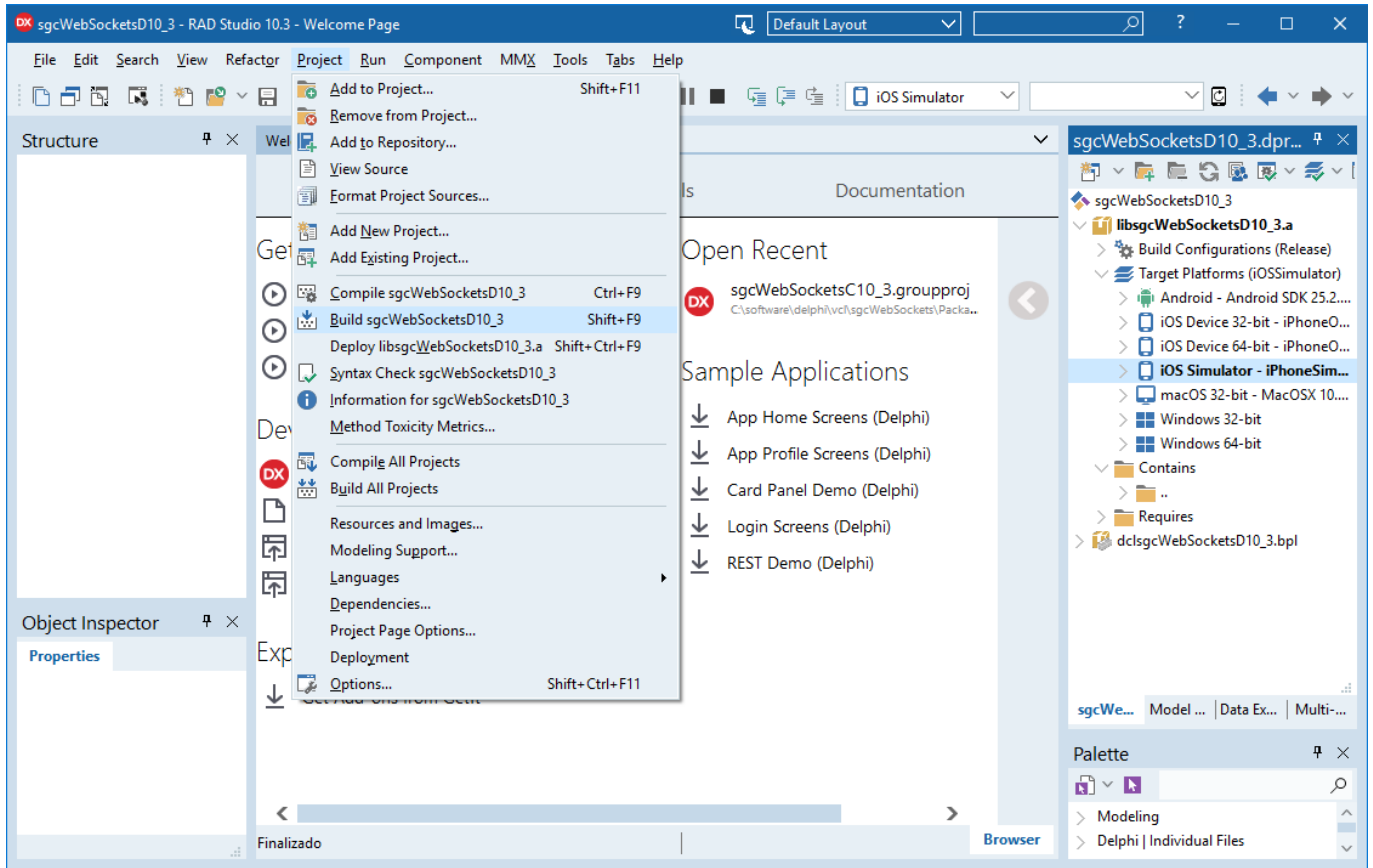
5. 대상 플랫폼으로 iOS Device 32 를 선택하고 패키지를 빌드하십시오.



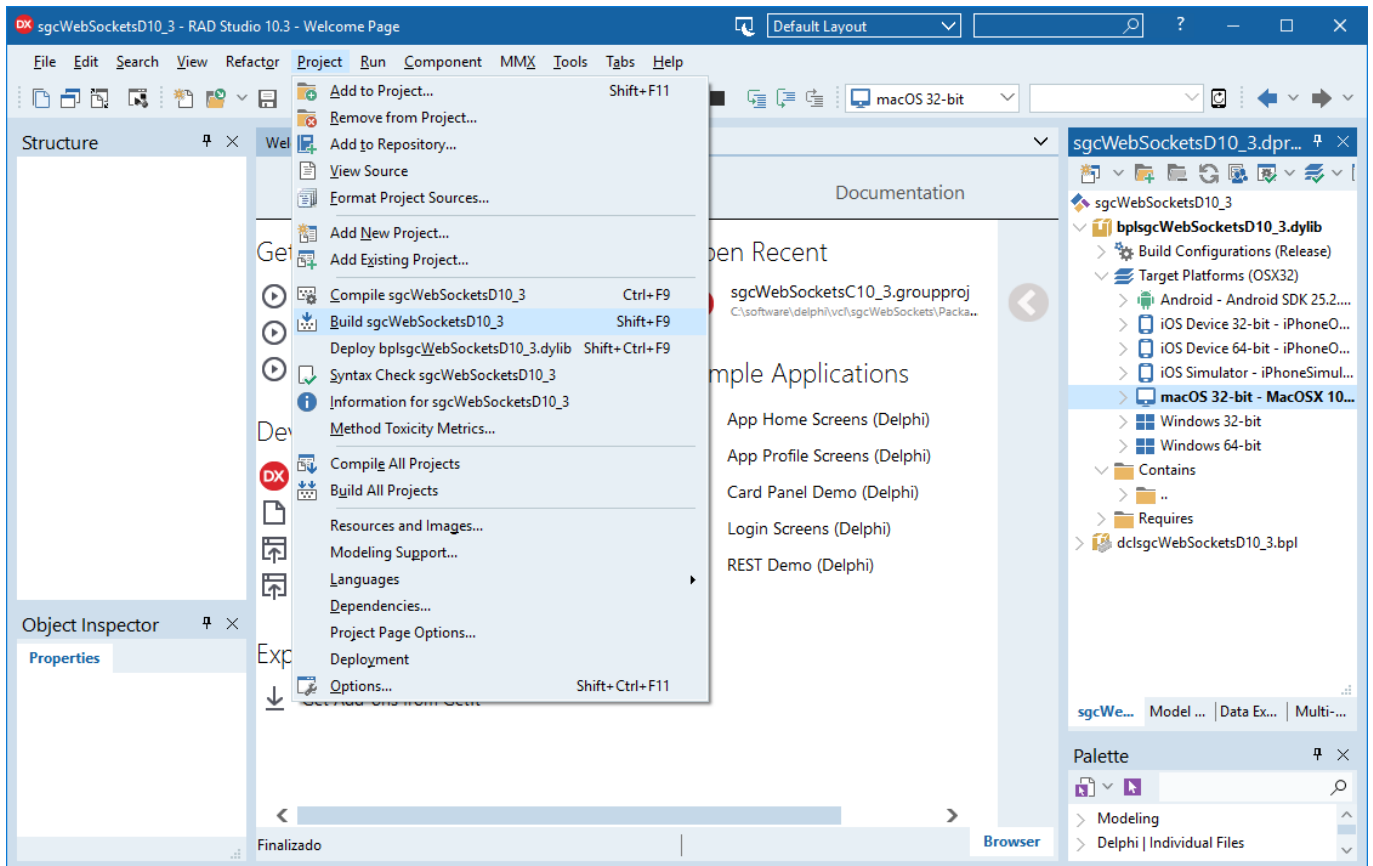
6. 대상 플랫폼으로 **iOS Device 64** 를 선택하고 패키지를 빌드하십시오.



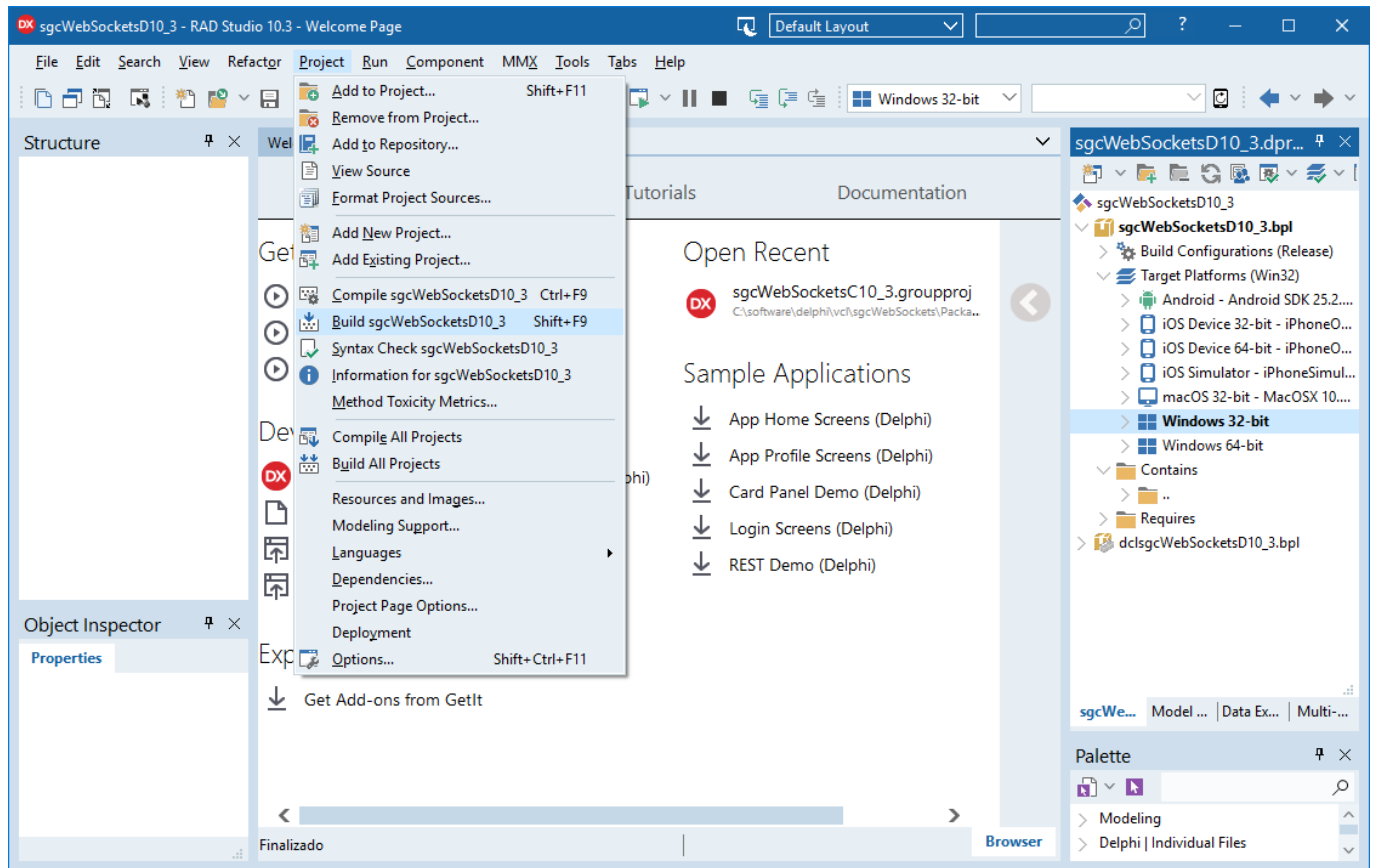
7. **iOS Device Simulator**를 대상 플랫폼으로 선택하고 패키지를 빌드합니다.



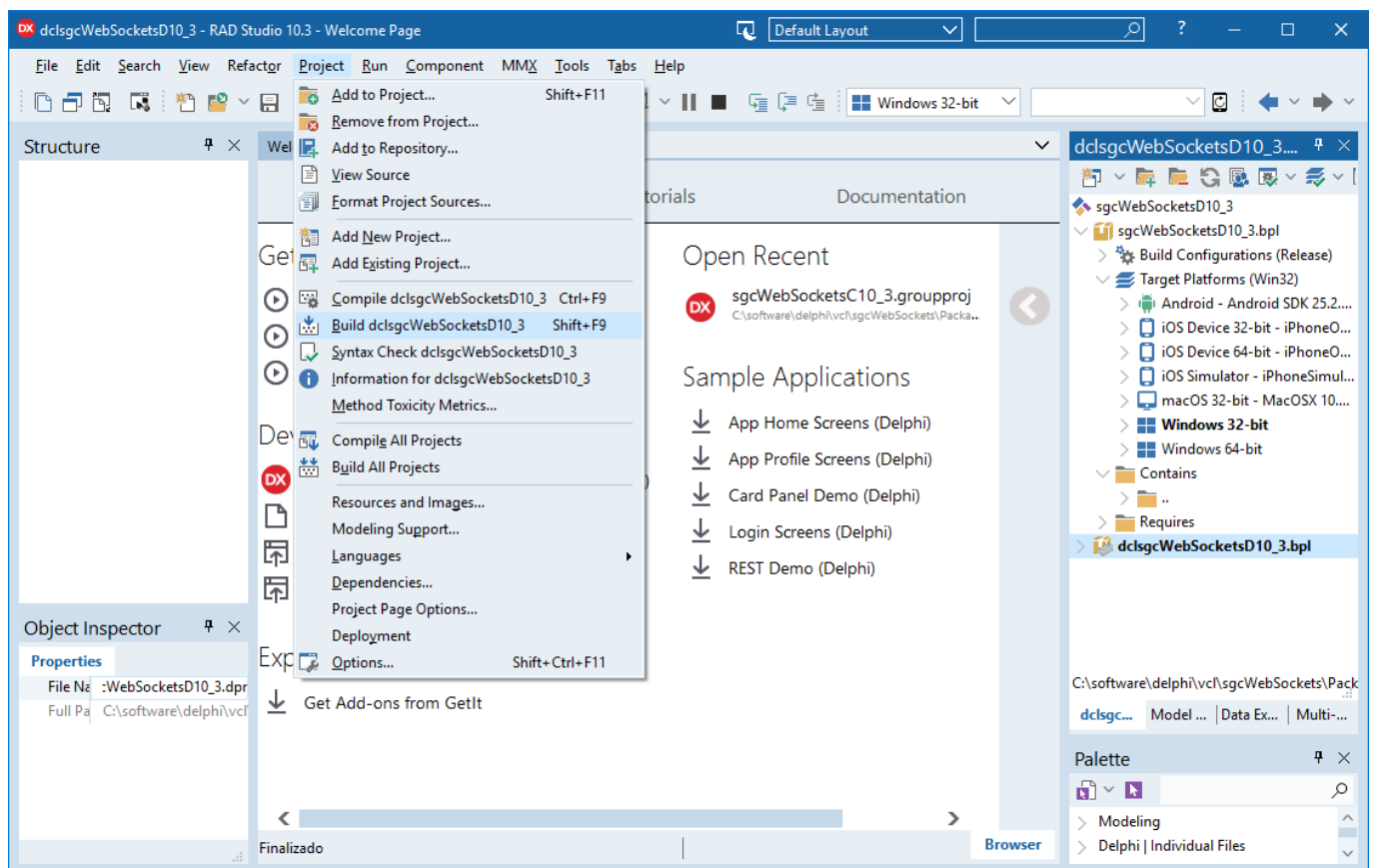
8. Target Platform으로 **MacOS 32**를 선택하고 패키지를 빌드하십시오.

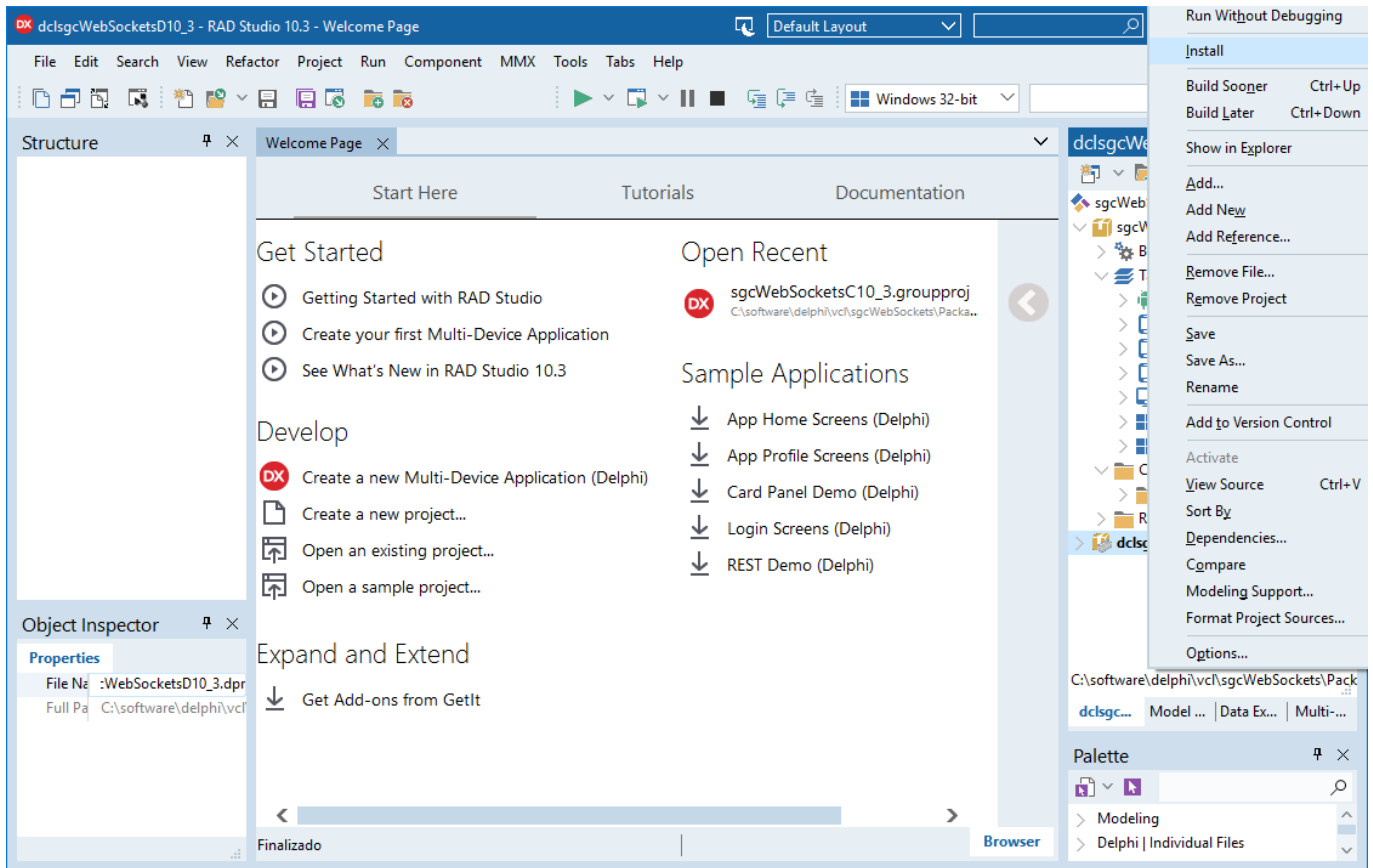


9. Target Platform으로 **Win32**를 선택하고 패키지를 빌드하십시오.

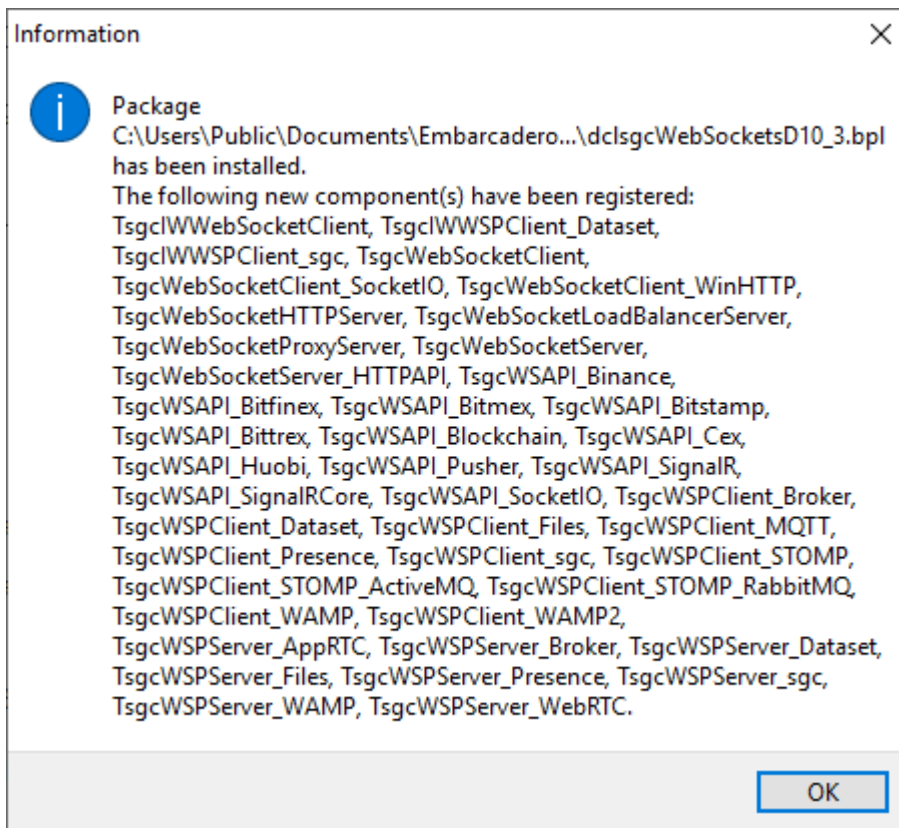


10. 모든 runtime 패키지가 컴파일되면, **design time 패키지**(이름이 dcl로 시작)를 선택하고 먼저 **build**한 다음 **install**하십시오(design time 패키지는 Win32만 대상 플랫폼으로 가짐).

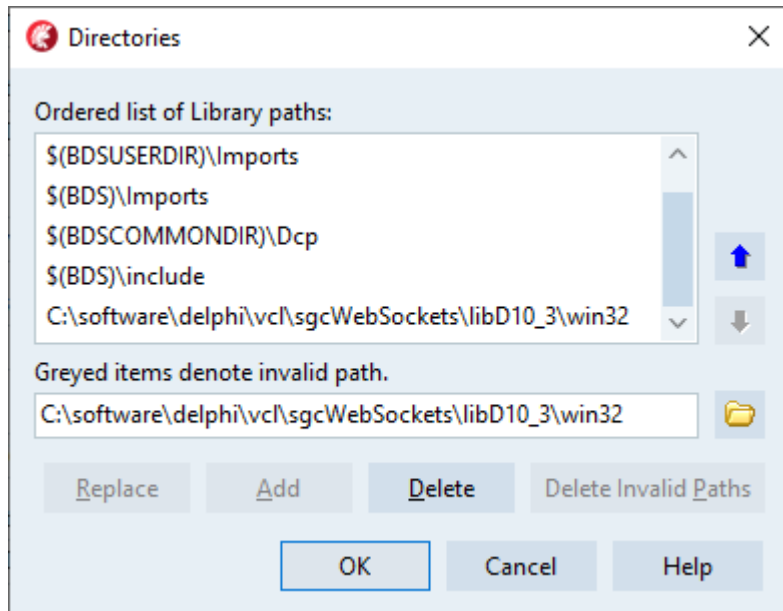




11. 설치가 성공하면 설치된 모든 구성 요소가 있는 메시지가 표시됩니다.



12. 그런 다음 컴파일된 파일이 있는 디렉토리를 **Rad Studio Library Path**에 추가하기만 하면 됩니다. 모든 Target Platform(win32, win64, osx64...)에 대해 이를 추가해야 합니다



* **Datasnap** 서버를 사용하는 경우, 이들은 설치할 수 없기 때문에 sgcWebSockets 패키지에 포함되지 않습니다. 런타임 전용 구성 요소입니다. 이 경우, 라이브러리 경로에 Source 폴더도 추가해야 합니다.

Install Errors

구성 요소를 설치할 때 일부 오류가 발생할 수 있습니다.

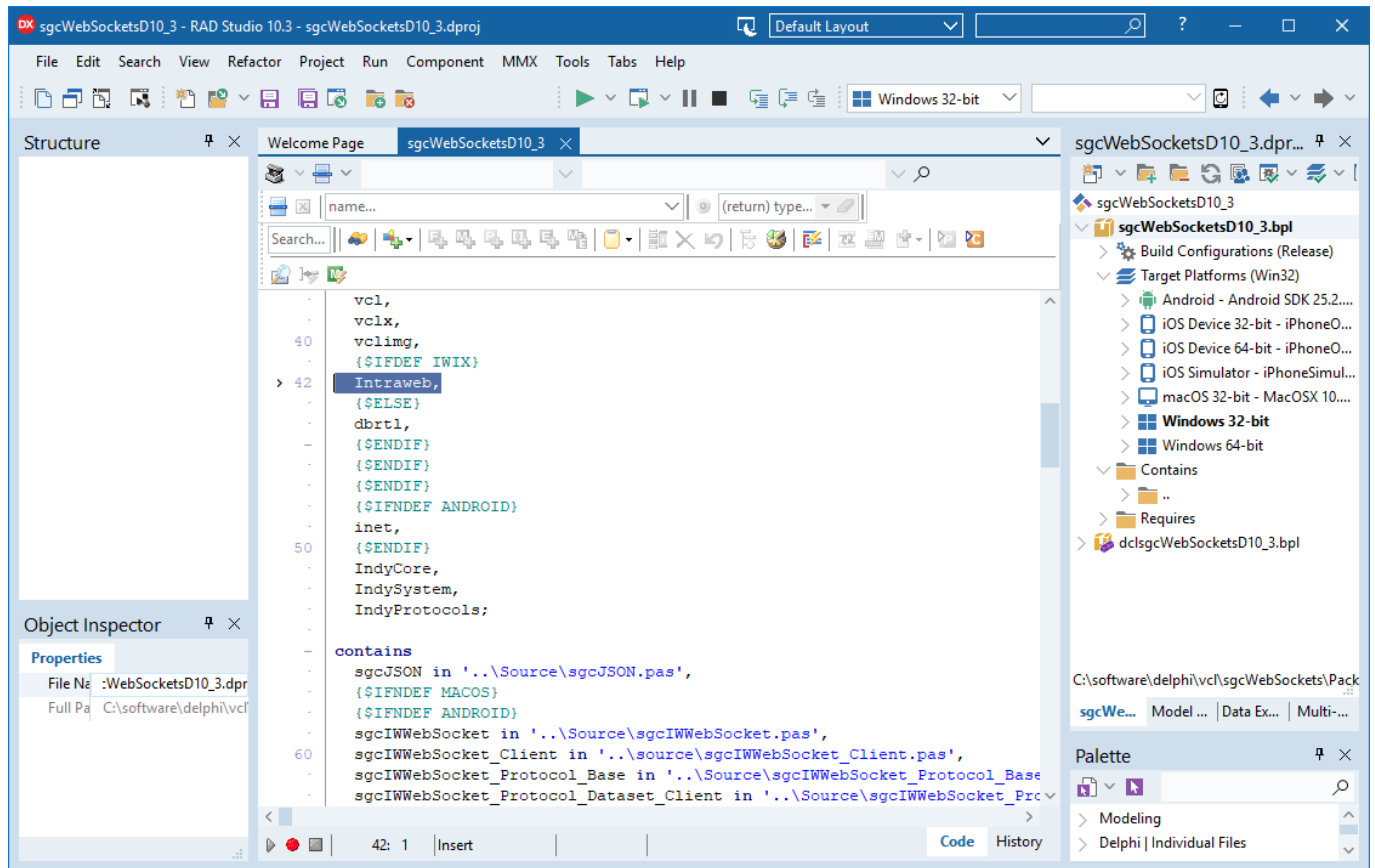
Intraweb package not found

sgcWebSockets는 Delphi와 함께 제공되는 기본 **Intraweb 버전**을 사용하여 컴파일됩니다. Intraweb이 설치되어 있지 않은 경우, sgcVer.inc 파일(Source 폴더에 위치)을 수정할 수 있습니다. Delphi 버전을 검색하고 **Intraweb에 대한 모든 컴파일러 define**을 주석 처리하십시오(IW로 시작). 예: Delphi 10.4의 경우 Intraweb에 대한 모든 컴파일러 define을 주석 처리하십시오

```
{ $IFDEF VER340 } { Delphi 10.4 }
{$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}
```

Intraweb가 설치되어 있지만 Delphi에 기본으로 제공되는 버전과 다른 경우, Intraweb 패키지의 이름이 다를 수 있습니다. 그런 다음 sgcWebSockets 런타임 패키지를 열고 프로젝트 소스에서 **Intraweb 이름 변경**을 하십시오.



Indy 패키지를 찾을 수 없음

sgcWebSockets는 IDE에 구성 요소를 설치하려면 **Indy**가 필요합니다. Trial 설치시 Delphi / CBuilder와 함께 제공된 Indy 라이브러리에 대해 컴파일되므로 다음과 같은 메시지가 표시되는 경우:

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

대부분의 경우, **최신 Indy 버전**이 있으므로, trial을 설치하려면 이 버전을 삭제하고 Delphi / CBuilder 설치를 사용하여 내장 indy 버전을 설치해야 합니다.

전체 소스 코드가 있는 경우 다음만 확인하면 됩니다:

1. 필요한 Indy 패키지: IndyCore, IndySystem, IndyProtocols. 더 새로운 Indy 버전이 있는 경우 패키지는 대부분 다른 이름 (버전 포함)을 가지므로, **"Component / Install Packages"** 메뉴에 액세스하여 Indy 패키지의 이름을 확인하고 프로젝트에서 그에 맞게 변경하십시오.

2. sgcWebSockets는 여러 Indy 버전을 지원하며, 모든 Indy 버전에 대해 컴파일할 수 있도록 컴파일러 정의가 있습니다. 소스 폴더에 있는 **sgcVer.inc**를 열고 Indy 버전에 맞게 변경하십시오(IdVers.inc Indy 파일의 gslVersion임). 일부 컴파일러 정의:

```

{$DEFINE INDY10_1}
{$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 CBuilder에서

CBuilder를 사용하여 프로젝트를 컴파일하고 이 오류가 발생하면, 프로젝트에 다음 옵션을 설정하십시오:

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

Project > Options > Packages > Runtime Packages
 "Link with runtime packages" 선택 해제

CBuilder Win64에서 Unable to find package import: sgcWebSocketsCXXX.bpi

win64용 런타임 패키지를 컴파일할 때는 Release와 Debug를 모두 컴파일해야 합니다.

Ambiguous reference System.ZLib.hpp and IdZLib.hpp CBuilder

sgcWebSockets Standard 및 Professional은 일부 구성 요소에 Indy를 사용하며, Indy는 ZLib 유닛을 사용하지 않고 자체 ZLib 복사본을 사용합니다: IdZLib, IdZLibHeaders... 프로젝트가 ZLib와 indy ZLib 유닛에 연결되므로, 컴파일할 때 이름이 동일하기 때문에 컴파일러가 어느 것이 올바른 참조인지 알지 못합니다. 두 가지 해결책이 있습니다:

1. System.ZLib.hpp에 대한 링크가 포함된 위치를 검색하여 삭제하거나 IdZLibHeaders.hpp 뒤로 이동하십시오.
2. 프로젝트 옵션에서 NO_USING_NAMESPACE_SYSTEM_ZLIB 또는 DELPHIHEADER_NO_IMPLICIT_NAMESPACE_USE 조건부 정의를 사용하여 System.Zlib.hpp 사용을 피하십시오

모호한 참조 System.ZLib.hpp 및 sgclZLib.hpp CBuilder

sgcWebSockets Enterprise는 일부 구성 요소에 대해 사용자 정의 Indy 버전을 사용하며 Indy는 ZLib 유닛을 사용하지 않고 자체 ZLib 복사본을 사용합니다: sgclZLib, sgclZLibHeaders... 프로젝트가 ZLib 및 indy ZLib 유닛에 링크하고 있으므로 컴파일할 때 이름이 동일하기 때문에 컴파일러가 올바른 참조를 알 수 없습니다. 두 가지 해결책이 있습니다:

1. System.ZLib.hpp에 대한 링크가 포함된 위치를 검색하여 sgclZLibHeaders.hpp 뒤로 삭제하거나 이동하십시오
2. 프로젝트 옵션에서 NO_USING_NAMESPACE_SYSTEM_ZLIB 또는 DELPHIHEADER_NO_IMPLICIT_NAMESPACE_USE 조건부 정의를 사용하여 System.Zlib.hpp 사용을 피하십시오

CBuilder 및 Android에서 Sgcwebsocket...에 대한 vTable 참조 정의되지 않음

오류를 수정하려면 다음 해결 방법을 사용하십시오. dcp/android 기본 폴더에 있는 libsgcwebsocketsC*.a 파일을 "Project/ Add to Project" 메뉴를 사용하여 프로젝트에 추가하십시오.

예: CBuilder 11의 경우 프로젝트에 "libsgcWebSocketsC11.a" 파일을 추가하십시오. 이 파일은 기본적으로 "C:\Users\Public\Documents\Embarcadero\Studio\12.0\DCP\Android\Release" 폴더에 있습니다.

Lazarus에서 체크섬이 변경됨

이 오류는 IDE를 빌드하는 프로필이 "Optimized IDE"가 아닌 경우 Lazarus에서 구성 요소를 설치하려고 할 때 발생할 수 있습니다. 체험판은 "Optimized IDE" 프로필로 컴파일됩니다.

Y에서 사용하는 X를 찾을 수 없음, 호환되지 않는 ppu

다음 해결 방법을 시도하십시오 메뉴 옵션에서 "Run / Clean up and rebuild".

Configure Install

소스 폴더에는 모든 Delphi, CBuilder 및 Lazarus IDE에 대한 모든 컴파일러 정의가 포함된 sgcVer.inc라는 파일이 있습니다.

여기에서 Intraweb, Indy... 구성을 사용자 지정할 수 있습니다. 일부 기능을 활성화/비활성화하려는 경우가 아니라면 일반적으로 변경할 필요가 없습니다.

컴파일러 define을 신중하게 변경하고 도움이 필요한 경우 당사에 문의하십시오.

모든 Delphi 버전에 대해 모든 컴파일러 define을 구성할 수 있는 섹션이 있습니다. Delphi 10.4 예제

```
{ $IFDEF VER340 } { Delphi 10.4 }
{$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

Indy 라이브러리에 대한 일부 컴파일러 define이 있습니다. 이는 설치된 Indy 버전에 따라 다르며, 기본적으로 Delphi와 함께 포함된 Indy 패키지용으로 구성되어 있습니다. Indy 버전은 IdVers.inc Indy 파일의 gsidVersion 매개변수입니다.

Intraweb

Intraweb이 설치되지 않은 경우, Intraweb에 대한 컴파일러 define(IW...로 시작하는 것)을 주석 처리하기만 하면 됩니다.

SGC_RESOURCES

기본적으로 sgcVer.inc는 {\$DEFINE SGC_RESOURCES}를 선언하며, 이는 런타임 패키지가 리소스 파일 sgcResources.RES를 임베드하게 합니다. 이는 sgcWebSockets 소스에 정적으로 링크되는 모든 실행 파일에 추가 바이트를 더합니다.

리소스에는 sgcWebSockets 서버가 원격 브라우저에 HTTP를 통해 전달할 수 있는(브라우저가 sgc 서버프로토콜을 사용할 수 있도록) 번들된 JavaScript 클라이언트 코드가 포함됩니다. 다음 경우에만 필요합니다:

- JS 클라이언트 번들을 브라우저에 전달하는 sgcWebSockets HTTP/WebSocket 서버를 사용합니다 (TsgcWSProtocol_JS * 기반 서버에 일반적임).

다음의 경우에는 필요하지 않습니다:

- Delphi / C++Builder WebSocket 클라이언트(데이터가 브라우저가 아닌 네이티브 코드에서 소비됨)입니다.
- 번들 JS 클라이언트를 제공하지 않는 서버(예: WebSocket 전용 서버, 네이티브 대 네이티브 서버).

임베이드 리소스를 제외하려면 sgcVer.inc에서 해당 줄을 주석 처리하십시오:

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

또는, source 에디션 [Install Setup](#)을 실행할 때, Options 페이지에서 **Include Resources** 옵션을 선택 해제하십시오. 그러면 설치 관리자가 패키지를 컴파일하기 전에 sgcVer.inc를 다시 작성합니다.

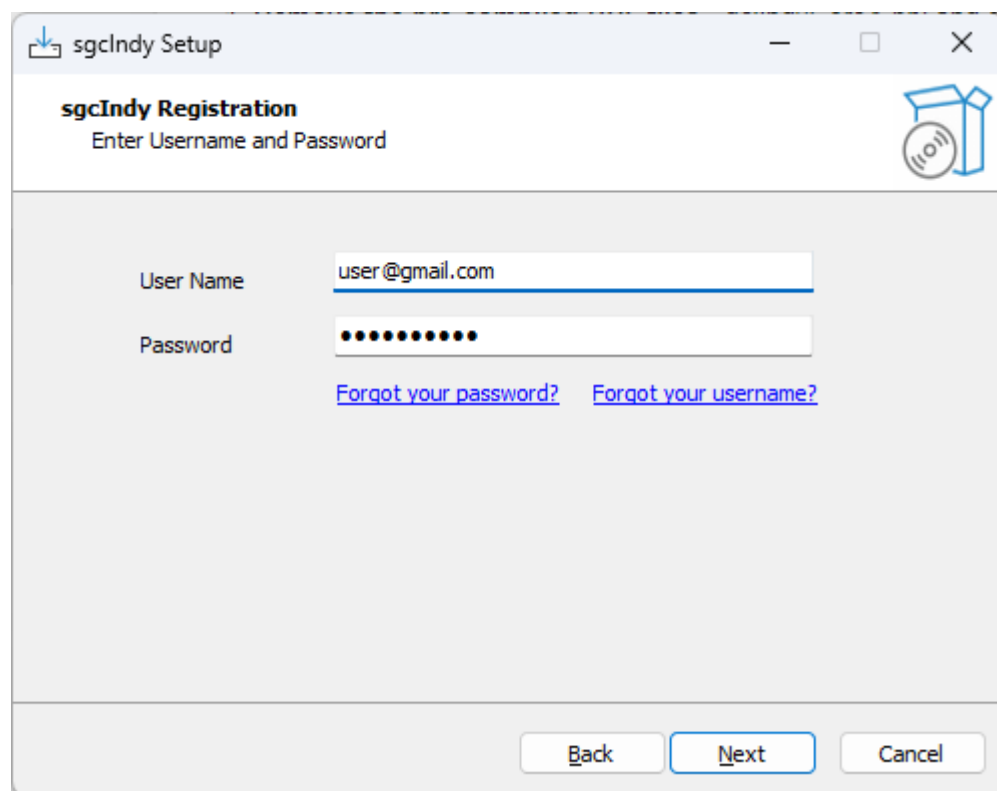
sgcIndy 패키지 설치

설정 설치

*최소 Windows Vista가 필요합니다(Windows 2000, XP 및 Server 2003은 지원되지 않음).

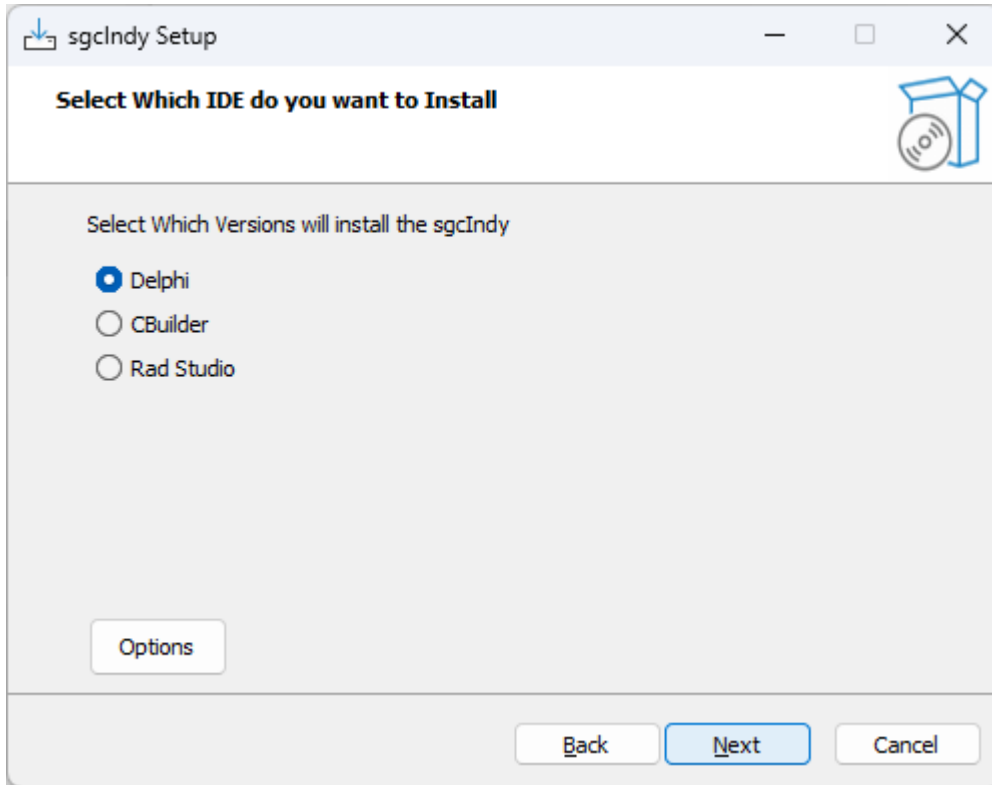
라이선스를 구매한 사용자는 설치 프로그램을 사용하여 sgcIndy 패키지를 설치할 수 있습니다. 패키지 설치를 위한 단계별 지침을 아래에서 확인하십시오.

- 설치 프로그램을 실행하십시오.
- 먼저 비공개 eSeGeCe 계정의 사용자 이름/비밀번호를 설정해야 합니다. 이것은 한 번만 입력하면 되며, 다음에 설정을 사용할 때 설치 프로그램이 최신 값을 읽습니다.

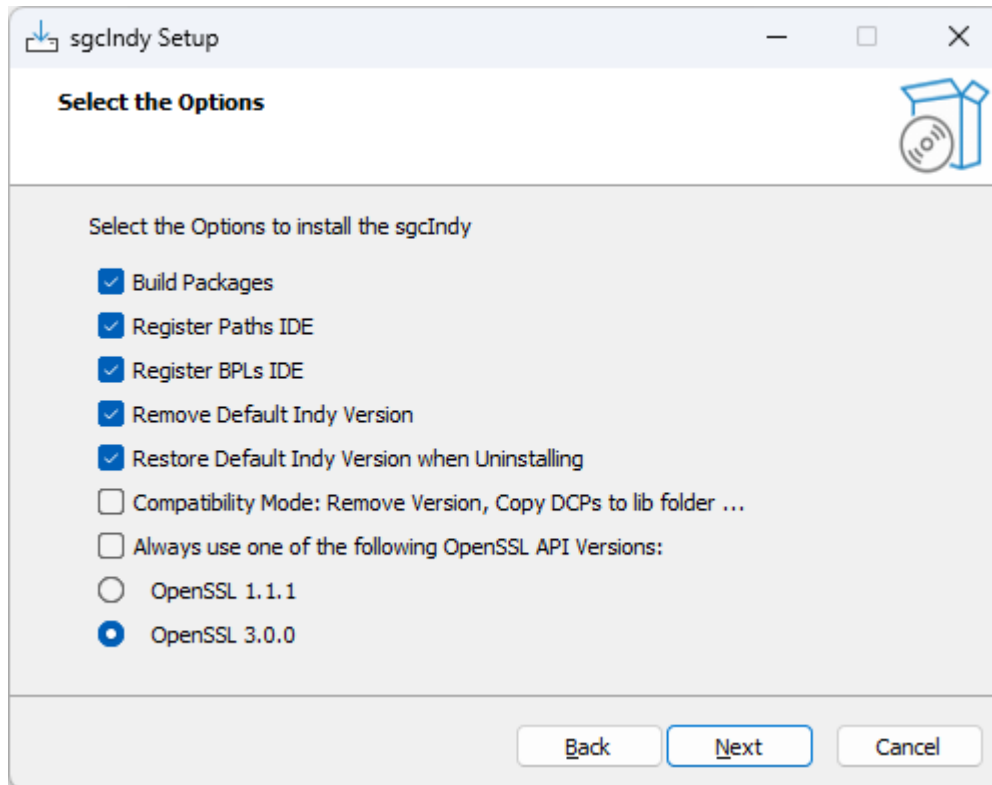


The image shows a Windows application window titled "sgcIndy Setup". Inside the window, there is a section titled "sgcIndy Registration" with the subtitle "Enter Username and Password". To the right of this section is a small icon of a box with a dollar sign. Below the subtitle, there are two input fields: "User Name" with the text "user@gmail.com" and "Password" with a masked password represented by dots. Below the password field are two blue links: "Forgot your password?" and "Forgot your username?". At the bottom of the window, there are three buttons: "Back", "Next" (which is highlighted with a blue border), and "Cancel".

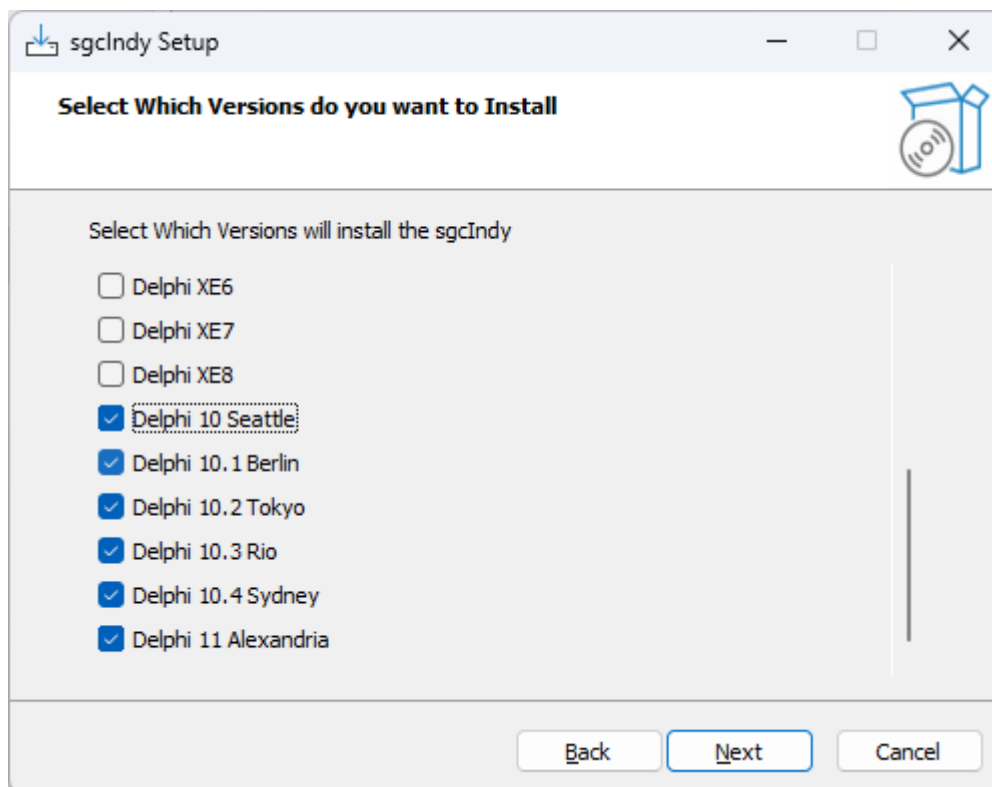
- 사용자가 성공적으로 로그인했다면 Delphi, CBuilder 또는 Rad Studio IDE 중 어디에 설치할지 선택하십시오.



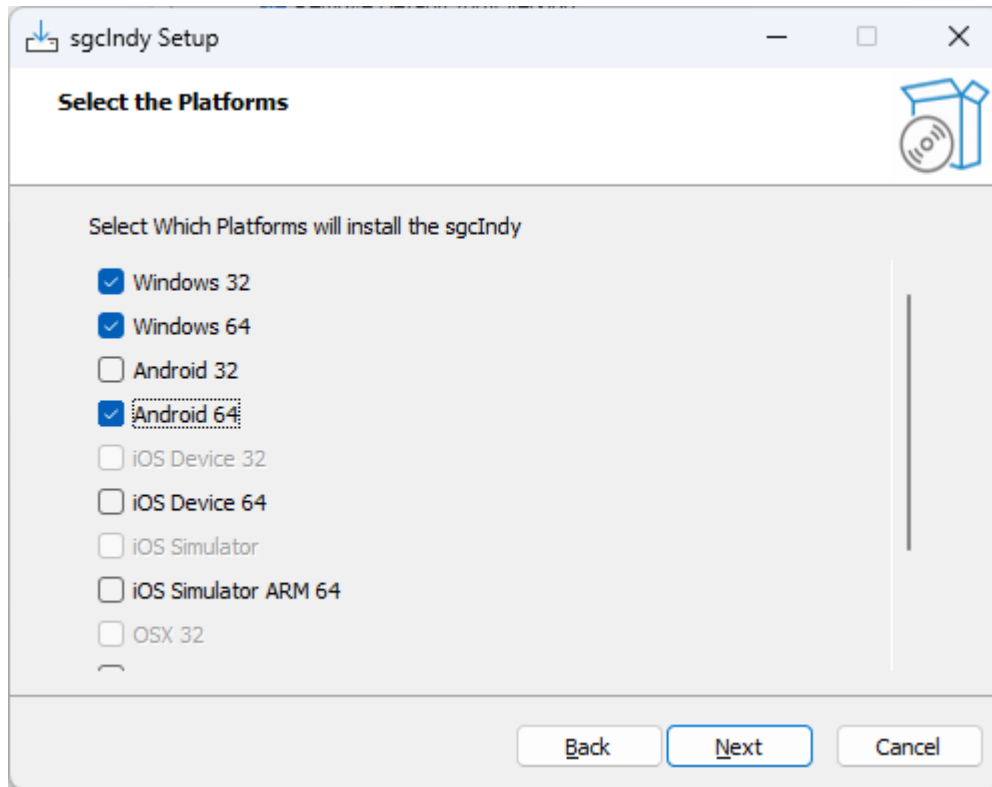
- 설치 프로그램을 사용할 때마다 사용자 정의할 수 있는 일부 옵션이 있습니다. 이러한 속성에 액세스하려면 **옵션** 버튼을 누르십시오.
 - **Build Packages:** 선택하면 설치 프로그램이 패키지를 빌드하려고 시도합니다.
 - **Register Paths IDE:** 선택하면 설치 프로그램이 IDE에 필요한 라이브러리 경로를 등록합니다.
 - **Register BPLs IDE:** 선택하고 설치 프로그램이 패키지를 성공적으로 빌드한 경우, 설치 프로그램이 디자인 타임 패키지를 IDE에 등록합니다.
 - **Remove Default Indy Version:** 선택하면 설치 프로그램이 먼저 Rad Studio와 함께 제공되는 표준 Indy 버전을 제거합니다.
 - **Restore Default Indy Version when Uninstalling:** 선택하면 패키지가 제거될 때 설치 프로그램이 제거된 표준 Indy 버전을 롤백합니다.
 - **Compatibility Mode:** 선택하면 dcp 파일이 버전 없이 컴파일되어 Embarcadero/lib 폴더에 복사됩니다. DevExpress와 같은 다른 패키지가 Indy 패키지를 사용하는 경우 이 옵션을 확인하십시오. 또한 설치된 sgclndy 버전으로 Embarcadero IP Abstraction 단위를 컴파일합니다.
 - **Always use of the following OpenSSL API Versions:** OpenSSL 1.1.1 또는 OpenSSL 3.0.0 API의 사용을 강제하려면 이 옵션을 확인하십시오.



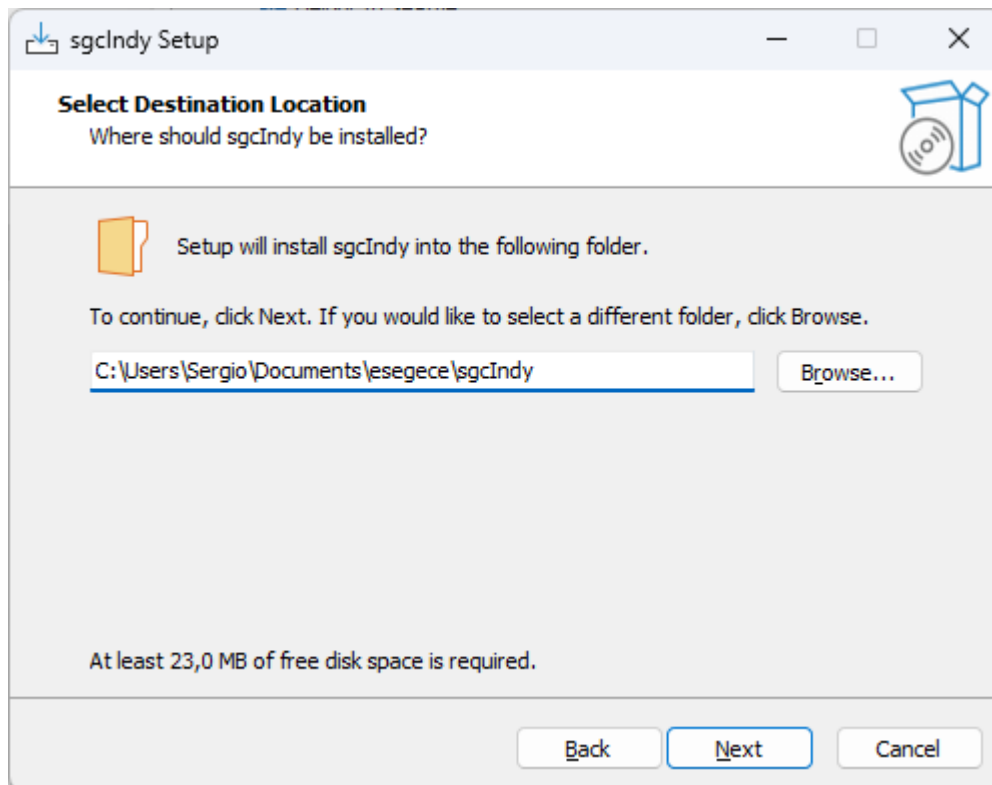
- 이제 설치할 IDE 버전을 선택할 수 있습니다. 설치 프로그램이 설치된 것으로 감지한 IDE 버전만 사용할 수 있습니다.



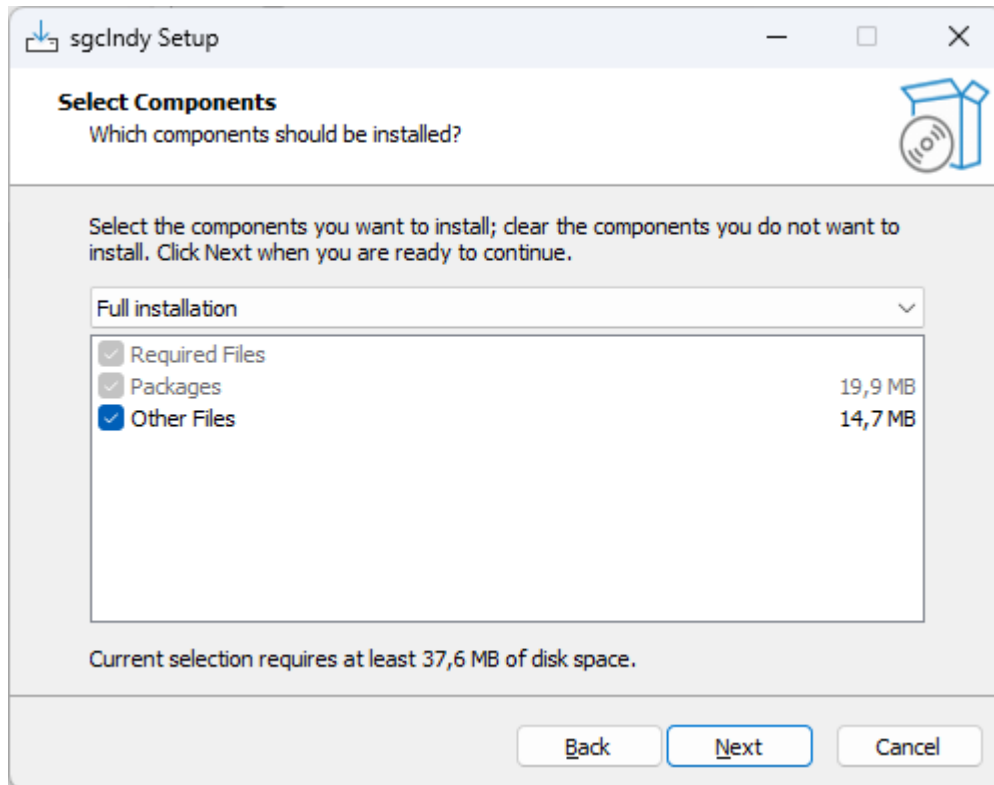
- 다음 단계는 Platform을 선택하는 것입니다.



- 패키지가 설치될 폴더를 선택하십시오. 패키지를 다시 설치하면 설치 프로그램이 이전 설치에서 선택한 동일한 폴더를 기본적으로 선택합니다.



- 설치할 구성 요소를 선택하십시오.



- 마지막으로, 파일을 추출하고, 패키지를 컴파일 및 설치하고, IDE에 필요한 경로를 등록합니다.

Install Errors

- MsBuild는 **Library Path**의 길이가 너무 길면 오류를 발생시킵니다. 이 문제를 해결하려면 library path에서 사용하지 않는 경로를 삭제해 보십시오. MsBuild에는 32K 문자 제한이 있습니다.

수동 설치

Indy가 이미 설치되어 있으면 먼저 제거해야 합니다.

1. 사전 컴파일된 **BPL 파일 제거** - dclIndyCoreX.bpl 및 dclIndyProtocolsX.bpl - "Components > Install Packages" 대화 상자를 통해 IDE에서 제거하십시오.
2. 그런 다음 **기존 바이너리를 모두 삭제하십시오**(IndySystemX., IndyCoreX., IndyProtocolsX., dclIndyCoreX. 및 dclIndyProtocolsX.*)와 존재하는 경우 Indy 10 소스 파일도 삭제하십시오.
3. IDE의 \bin, \lib 및 \source 폴더, \Indy 하위 폴더 및 OS 시스템 폴더에서 **파일을 확인**하십시오."

sgclndy 패키지를 빌드하려면 다음 중 하나를 수행할 수 있습니다

1. **(CBuilder만 해당)** CBuilder 버전에 해당하는 명령줄 **FULLC#.BAT** 스크립트를 사용하십시오.
2. IDE에서 개별 **DPK** 파일을 열고 다음 순서로 **컴파일**하십시오:
 1. IndySystemX.dpk (in Lib\System)
 2. IndyCoreX.dpk (in Lib\Core)
 3. IndyProtocolsX.dpk (in Lib\Protocols)
 4. dclIndyCoreX.dpk (in Lib\Core)
 5. dclIndyProtocolsX.dpk (in Lib\Protocols)

Indy 패키지가 빌드되면 **Components / Install Packages** 메뉴로 이동하여 Indy 디자인타임 패키지를 설치하십시오

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

마지막으로 IDE에서 sgclndy Packages에 대한 경로를 설정하십시오.

Configure ZLib

ZLib version: 1.2.12

sgcWebSockets는 WebSocket [Compression PerMessage Deflate](#) 확장이 활성화될 때 ZLib 압축을 사용합니다. 기본적으로 ZLib은 애플리케이션과 정적으로 링크되므로 ZLib 라이브러리를 배포할 필요가 없습니다.

특정 라이브러리를 사용하려면 프로젝트에 다음 Conditional Define을 추가하십시오:

SGC_DYNAMICLOAD_ZLIB

또는 sgclndy.inc 파일(소스 폴더에 위치)을 편집하여 다음 줄을 추가할 수 있습니다

```
{$DEFINE SGC_DYNAMICLOAD_ZLIB}
```

마지막으로 ZLib 라이브러리가 있는 위치를 설정해야 합니다. 이를 위해 다음 메서드를 사용하여 위치한 전체 경로(라이브러리 이름 제외)를 전달하십시오.

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

**이 구성은 Source code가 있는 sgcWebSockets Enterprise Edition에서만 유효합니다.*

QuickStart

WebSockets Components

새 WebSocket 서버 또는 WebSocket 클라이언트를 생성하는 것은 매우 간단합니다. 클래스의 새 인스턴스를 생성하고, Host / Port를 구성한 다음, Active = true 속성을 설정하여 프로세스를 시작하기만 하면 됩니다.

[QuickStart WebSockets](#)

HTTP Components

HTTP/2 프로토콜을 사용하면 HTTP/1 프로토콜을 사용하는 것보다 훨씬 빠른 HTTP 서버/클라이언트를 생성할 수 있습니다. HTTP/2 서버는 WebSocket 서버에 포함되어 있으며, HTTP/2 클라이언트는 HTTP/2 프로토콜을 구현하는 전용 구성 요소입니다.

[QuickStart HTTP](#)

스레딩 흐름

sgcWebSockets 구성 요소는 스레딩되며, 이는 **연결이 보조 스레드에서 실행됨**을 의미합니다. 기본적으로, 주요 이벤트는 메인 스레드에서 디스패치됩니다. 이는 디스패치할 이벤트 수가 적을 때 유용하지만, 더 나은 성능을 위해 **이벤트가 연결 스레드의 컨텍스트에서 디스패치되는** 구성 요소를 구성할 수 있습니다. 스레딩 흐름을 구성하는 방법을 설명하는 다음 문서를 읽으십시오:

[NotifyEvents 구성 방법](#)

애플리케이션을 빌드하는 방법

sgcWebSockets 라이브러리로 애플리케이션을 구축하는 것은 매우 쉽습니다. 다음 팁을 따르기만 하면 애플리케이션을 성공적으로 구축하는 데 도움이 됩니다.

[빌드](#)

Fast Performance Server

sgcWebSockets에는 2개의 서버 구현이 있습니다: 하나는 **Indy** 서버 기반이고 다른 하나는 **HTTP.SYS Microsoft Server** 기반입니다. 후자는 수천 개의 동시 연결을 처리해야 하는 고성능 서버에 권장됩니다. 서버 성능을 개선하는 방법을 설명하는 다음 문서를 확인하십시오.

[고성능 서버](#)

Memory Manager

적절한 메모리 관리자를 선택하면 애플리케이션의 성능을 향상시킬 수 있습니다. 여러 메모리 관리자 간의 비교를 보여주는 다음 글을 확인하십시오.

[Memory Manager](#)

OpenSSL

애플리케이션에 보안 연결이 필요한 경우, 일반적으로 **openssl 라이브러리**가 **통신을 암호화**하는 데 필요합니다. openssl 라이브러리로 애플리케이션을 성공적으로 구성하려면 다음 단계를 따르십시오.

[Configure OpenSSL](#)

Indy

Indy 라이브러리는 일부 sgcWebSockets 구성 요소의 기반으로 사용되며, sgcWebSockets Enterprise 에디션에는 openssl 1.1.1 및 openssl 3.0.0, ALPN 등을 사용할 수 있는 사용자 정의 indy 버전이 포함됩니다...

[Indy](#)

Linux (Lazarus)

Linux용 Lazarus 프로젝트를 컴파일하고 이 메시지가 나오면:

`Semaphore` 초기화 실패 (동시 스레드가 너무 많을 수 있음)

프로젝트 파일에 **cthreads** 유닛을 추가하기만 하면 됩니다.

QuickStart | WebSockets

Server WebSocket과 2가지 클라이언트 WebSocket 유형, 즉 Application Client와 Web Browser Client를 생성해야 하는 기본 예제로 시작하겠습니다.

WebSocket Server

1. 새 Window Forms 애플리케이션 생성
2. Form에 TsgcWebSocketServer를 놓으십시오.
3. Events 탭에서 OnMessage Event를 더블 클릭하고 다음 코드를 입력하십시오:

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

4. 폼에 Button을 놓고, 더블 클릭하여 이 코드를 입력하십시오:

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

WebSocket Client

1. 새 Window Forms 애플리케이션 생성
2. TsgcWebSocketClient를 폼에 놓고 서버에 연결하기 위해 Host 및 Port 속성을 구성합니다.
3. Form에 TButton을 놓고, 더블 클릭하여 다음 코드를 입력하십시오:

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

4. 폼에 Button을 놓고, 더블 클릭하여 이 코드를 입력하십시오:

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

Web Browser Client

1. 새 HTML 파일 생성
2. 텍스트 편집기로 파일을 열고 다음 코드를 복사하십시오:

```
<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>
```

사용자 지정 Host 및 Port에 맞게 이 파일의 host와 port를 교체해야 합니다!!

3. 파일을 저장하면 끝입니다. 기본 WebSocket 웹 브라우저 클라이언트를 구성했습니다.

How To Use

1. Server 애플리케이션을 시작하고 버튼을 눌러 WebSocket Server가 새 연결을 수신 대기하도록 시작하십시오.
2. 클라이언트 애플리케이션을 시작하고 button1을 눌러 서버에 연결하고 button2를 눌러 메시지를 보내십시오. 서버 측에서는 클라이언트가 보낸 텍스트가 포함된 메시지를 볼 수 있습니다.
3. 웹 브라우저(Chrome, Firefox, Safari 또는 Internet Explorer 10+)로 HTML 파일을 열고, Open을 눌러 연결을 열고 send를 눌러 서버에 메시지를 보내십시오. 서버 측에서는 웹 브라우저 클라이언트가 보낸 텍스트가 담긴 메시지를 볼 수 있습니다.

Linux Compiler

간단한 서버 예제(포트 5000에서 수신 대기).

```
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)

Linux용 Lazarus 프로젝트를 컴파일하고 이 메시지가 나오면:

Semaphore 초기화 실패 (동시 스레드가 너무 많을 수 있음)

프로젝트 파일에 **cthreads** 유닛을 추가하기만 하면 됩니다.

QuickStart | HTTP

HTTP/2 서버와 HTTP/2 클라이언트를 생성해야 하는 기본 예제부터 시작하겠습니다.

HTTP/2 Server

1. 새 Window Forms 애플리케이션 생성
2. 폼에 TsgcWebSocketHTTPServer를 놓으십시오.
3. Events 탭에서 OnCommandGet 이벤트를 더블 클릭하고 다음 코드를 입력하십시오:

```
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. 기본적으로 서버는 HTTP/1 연결만 활성화하므로, **HTTP2Options.Enabled = true** 속성에서 HTTP/2 옵션을 활성화한 다음 SSL 옵션을 구성하십시오. 보안 연결에는 [OpenSSL 라이브러리](#)가 필요합니다.

```
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. 폼에 Button을 끌어다 놓고 더블 클릭한 다음 이 코드를 입력하십시오:

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

HTTP/1 클라이언트

1. 새 Window Forms 애플리케이션 생성
2. Form에 TButton을 놓고, 더블 클릭한 다음 이 코드를 입력하십시오:

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

HTTP/2 Client

1. 새 Window Forms 애플리케이션 생성
2. Form에 TButton을 놓고, 더블 클릭한 다음 이 코드를 입력하십시오:

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


QuickStart | Threading Flow

sgcWebSockets 구성 요소는 스레드화되어 있습니다. 예를 들어 **TsgcWebSocketHTTPServer** (Indy 라이브러리 기반)는 모든 연결에 대해 하나의 스레드를 생성하는 반면, **TsgcWebSocketServer_HTTPAPI**(Microsoft HTTP.SYS 기반)는 스레드 풀을 실행하고 연결은 이 스레드 풀(최대 64개 스레드)에 의해 처리되며, **TsgcWebSocketClient**는 WebSocket 서버의 응답을 비동기적으로 처리하기 위해 자체 스레드를 실행합니다.

기본적으로 **NotifyEvents**라는 속성이 있으며, **neAsynchronous** 값을 가집니다. 이는 WebSocket 클라이언트가 메시지를 수신하면 이 메시지가 큐에 넣어지고 나중에 OS에 의해 메인 스레드에서 디스패치됨을 의미합니다. 이는 메시지를 많이 수신하지 않는 클라이언트에 잘 작동하며, 예를 들어 폼의 컨트롤을 업데이트하려는 경우 메인 스레드와 동기화할 필요가 없기 때문에 사용하기 쉽습니다.

그러나 서버 / 클라이언트가 짧은 시간 내에 여러 메시지를 처리해야 하는 경우, 이벤트가 연결 스레드 컨텍스트에서 디스패치되는 스레딩 흐름으로 변경하는 것이 좋습니다. 이렇게 하려면 **NotifyEvents** 속성을 **neNoSync**로 설정하기만 하면 됩니다. 이렇게 하면 예를 들어 클라이언트가 서버로부터 메시지를 받을 때 이 메시지가 보조 스레드 컨텍스트에서 디스패치되므로, 폼의 컨트롤을 업데이트해야 하는 경우 먼저 메인 스레드와 동기화한 다음 폼 컨트롤을 업데이트하십시오(폼 컨트롤은 스레드 안전하지 않기 때문). 공유 객체에 액세스하려는 경우에도 동일하게 적용되며, 자체 동기화 메서드를 구현해야 합니다.

Threading Flow Easy Mode (NotifyEvents = neAsynchronous) 및 Low Performance

이것은 기본 스레딩 흐름이며 일반적으로 데모 샘플에 사용됩니다. 초당 여러 메시지를 처리할 것으로 예상하지 않고 폼 컨트롤을 업데이트하거나 공유 객체에 액세스해야 하는 경우 이 모드를 선택하십시오.

NotifyEvents = neAsynchronous

스레딩 흐름 최고 성능 (NotifyEvents = neNoSync)

여러 메시지를 처리할 것으로 예상되어 고성능이 필요한 서버 구성 요소와 클라이언트에 이 스레딩 흐름을 설정하십시오. 이 구성을 사용하면 이벤트가 연결 스레드 컨텍스트에서 디스패치되므로, 폼 컨트롤을 업데이트하려면 먼저 메인 스레드와 동기화하십시오.

NotifyEvents = neNoSync

메인 스레드를 동기화하는 방법

TThread.Synchronize 또는 **TThread.Queue**를 호출하여 Main Thread와 동기화할 수 있습니다. 두 방법 모두 사용할 수 있으며, 하나를 선택하는 것은 동기화를 구현하려는 방식에 따라 달라집니다.

TThread.Synchronize

이 메서드는 블로킹이며, 이는 **Synchronize**를 호출할 때 코드가 메인 스레드와 동기화될 때까지 블로킹됨을 의미합니다.

TThread.Queue

이 메서드는 차단되지 않으므로 **queue**를 호출하면 메시지가 큐에 넣어지고 나중에 디스패치됩니다.

예제 코드

WebSocket Client에서 받은 메시지로 Memo를 업데이트합니다.

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

```
}    });
```

QuickStart | Build

sgcWebSockets 라이브러리로 애플리케이션을 빌드하는 것은 매우 쉽습니다. 구성 요소가 OpenSSL 라이브러리를 필요로 하는지 여부만 염두에 두십시오. 애플리케이션이 보안 연결을 필요로 하는 경우 OpenSSL 라이브러리를 배포해야 합니다(클라이언트 구성 요소에서 [Windows용 SChannel](#)을 사용하는 경우 제외).

Windows 애플리케이션의 경우, 애플리케이션이 위치한 동일한 폴더에 OpenSSL 라이브러리를 배포하는 것으로 충분합니다.

다른 personality에 대해서는 다음 글을 확인하십시오:

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

CBuilder DEBUG

CBuilder를 사용 중이고 sgcWebSockets 라이브러리를 디버깅하려면 다음 단계를 따르십시오:

- Project Options로 이동하십시오.
- Delphi Compiler > Compiling에서 Use debug .dcus를 활성화하십시오.
- C++ Linker에서 Dynamic RTL과의 링크를 비활성화합니다.
- Packages > Runtime Packages에서 Link with Runtime Packages를 비활성화하십시오.

그러면 VCL/RTL 소스 코드로 단계별 실행할 수 있습니다.

Build | OSX Application

sgcWebSockets 라이브러리로 OSX 애플리케이션을 빌드하려면 OSX 애플리케이션을 빌드하기 위한 Embarcadero 웹사이트를 따라야 합니다.

MacOS에 PASServer 설치

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

Developer ID 인증서 획득

유효한 Apple Developer Account로 <https://developer.apple.com>에 로그인하고 Certificates 메뉴에서 새 "Developer ID Application"을 만드십시오.

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

새 Apple Id 생성

그런 다음 <https://appleid.apple.com/account> 으로 이동하여 새 Apple Id를 생성하십시오.

프로비저닝 구성

마지막으로 **Project / Options / Provisioning** 메뉴를 열고 OSX 애플리케이션을 notarize하는 데 필요한 데이터를 채우십시오.

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

프로젝트에 일부 라이브러리가 필요한 경우, 메뉴 **Project / Deployment**에 포함하는 것을 잊지 마십시오. Remote Path를 "Contents\MacOS\"로 설정하십시오

Local Path	Local Name	Type	Configurat...	Platforms	Remote Path	Remote Name	Remote Status	Overwrite
☒ libcrypto.1.0.0.dylib	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_PROJECTCNS.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	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

이러한 라이브러리는 애플리케이션이 공증될 때 자동으로 서명되며, 다음 명령을 사용하여 라이브러리가 서명되었는지 확인할 수 있습니다:

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

openssl OSX [구성 방법](#)에 대해 자세히 읽어 보십시오.

빌드 | Android 애플리케이션

sgcWebSockets 라이브러리로 Android 애플리케이션을 빌드하려면 Embarcadero 웹사이트에서 Android 애플리케이션을 빌드하는 단계를 따라야 합니다.

Android 앱 생성

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

Project Deployment

프로젝트에 일부 라이브러리가 필요한 경우, **Project / Deployment** 메뉴에 포함하는 것을 잊지 마십시오. Remote Path를 ".\assets\internal"로 설정하십시오

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

openSSL Android 구성 방법에 대해 자세히 읽어보십시오.

Build | iOS Application

sgcWebSockets 라이브러리로 iOS 애플리케이션을 빌드하려면 Embarcadero 웹사이트의 iOS 애플리케이션 빌드 단계를 따라야 합니다.

MacOS에 PASServer 설치

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

iOS 개발 인증서 획득

유효한 Apple Developer Account로 <https://developer.apple.com>에 로그인하여 Certificates 메뉴에서 새 "iOS Development Certificate"를 생성하십시오.

iOS 앱을 위한 새 Identifier와 새 provisioning profile을 생성하십시오.

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

Bundle Identifier 구성

Project / Options / Application / Version Info 메뉴를 열고 Bundle Identifier를 설정하십시오

Version Info

Target
Release configuration - iOS Device 64-bit platform

☒ Include version information in project

Module version number

Major version: 1, Minor version: 0, Build: 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

프로젝트에 일부 정적 라이브러리가 필요한 경우, 이러한 라이브러리를 Embarcadero lib/iosDevice64 폴더에 복사하십시오:

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

openssl iOS 구성 방법에 대해 자세히 알아보십시오 [Configure openssl iOS](#)

Provisioning

마지막으로 **Project / Options / Deployment** 메뉴에서 인증서가 성공적으로 로드되었는지 확인하십시오.

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

고성능 서버

Indy 라이브러리 기반 서버

`TsgcWebsocketServer`와 `TsgcWebsocketHTTPServer`는 Indy 라이브러리를 기반으로 하므로, 모든 연결은 스레드에 의해 처리됩니다. 따라서 1000개의 동시 연결이 있으면 이러한 연결을 처리하기 위해 최소 1000개의 스레드가 있게 됩니다. 성능이 중요할 때, 성능을 높이고 서버 작업을 개선하기 위해 일부 "조정"을 수행해야 합니다. **sgcWebSockets 4.3.3부터 Indy 서버는 IOCP도 지원합니다. 자세히 읽으십시오.**

서버 성능을 높이려면 다음 팁을 사용하십시오.

1. Server 구성 요소 속성에서 **NotifyEvents := neNoSync**를 설정하십시오. 이는 이벤트가 연결 스레드 컨텍스트에서 발생하므로 동기화 메커니즘이 없음을 의미합니다. VCL 컨트롤이나 공유 객체에 액세스해야 하는 경우 자체 동기화 메커니즘을 사용하십시오.

2. 서버 구성 요소 속성에서 **Optimizations.Connections.Enabled := True**를 설정하십시오. 서버에 1000개 이상의 동시 연결이 있을 계획이고 Server.WriteData 메서드를 많이 호출하는 경우, 이 속성을 활성화하십시오. 기본적으로 Indy 연결 목록에 접근하는 것보다 검색이 빠른 캐시 목록에 연결을 저장합니다.

2.1 CacheSize: 빠른 캐시에 저장되는 연결 수입니다. 기본값 = 100.

2.2 GroupLevel: 첫 번째 문자로 분할된 여러 목록을 내부적으로 생성하므로 연결이 많은 경우 검색이 더 빠릅니다. 기본값 = 1.

3. Server 구성 요소 속성에서 **Optimizations.Channels.Enabled := True**로 설정하십시오. 이 속성을 활성화하면 채널이 목록에 저장되며, 여기서 검색이 이전 방법보다 빠릅니다.

4. Server 구성 요소 속성에 **Optimizations.ConnectionsFree.Enabled := True**를 설정하십시오. 이 속성이 활성화되면, 연결 풀감이 있을 때마다 TsgcWSConnection을 소멸시키는 대신 객체가 List에 저장되고 X초마다 이 list에 저장된 모든 객체가 소멸됩니다. 이 속성을 활성화하면 메모리 소비가 더 높아지므로, 짧은 기간에 많은 연결 풀감이 있는 경우 이 속성을 false로 설정하십시오.

4.1 Interval: 목록에 저장된 모든 연결 풀린 연결이 파괴되는 초 수. 기본값은 60입니다.

5. 기본적으로 sgcWebSockets는 공유 객체에 대한 액세스를 보호하기 위해 **Critical Sections**를 사용합니다. 그러나 critical section 대신 TMonitor 또는 SpinLocks를 사용할 수 있습니다. 다음 컴파일러 정의 중 하나로 프로젝트를 컴파일하기만 하면 됩니다

```
3.1 {$DEFINE SGC_SPINLOCKS}
3.2 {$DEFINE SGC_TMONITOR}
```

6. 최신 **FastMM4**를 사용하십시오. 다음에서 다운로드할 수 있습니다: <https://github.com/plieriche/FastMM4>
FastMM4는 매우 우수한 메모리 관리자이지만, 멀티스레드 애플리케이션에서 잘 확장되지 않는 경우가 있습니다. 애플리케이션에서 다음 컴파일러 define을 사용하십시오:

```
{$DEFINE UseReleaseStack}
```

그런 다음, 프로젝트 uses에 FastMM4를 첫 번째 유닛으로 추가하고 다시 컴파일하십시오. 높은 동시성 서버의 경우 성능 향상을 확인할 수 있습니다.

이 조정은 다음을 수행합니다: FreeMem 호출 중에 블록을 즉시 해제할 수 없는 경우, 해당 블록은 나중에 백그라운드 정리 스레드에서 또는 다음 FreeMem 호출 중에 해제될 블록 목록에 추가됩니다.

7. FastMM4보다 우수한 최신 **FastMM5**를 사용하십시오. 다음에서 다운로드할 수 있습니다: <https://github.com/plieriche/FastMM5>

이것은 FastMM4와 동일한 개발자의 새 버전으로, Delphi XE3 이상을 지원하며 Windows32 및 Windows64에서 사용할 수 있습니다.

FastMM5는 이중 라이선스이므로 GPL과 Commercial 두 가지 라이선스가 있습니다. 따라서 상업용 프로젝트에서 사용하려면 라이선스를 구매해야 합니다.

아래에서 1, 10, 100, 500 및 1000개의 동시 클라이언트를 사용하여 100,000개의 websocket 요청 및 응답을 수행하면서 FastMM4와 FastMM5 간의 성능을 비교하는 표를 확인하십시오. 멀티스레드 애플리케이션에서 FastMM5의 성능은 FastMM4를 사용하는 것보다 훨씬 우수합니다.

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

Indy Server Windows

sgcWebSockets Enterprise Edition은 Windows에서 **IOCP**를 지원하며, 이는 모든 연결에 대해 1개의 스레드를 생성하는 대신 스레드 풀이 모든 연결을 처리함을 의미합니다. IOCP를 활성화하려면 IOHandler를 IOCP로 설정하기만 하면 됩니다.

```
IOHandlerOptions.IOHandlerType = iohIOCP
```

IOHandlerOptions.IOCP 속성을 사용하면 IOCP 속성을 사용자 지정할 수 있습니다.

- **IOCPThreads:** 연결을 처리하는 데 사용되는 스레드입니다. 기본값은 0이며, 이는 프로세서 수를 사용하여 스레드가 자동으로 계산됨을 의미합니다(Delphi 7에서 Delphi 2007까지는 CPU count 함수가 지원되지 않으므로 이 값이 32로 설정됩니다).
- **WorkOpThreads:** 모든 연결에 대한 요청이 항상 동일한 스레드에 의해 처리되기를 원하면 0보다 큰 값을 설정하십시오. 기본적으로 IOCP 요청은 임의의 스레드에 의해 처리됩니다. 연결이 항상 동일한 스레드에 의해 처리되기를 원하면 0보다 큰 값을 설정하십시오. 예: WorkOpThreads = 32로 설정하면 서버가 32개의 스레드를 생성하고, 새 요청이 있을 때마다 연결이 이전에 이미 처리된 경우 동일한 스레드에 큐잉됩니다.

IOCP는 수천 개의 동시 연결을 처리하려는 경우에 권장됩니다.

Indy Server Linux

sgcWebSockets Enterprise Edition은 Linux에서 **EPOLL**을 지원합니다. 이는 모든 연결에 대해 스레드 하나를 생성하는 대신 스레드 풀이 모든 연결을 처리함을 의미합니다. EPOLL을 활성화하려면 IOHandler를 EPOLL로 설정하기만 하면 됩니다.

```
IOHandlerOptions.IOHandlerType = iohEPOLL
```

IOHandlerOptions.EPOLL 속성을 사용하면 EPOLL 속성을 사용자 지정할 수 있습니다.

- **EPOLLThreads:** 연결을 처리하는 데 사용되는 스레드입니다. 기본적으로 값은 0이며, 이는 프로세서 수를 사용하여 스레드가 자동으로 계산됨을 의미합니다.
- **WorkOpThreads:** 모든 연결에 대한 요청이 항상 동일한 스레드에 의해 처리되기를 원하는 경우 0보다 큰 값을 설정하십시오. 기본적으로 EPOLL 요청은 무작위 스레드에 의해 처리됩니다. 연결이 항상 동일한 스레드에 의해 처리되기를

원하면 0보다 큰 값을 설정하십시오. 예: WorkOpThreads = 32를 설정하면, 서버는 32개의 스레드를 생성하고 새 요청이 있을 때마다 연결이 이전에 이미 처리된 경우 동일한 스레드에 큐에 들어갑니다.

EPOOL은 수천 개의 동시 연결을 처리하려는 경우에 권장됩니다.

HTTP.SYS 기반 서버

[TsgcWebSocketServer_HTTPAPI](#) 구성 요소는 Microsoft HTTP API를 기반으로 하며 IOCP와 함께 작동하도록 설계되었으므로, 서버가 수천 개의 연결을 처리해야 할 때 권장되지만 Windows에서만 실행할 수 있다는 제한이 있습니다.

서버는 동일한 포트에서 **WebSocket** 및 **HTTP/2** 프로토콜을 처리할 수 있으며, 일부 엔드포인트만 처리하도록 구성할 수 있으므로 다른 구현과 함께 작동할 수 있습니다.

예제: 이 서버가 당사의 sgcWebSockets 라이브러리로 websocket 연결을 처리하도록 구성하고, 다른 구현/서드파티 또는 무엇이든 다른 엔드포인트를 사용하도록 할 수 있습니다.

- 엔드포인트: <https://server/ws>는 sgcWebSockets를 사용하여 WebSocket 프로토콜을 사용하는 연결을 처리합니다.
- 엔드포인트: <https://server/other>는 다른 라이브러리를 사용하여 연결을 처리합니다.

최신 **FastMM5**를 사용하십시오. 다음에서 다운로드할 수 있습니다: <https://github.com/plieriche/FastMM5>

이것은 FastMM4와 동일한 개발자의 새 버전으로, Delphi XE3 이상을 지원하며 Windows32 및 Windows64에서 사용할 수 있습니다.

FastMM5는 이중 라이선스이므로 GPL과 Commercial 두 가지 라이선스가 있습니다. 따라서 상업용 프로젝트에서 사용하려면 라이선스를 구매해야 합니다.

아래에서 1, 10, 100, 500 및 1000개의 동시 클라이언트를 사용하여 100,000개의 websocket 요청 및 응답을 수행하면서 FastMM4와 FastMM5 간의 성능을 비교하는 표를 확인하십시오. 멀티스레드 애플리케이션에서 FastMM5의 성능은 FastMM4를 사용하는 것보다 훨씬 우수합니다.

클라이언트	Windows	FMM4	FMM5	Difference
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%

Memory Manager

최근 Pierre le Riche가 개발한 새로운 버전의 FastMM이 출시되었으며, 새 버전은 **FastMM5**라고 불리고 멀티 스레드 애플리케이션의 성능을 개선하도록 다시 작성되었으며, 더 나은 속도 또는 더 적은 메모리 사용 등을 위해 구성할 수 있습니다.

Delphi **XE3 Compiler**부터 지원하며 **Windows32** 및 **Windows64**에서 사용할 수 있습니다.

FastMM5는 **이중 라이선스**이므로, 2개의 라이선스가 있습니다: **GPL** 및 **Commercial**. 따라서 상업적 프로젝트에서 사용하려면 라이선스를 구매해야 합니다. 자세한 내용은 여기에 있습니다

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

FastMM4에는 Maxim Masiutin이 개발한 **FastMM4-AVX**라는 새로운 fork가 있으며, 더 효율적인 동기화, 더 빠른 메모리 복사를 위한 AVX 명령어, 속도 향상 등과 같은 매우 흥미로운 기능을 추가합니다. FastMM4-AVX는 이중 라이선스입니다: MPL 및 GPL. 자세한 내용은 여기:

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

구성

우리 구성 요소로 성능을 테스트하기 위해, 새 windows 콘솔 애플리케이션 **sgcBenchmark.exe**가 만들어졌으며, 이는 우리 sgcWebSockets 구성 요소를 사용하여 모든 메모리 관리자의 성능을 측정하는 데 사용됩니다.

테스트는 매우 간단합니다. 클라이언트(또는 둘 이상의 클라이언트)가 서버에 연결하고, 서버에 메시지를 보내고, 서버가 동일한 메시지로 클라이언트에 응답합니다. 이것이 100.000번 반복됩니다. 테스트는 동시 클라이언트 수를 변경하며 반복되며, 먼저 1, 그다음 10, 100... 측정된 시간은 클라이언트가 보낸 첫 번째 메시지와 서버로부터 받은 마지막 메시지 사이의 경과 시간입니다(따라서 서버에 연결하는 데 사용된 시간은 측정되지 않음).

이 벤치마크는 Delphi 10.4.1과 함께 제공되는 기본 메모리 관리자, FastMM5 및 FastMM4-AVX를 사용한 성능을 비교합니다.

Benchmark Indy WebSocket Server

첫 번째 벤치마크에서 사용된 서버는 [Indy WebSocket Server](#)입니다. 이 서버는 Indy TCP Server를 기반으로 하므로 모든 연결이 스레드 1개를 생성합니다.

값은 밀리초 단위로 측정되므로, 예를 들어 기본 메모리 관리자를 사용하여 Windows32 플랫폼에서 1개의 클라이언트로 수행되는 첫 번째 테스트는 4135밀리초가 걸리고, FastMM5를 사용하면 4214밀리초, FastMM4-AVX를 사용하면 4823밀리초가 걸립니다. 계산된 백분율은 참조 값에 대한 것이며, 이 경우 delphi와 함께 제공되는 기본 메모리 관리자에 대한 것입니다. 백분율이 낮을수록 성능이 더 좋습니다.

벤치마크는 3번 수행되었으며 표시된 값은 벤치마크의 합계 / 3입니다.

벤치마크에 사용된 서버는 다음과 같습니다:

- Windows 2016 Server Datacenter
- 16 Virtual Processors
- 32 GB RAM
- 2.2 GHz

사용된 Delphi 버전은 Delphi 10.4.1이며, github 서버의 최신 FastMM5 및 FastMM4-AVX 버전을 사용했습니다.

벤치마크 결과를 아래에서 확인하십시오.

클라이언트	Platform	기본값 (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

두 번째 벤치마크에서 사용된 서버는 [HTTP.SYS WebSocket 서버](#)이며, 이 서버는 HTTP API Microsoft Framework를 기반으로 하고 연결은 스레드 풀에 의해 처리됩니다.

값은 밀리초 단위로 측정되므로, 예를 들어 Windows32 플랫폼에서 1개의 클라이언트로 수행되는 첫 번째 테스트는 기본 메모리 관리자를 사용할 때 5364밀리초, FastMM5를 사용할 때 5182밀리초, FastMM4-AVX를 사용할 때 5838밀리초가 걸립니다. 계산된 백분율은 기준 값, 이 경우 Delphi와 함께 제공되는 기본 메모리 관리자에 대한 것이며, 백분율이 낮을수록 성능이 더 좋습니다.

벤치마크는 3번 수행되었으며 표시된 값은 벤치마크의 합계 / 3입니다.

벤치마크에 사용된 서버는 다음과 같습니다:

- Windows 2016 Server Datacenter
- 16 Virtual Processors
- 32 GB RAM
- 2.2 GHz

사용된 Delphi 버전은 Delphi 10.4.1이며, github 서버의 최신 FastMM5 및 FastMM4-AVX 버전을 사용했습니다.

벤치마크 결과를 아래에서 확인하십시오.

클라이언트	Platform	기본값 (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%

벤치마크에 대한 설명

3가지 다른 메모리 관리자를 벤치마크한 후 얻은 결과에 대한 몇 가지 설명을 아래에서 확인하십시오:

- 단일 스레드 애플리케이션에서 사용하는 경우, FastMM4, FastMM5 및 FastMM4-AVX 간에 성능상 큰 차이가 없습니다.
- **FastMM5** 와 **FastMM4-AVX**는 멀티스레드 애플리케이션에서 훨씬 더 잘 작동합니다.
- FastMM5와 FastMM4-AVX 간의 차이는 적어도 이러한 벤치마크를 수행할 때 작습니다.
- **Windows 32** 벤치마크는 **Windows 64** 테스트보다 더 나은 성능을 보입니다. Windows 64 애플리케이션에서 FastMM5 또는 FastMM4-AVX를 사용하면 Windows 32보다 성능이 더 많이 향상됩니다.

하나 또는 다른 메모리 관리자를 선택하는 최종 결정은 프로젝트에 따라 다릅니다. 모든 조건에서 가장 잘 작동하는 단일 메모리 관리자는 없다고 생각하므로, 하나 또는 다른 것을 선택하기 전에 자신의 요구에 가장 적합한 성능을 보려면 테스트하고, 테스트하고, 다시 테스트하십시오.

OpenSSL

OpenSSL은 컴퓨터 네트워크를 통한 통신을 도청으로부터 보호하거나 상대방을 식별해야 하는 애플리케이션을 위한 소프트웨어 라이브러리입니다. 대다수의 HTTPS 웹사이트를 포함하여 인터넷 서버에서 널리 사용됩니다.

이 라이브러리는 보안 연결이 필요할 때 Indy Library 기반 구성 요소에 필요합니다. 애플리케이션에 OpenSSL이 필요한 경우, 애플리케이션을 배포하기 전에 파일 시스템에 필요한 파일이 있어야 합니다:

현재 sgcWebSockets는 다음 openssl 버전을 지원합니다: **1.0.2**, **1.1** 및 **3.0 ~ 3.3**.

Platform	API 1.0	API 1.1	API 3.*	정적/동적 링크
Windows (32비트 및 64비트)	libeay32.dll and ssleay32.dll	libcrypto-1_1.dll and libssl-1_1.dll	libcrypto-3.dll and 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 및 libssl.a	libcrypto.a 및 libssl.a	libcrypto.a 및 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

각 플랫폼에 대해 **OpenSSL을 구성**하는 방법은 아래에서 확인하십시오:

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

openssl 구성

sgcWebSockets Indy 기반 구성 요소를 사용하면 일부 OpenSSL 속성을 구성할 수 있습니다. 다음 속성에 액세스할 수 있습니다:

- 서버 구성 요소: `SSLOptions.OpenSSL_Options`.
- 클라이언트 구성 요소: `TLSOptions.OpenSSL_Options`.

API Version

표준 Indy 라이브러리는 **1.0.2 OpenSSL** 라이브러리만 로드할 수 있습니다. 이 라이브러리는 더 이상 사용되지 않으며 최신 OpenSSL 릴리스는 1.1.1 API를 사용합니다.

sgcWebSockets Enterprise를 사용하면 **1.1.1 openssl** 라이브러리를 로드할 수 있으며, 이 속성에서 어떤 openssl API 버전이 로드될지 구성할 수 있습니다. 프로세스당 하나의 API 버전만 로드할 수 있습니다(따라서 동일한 애플리케이션에서 openssl 1.0.2와 1.1.1 라이브러리를 혼합할 수 없습니다).

LibPath

이 속성을 사용하면 openssl 라이브러리의 위치를 설정할 수 있습니다. 이는 openssl 라이브러리의 위치를 설정해야 하는 Android 또는 OSX 프로젝트에 유용합니다.

다음 값을 수락합니다:

- **oslpNone:** 이 값은 라이브러리 경로 값을 설정하지 않습니다(기본값).
- **oslpDefaultFolder:** 이 값은 openssl 라이브러리의 기본 폴더를 설정합니다. 이 경로는 모든 퍼스널리티(windows, osx...)마다 다릅니다.

Load Additional OpenSSL Functions

기본적으로 정의되지 않은 추가 OpenSSL 함수를 로드하려면 콜백을 사용하십시오. [OpenSSL Load Additional Functions](#)에서 자세히 읽을 수 있습니다.

Ciphers

서버에서 TLS 1.2 및 1.3에 대한 지원을 제공하고 최상의 보안과 성능을 사용하려면 다음 구성을 사용하십시오:

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

그리고 다음 암호화 목록을 설정하십시오.

```
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
```

자체 서명 인증서

테스트 목적으로 자체 서명된 인증서를 사용할 수 있습니다. 자체 서명된 인증서를 생성하려면 다음 명령을 실행하기만 하면 됩니다:

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

cert.pem(인증서)과 key.pem(개인 키) 2개의 파일이 생성됩니다. 두 파일을 하나로 결합할 수 있습니다. 새 파일을 만들고 두 파일의 내용을 그 안에 복사하기만 하면 됩니다. 그러면 다음과 같은 구조가 됩니다:

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

일반적인 오류

SSL_GET_RECORD: wrong version number

이 오류는 서버와 클라이언트가 서로 다른 버전의 SSL/TLS 프로토콜을 사용하고 있음을 의미합니다. 이를 해결하려면 서버 및/또는 클라이언트 구성 요소에서 올바른 버전을 설정해 보십시오.

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

SSL3_GET_RECORD: decryption failed or bad record mac

일반적으로 이 오류는 다음 경우에 발생합니다:

1. 최신 OpenSSL 버전을 사용하고 있는지 확인하십시오. 너무 오래된 경우, 지원되는 최신 버전으로 업데이트하십시오.
2. 이 오류가 무작위로 나타나는 경우, 일반적으로 둘 이상의 스레드가 OpenSSL 연결에 액세스하기 때문입니다. `NotifyEvents = neNoSync`로 설정해 보십시오. 이는 이벤트(`OnConnect`, `OnDisconnect`, `OnMessage...`)가 스레드 연결의 컨텍스트에서 발생함을 의미하며, 이는 일부 동기화 문제를 방지하고 더 나은 성능을 제공합니다. 단점으로, 예를 들어 메시지를 수신할 때 폼의 시각적 컨트롤을 업데이트하는 경우, 시각적 컨트롤은 스레드 안전하지 않으므로 자체 동기화 메서드를 구현해야 합니다.

OpenSSL | Windows

32비트용 버전과 64비트용 버전이 있습니다. 이러한 라이브러리를 애플리케이션이 있는 동일한 폴더 또는 시스템 경로에 복사해야 합니다.

운영 체제가 Windows 32비트인 경우, System32 폴더에 복사하기만 하면 됩니다.

운영 체제가 Windows 64비트인 경우, 64비트 버전을 System32 폴더에, 32비트 버전을 SysWow64 폴더에 복사하십시오.

API 1.0

다음 라이브러리가 필요합니다:

- libeay32.dll
- ssleay32.dll

운영 체제가 Windows 32비트인 경우, System32 폴더에 복사하기만 하면 됩니다.

운영 체제가 Windows 64비트인 경우, 64비트 버전을 System32 폴더에, 32비트 버전을 SysWow64 폴더에 복사하십시오.

계정에서 최신 라이브러리를 다운로드할 수 있습니다(라이브러리는 외부 종속성이 없으며 디지털 서명되어 있습니다).

API 1.1

다음 라이브러리가 필요합니다:

Windows 32

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

Windows 64

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

운영 체제가 Windows 32비트인 경우, System32 폴더에 복사하기만 하면 됩니다.

운영 체제가 Windows 64비트인 경우, 64비트 버전을 System32 폴더에, 32비트 버전을 SysWow64 폴더에 복사하십시오.

계정에서 최신 라이브러리를 다운로드할 수 있습니다(라이브러리는 외부 종속성이 없으며 디지털 서명되어 있습니다).

API 3.*

다음 라이브러리가 필요합니다:

Windows 32

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

Windows 64

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

운영 체제가 Windows 32비트인 경우, System32 폴더에 복사하기만 하면 됩니다.

운영 체제가 Windows 64비트인 경우, 64비트 버전을 System32 폴더에, 32비트 버전을 SysWow64 폴더에 복사하십시오.

계정에서 최신 라이브러리를 다운로드할 수 있습니다(라이브러리는 외부 종속성이 없으며 디지털 서명되어 있습니다).

p12 인증서를 사용하는 경우 legacy.dll 라이브러리를 배포해야 합니다. [OpenSSL p12 인증서](#)에 대해 자세히 알아보십시오.

OpenSSL | OSX

최신 버전의 OSX에는 openssl 라이브러리가 포함되어 있지 않거나 너무 오래되었으므로, 애플리케이션과 함께 배포해야 합니다. 다음 단계를 사용하여 이러한 라이브러리를 배포하십시오:

- 프로젝트에서 Project/Deployment를 여십시오.
- 필요한 라이브러리를 추가하십시오.
- RemotePath = 'Contents\Macos\'로 설정하십시오.
- openssl LibPath를 기본 폴더로 구성하십시오:
 - Client.TLSOptions.OpenSSL_Options.LibPath = oslpDefaultFolder.
 - Server.SSLOptions.OpenSSL_Options.LibPath = oslpDefaultFolder.

API 1.0

다음 라이브러리가 필요합니다:

- libcrypto.dylib
- libssl.dylib

계정에서 최신 라이브러리를 다운로드할 수 있습니다.

API 1.1

다음 라이브러리가 필요합니다:

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

32비트용과 64비트용의 두 가지 버전이 있습니다. 이 라이브러리를 애플리케이션이 위치한 동일한 폴더에 복사해야 합니다. 계정에서 최신 라이브러리를 다운로드할 수 있습니다.

API 3.0

다음 라이브러리가 필요합니다:

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

64비트 버전만 제공됩니다. 이러한 라이브러리를 애플리케이션이 위치한 동일한 폴더에 복사해야 합니다. 계정에서 최신 라이브러리를 다운로드할 수 있습니다.

OSX 애플리케이션에 OpenSSL 라이브러리를 포함하는 경우, 애플리케이션이 Notarize된 후 라이브러리에 서명되며, 다음 명령을 사용하여 이를 확인할 수 있습니다:

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

OpenSSL 라이브러리를 사용하여 MacOSX64 애플리케이션을 빌드하는 방법을 보여주는 다음 동영상을 확인하십시오.

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

오류

클라이언트는 안정적인 ABI가 없으므로 버전이 없는 `libcrypto.dylib`를 로드해서는 안 됩니다.

MacOS Monterey+에서, openssl 라이브러리를 로드하려고 할 때 이 오류가 발생할 수 있습니다. 이 오류는 버전 없는 openssl 라이브러리(예: `libcrypto.dylib`)를 먼저 로드하려고 할 때 발생합니다.

이 오류를 수정하려면 **OpenSSL.Options.UnixSymLinks** 속성에 **oslsSymLinksDontLoad** 값을 설정하십시오. 이렇게 하면 버전 없는 openssl 라이브러리의 로드를 방지합니다.

OpenSSL | Android

최신 버전의 Android에는 openssl 라이브러리가 포함되어 있지 않거나 너무 오래되었으므로 애플리케이션과 함께 배포해야 합니다. 다음 단계를 사용하여 이러한 라이브러리를 배포하십시오:

- 프로젝트에서 Project/Deployment를 여십시오.
- 필요한 라이브러리를 추가하십시오.
- RemotePath = '.\assets\internal'을 설정하십시오.
- openssl LibPath를 기본 폴더로 구성하십시오:
 - Client.TLSOptions.OpenSSL_Options.LibPath = oslpDefaultFolder.
 - Server.SSLOptions.OpenSSL_Options.LibPath = oslpDefaultFolder.

API 1.0

다음 라이브러리가 필요합니다:

- libcrypto.so
- libssl.so

계정에서 최신 라이브러리를 다운로드할 수 있습니다.

Android 64bits에서 TLS 1.2를 사용하면 다음 오류가 발생할 수 있습니다:

INT_RSA_VERIFY:bad signature

이것은 API 1.1에서 수정된 OpenSSL 오류입니다.

이 오류를 방지하기 위해 TLS 1.0 또는 TLS 1.1을 사용해 볼 수 있습니다(서버가 여전히 이러한 암호화 방법을 지원하는 경우).

API 1.1

다음 라이브러리가 필요합니다:

- libcrypto.so
- libssl.so

계정에서 최신 라이브러리를 다운로드할 수 있습니다.

API 3.0

다음 라이브러리가 필요합니다:

- libcrypto.so
- libssl.so

계정에서 최신 라이브러리를 다운로드할 수 있습니다.

OpenSSL | iOS

64비트 iOS 디바이스에 OpenSSL을 설치하려면 libcrypto.a 및 libssl.a SSL 라이브러리 파일을 시스템에 복사해야 합니다. iOS OpenSSL .zip을 다운로드하고, 압축을 풀고, lib 디렉터리에서 .a 파일을 찾으십시오. libcrypto.a 및 libssl.a SSL 라이브러리 파일을 다음 디렉터리에 복사해야 합니다:

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

uses 절에 **sgcIdSSLOpenSSLHeaders_static**(또는 sgcWebSockets 에디션이 Enterprise 가 아닌 경우 IdSSLOpenSSLHeaders_static) 유닛을 추가하십시오.

파일을 배포해야 하는 경우 RemotePath = StartUp\Documents로 설정하고 파일을 로드하려면 다음을 사용하십시오(uses 절에 System.IOUtils 추가 필요):

TPath.GetDocumentsPath + PathDelim + <your filename>

openssl 라이브러리는 iOS에서 Project/Deployment 메뉴를 사용하여 배포해서는 안 됩니다.

API 1.1

IdCompilerDefines.inc 를 수정하고 IOS 섹션에서 SGC_OPENSSL_API_1_1을 활성화하십시오:

```
{ $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
// 정적 OpenSSL 라이브러리를 프로젝트에 추가한 다음
// IdSSLOpenSSLHeaders_static unit in their uses clause. It hooks up the
// IdSSLOpenSSLHeaders 유닛이 사용할 정적으로 링크된 함수...
{ $DEFINE STATICLOAD_OPENSSL }
// sgc--> openssl API 1.1에 대해 활성화
{ $DEFINE SGC_OPENSSL_API_1_1 }
{ $ENDIF }
{ $ENDIF }
```

계정에서 라이브러리를 다운로드할 수 있습니다.

API 3.0

IdCompilerDefines.inc 를 수정하고 IOS 섹션에서 SGC_OPENSSL_API_1_1과 SGC_OPENSSL_API_3_0을 활성화하십시오:

```
{ $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
// 정적 OpenSSL 라이브러리를 프로젝트에 추가한 다음
// IdSSLOpenSSLHeaders_static unit in their uses clause. It hooks up the
// IdSSLOpenSSLHeaders 유닛이 사용할 정적으로 링크된 함수...
{ $DEFINE STATICLOAD_OPENSSL }
```

```
// sgc--> openssl API 1.1에 대해 활성화
{$DEFINE SGC_OPENSSL_API_1_1}
// sgc--> enable for openssl API 3.0
{$DEFINE SGC_OPENSSL_API_3_0}
{$ENDIF}
{$ENDIF}
```

계정에서 라이브러리를 다운로드할 수 있습니다.

OpenSSL | Own CA Certificates

Github 게시물

CA 인증서를 로컬 머신에 추가한 후 웹 브라우저(예: Chrome)에서 신뢰할 수 있는, 자체 CA로 서명된 인증서를 생성하려면.

1. 프롬프트 없이 인증서를 생성하기 위한 구성 파일을 준비하십시오.

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. CA 개인 키와 인증서 생성(5년간 유효)

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

3. 웹 서버 비밀 키 및 CSR 생성

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

4. 인증서를 생성하고 자체 certificate authority로 서명합니다(1년 유효)

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

5. 출력 파일은 다음과 같습니다:

- `CA.cnf` → OpenSSL CA 구성 파일. 인증서 생성 프로세스 후 삭제할 수 있습니다.
- `CA_cert.pem` → [Certificate Authority] 인증서입니다. 이 인증서는 이 CA를 사용하여 생성된 모든 인증서를 신뢰하도록 하기 위해 브라우저 로컬 권한 저장소에 추가해야 합니다.
- `CA_cert.srl` → 임의의 일련 번호. 인증서 생성 프로세스 후 삭제할 수 있습니다.
- `CA_key.pem` → 새 [localhost] 인증서를 생성할 때 사용해야 합니다. 인증서 생성 과정 후에 삭제할 수 있습니다(재사용할 계획이 없고 `CA_cert.pem`도 마찬가지로인 경우).
- `localhost.cnf` → OpenSSL SSL 인증서 구성 파일입니다. 인증서 생성 프로세스 후에 삭제할 수 있습니다.
- `localhost.csr` → Certificate Signing Request. 인증서 생성 프로세스 후 삭제할 수 있습니다.
- `localhost_cert.pem` → SSL 인증서. 서버의 **SSLOptions.CertFile** 속성에 구성되어야 합니다.
- `localhost_key.pem` → 비밀 키. 서버의 **SSLOptions.KeyFile** 속성에 설치되어야 합니다.

OpenSSL | P12 Certificates

OpenSSL 3.0은 더 이상 사용되지 않거나 안전하지 않은 여러 알고리즘을 legacy provider라는 내부 라이브러리 모듈로 이동했습니다. 기본적으로 로드되지 않으므로 암호화 작업에 OpenSSL을 사용하는 앱(또는 해당 언어 런타임)은 인증서를 로드하거나 메시지 다이제스트를 생성할 때 등에 이러한 알고리즘을 사용할 수 없습니다...

legacy provider의 알고리즘에는 MD2, MD4, MDC2, RMD160, CAST5, BF(Blowfish), IDEA, SEED, RC2, RC4, RC5, DES(3DES 제외)가 포함됩니다.

보안상의 이유로, 이러한 legacy 알고리즘의 사용을 폐기하는 것이 강력히 권장됩니다.

애플리케이션이 RC2-40-CBC와 같은 레거시 암호로 암호화된 파일에 저장된 클라이언트 인증서를 사용하는 경우, openssl 프로그램을 사용하여 인증서 파일을 다시 암호화함으로써 "현대화"할 수 있습니다.

예를 들어, 로컬 컴퓨터에 client.p12(또는 client.pfx) 인증서 파일이 있는 경우:

```
$ openssl pkcs12 -legacy -in client.p12 -nodes -out cert-decrypted.tmp
(프롬프트가 표시되면 passphrase 입력)
```

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

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

내보내진 client-new.p12 인증서 파일은 이제 동일한 키를 포함하지만 AES-256-CBC를 사용하여 암호화됩니다.

sgcWebSockets 및 sgcIndy 패키지에 대한 구성은 아래에서 확인하십시오:

sgcWebSockets

- **OpenSSL.Options.Legacy.Enabled** 속성을 True로 설정하십시오.
- Legacy 라이브러리의 위치를 설정합니다.
 - **OpenSSL.Options.Legacy.LibPath:** 여기에서 legacy 라이브러리가 위치한 곳을 구성할 수 있습니다
 - **oslpNone:** 기본값입니다. legacy 라이브러리는 바이너리가 있는 동일한 폴더 또는 알려진 경로에 있어야 합니다.
 - **oslpDefaultFolder:** 모든 IDE personality에 대해 라이브러리가 위치해야 하는 곳으로 legacy 라이브러리 경로를 자동으로 설정합니다.
 - **oslpCustomFolder:** 이 옵션이 선택된 경우 LibPathCustom 속성에서 전체 경로를 정의하십시오.
 - **OpenSSL.Options.Legacy.LibPathCustom:** LibPath = oslpCustomFolder인 경우 legacy 라이브러리가 위치한 전체 경로를 여기에 정의하십시오.

sgcIndy

- **SSLOptions.Legacy** 속성을 True로 설정하십시오.
- 서버나 클라이언트를 시작하기 전에 legacy.dll 라이브러리가 위치한 경로를 설정하십시오. **IdOpenSSLSetOSSLPPath** 함수를 사용하고 경로를 인수로 전달하십시오.

OpenSSL | Verify Certificate

OpenSSL을 사용하고 Verify Certificate 옵션을 설정할 때 다음 오류가 나타날 수 있습니다:

Error connecting with SSL.error:80000002:system library::No such file or directory.
OnVerifyPeer 이벤트를 처리하고 Error 매개변수의 값이 20인 경우, 오류는 다음을 의미합니다:

X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY

이 오류의 주된 이유는 원격 서버가 제시한 하나 이상의 인증서가 애플리케이션의 인증서 저장소에 없기 때문입니다. 이를 해결하려면 RootCertFile 속성을 사용하여 CA 파일이 위치한 경로를 설정할 수 있습니다. 없는 경우, 예를 들어 mozilla에서 다운로드할 수 있습니다.

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

RootCertFile을 설정한 후 이전 오류는 사라져야 합니다.

OpenSSL | Load Additional Functions

기본적으로 Indy는 통신을 암호화하는 데 필요한 가장 일반적인 OpenSSL 함수를 정의하지만, 때로는 암호화, 서명 등을 위해 더 많은 함수가 필요합니다. `IdOpenSSLSetLoadFuncsCallback` 메서드를 사용하여 추가 OpenSSL 함수를 동적으로 로드하는 콜백을 할당할 수 있습니다.

IdOpenSSLSetLoadFuncsCallback

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

이것은 콜백 역할을 하는 프로시저 유형이며, 세 개의 매개변수를 받습니다:

- **hIdSSL**: TIdLibHandle - 로드된 SSL 라이브러리에 대한 핸들.
- **hIdCrypto**: TIdLibHandle - 로드된 Crypto 라이브러리에 대한 핸들입니다.
- **FailedLoadList**: TStringList - 로드에서 실패한 함수 목록입니다.

이 콜백의 목적은 OpenSSL 함수가 로드될 때 사용자가 사용자 지정 처리를 수행할 수 있게 하는 것입니다. 예: 실패한 함수 로딩 또는 오류 처리.

IdOpenSSLSetUnLoadFuncsCallback

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

SSL 함수를 언로드하기 위한 콜백 역할을 합니다. OpenSSL 라이브러리가 언로드될 때 정리 작업을 수행하는 데 유용합니다.

사용자 정의 **function**을 로드하는 방법

아래에서 콜백을 사용하여 함수 `EVP_PKEY_CTX_set_rsa_padding`를 로드하는 방법에 대한 간단한 예제를 찾을 수 있습니다.

```
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* FailedLoadList) {
    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

Indy 라이브러리는 TCP/UDP/RAW 소켓뿐만 아니라 SMTP, POP3, IMAP, NNTP, HTTP, FTP를 비롯한 100개 이상의 상위 수준 프로토콜을 지원하는 오픈 소스 클라이언트/서버 통신 라이브러리입니다. Indy는 Delphi로 작성되었지만 C++Builder 및 FreePascal에서도 사용할 수 있습니다. sgcWebSockets는 일부 구성 요소의 기반으로 Indy를 사용하며, 다양한 sgcWebSockets 에디션은 Indy 라이브러리를 서로 다르게 사용합니다.

sgcWebSockets는 ALPN 사용이 필요한 HTTP/2 같은 프로토콜을 지원하고, openssl 1.1.1 또는 openssl 3.0.0을 사용하여 TLS 1.3을 사용할 수 있는 등의 기능을 지원합니다. 이러한 기능은 표준 Indy 라이브러리에서 지원되지 않으므로, sgcWebSockets Enterprise edition에는 이러한 기능을 지원하는 사용자 지정 Indy 라이브러리가 포함되어 있습니다. IDE에서 표준 Indy 라이브러리를 제거하지 않기 위해, 필요한 Indy 파일은 접두사 "sgc"를 추가하여 이름이 변경됩니다. 예를 들어 "IdGlobal" 유닛은 "sgcIdGlobal"로 이름이 변경됩니다. 이렇게 하면 두 버전이 문제없이 공존할 수 있습니다.

모든 sgcWebSockets 에디션에서 사용되는 Indy 버전을 아래에서 확인하십시오:

sgcWebSockets Edition	Indy 버전
STANDARD	Standard
PROFESSIONAL	Standard
ENTERPRISE	Custom

sgcWebSockets 패키지가 분할되기 전의 오래된 라이선스인 "Registered" 라이선스를 가진 고객은 다음 sgcWebSockets 패키지 버전을 찾을 수 있습니다:

sgcWebSockets Edition	Indy 버전
sgcWebSockets	Standard
sgcWebSockets min	Standard
sgcWebSockets min Indy*	Custom

*sgcWebSockets_min_indy는 sgcWebSockets Enterprise 에디션과 동일합니다.

사용자 정의 Indy 버전의 사용은 source 폴더에 있는 "sgcVer.inc" 파일에 정의되어 있습니다. 이 indy 버전의 사용을 활성화 하거나 비활성화하는 "SGC_CUSTOM_INDY"라는 컴파일러 define이 있습니다. Enterprise Edition을 사용하고 사용자 정의 indy 사용을 비활성화하려면 다음 컴파일러 define을 삭제하기만 하면 됩니다:

```
{ $DEFINE SGC_CUSTOM_INDY }
```

물론, SGC_CUSTOM_INDY를 활성화했지만 소스 폴더에 필요한 사용자 지정 indy 버전 유닛이 없는 경우, 이 컴파일러 define은 작동하지 않습니다.

sgcIndy 패키지

사용자 지정 Indy 버전의 사용은 sgcWebSockets 구성 요소로 제한되지 않습니다. 일부 고객은 예를 들어 SMTP와 같은 표준 Indy 구성 요소에서 이 사용자 지정 Indy 버전의 새 기능을 사용하기를 원합니다. 그래서 이메일을 보낼 때 TLS 1.3을 사용하거나 FTP 서버를 사용할 수 있습니다. eSeGeCe ALL-ACCESS 에디션은 이러한 모든 기능이 있는 추가 전체 Indy 패키지를 제공합니다. "sgcIndy package"라고 하는 이 패키지에는 openssl 1.1.1 및 openssl 3.0.0을 지원하는 전체 Indy 라이브러리가 포함되어 있습니다. 따라서 먼저 IDE에 설치된 현재 Indy 라이브러리를 제거한 다음 이 버전을 설치해야 합니다. sgcIndy 패키지를 설치하는 프로세스는 다른 Indy 라이브러리와 정확히 동일합니다(여기서는 유닛 이름이 변경되지 않습니다).

openssl 라이브러리를 사용하려면, OpenSSL 라이브러리를 로드하기 전에 전역 변수 OPENSSL_API_VERSION을 원하는 OpenSSL API 버전으로 설정하기만 하면 됩니다. 이 전역 변수는 IdSSLOpenSSLHeaders 유닛에 있습니다.

예: openssl 1.1.1 라이브러리를 사용하려면

```
OPENSSL_API_VERSION := opSSL_1_1;
```

단일 sgclndy 패키지를 사용하는 방법

eSeGeCe ALL-ACCESS와 sgclndy 패키지를 동일한 애플리케이션에서 사용할 때, sgWebSockets Enterprise 버전이 Indy 유닛 이름이 변경된 사용자 정의 Indy 버전을 사용하기 때문에 소스가 중복될 수 있습니다. 예를 들어 sgclndGlobal.pas와 IdGlobal.pas와 같은 유닛이 동일한 애플리케이션에서 컴파일됨을 의미합니다(첫 번째는 sgWebSockets Enterprise 패키지의 구성 요소를 사용할 때 사용되고, 두 번째는 ftp, smtp 등 sgclndy 패키지의 구성 요소를 사용할 때 사용됩니다).

이 동작을 피하기 위해, sgWebSockets 패키지는 설치된 sgclndy 버전을 사용하면서도 모든 구성 요소를 활용하도록 구성할 수 있습니다. 이를 위해 아래 지침을 따르십시오:

1. sgWebSockets 패키지의 Source 폴더에 있는 sgVer.inc 파일을 여십시오.
2. 컴파일러 지시문 비활성화: SGC_CUSTOM_INDY. 이 옵션은 sgclnd*.pas로 시작하는 파일이 존재하며 sgWebSockets Enterprise Package를 컴파일할 때 사용되어야 함을 컴파일러에 알립니다.
3. 다음 컴파일러를 활성화하십시오: SGC_INDY_LIB. 이 옵션은 sgclndy 패키지가 설치되어 있으며 패키지를 컴파일할 때 사용해야 함을 컴파일러에 알립니다.

```
{ $DEFINE SGC_INDY_LIB }
```

이전 구성을 사용하면 sgWebSockets Enterprise 패키지가 설치된 sgclndy 패키지를 사용하며, 이 패키지를 사용하는 모든 기능(예: http/2, IOCP, openssl 3.0...)이 활성화됩니다.

WebSocket 이벤트

WebSocket 연결에는 다음 이벤트가 있습니다:

OnConnect

새 연결이 설정될 때 발생하는 이벤트입니다.

OnDisconnect

연결이 닫힐 때 발생하는 이벤트입니다.

OnError

연결에 오류가 발생할 때 발생하는 이벤트입니다.

OnMessage

새 텍스트 메시지가 수신될 때 발생하는 이벤트입니다.

OnBinary

새 바이너리 메시지가 수신될 때 발생하는 이벤트입니다.

기본적으로 `sgcWebSockets`는 이러한 이벤트를 발생시키기 위해 **비동기** 메커니즘을 사용합니다. 이러한 이벤트 중 하나가 내부적으로 발생하면, 이 메시지를 큐에 넣고 허용될 때 운영 체제에 의해 디스패치됩니다. 이 동작은 **NotifyEvents**라는 속성을 사용하여 수정할 수 있으며, 기본적으로 **neAsynchronous**가 선택됩니다. **neNoSync**가 선택되면 메인 스레드와 동기화하지 않고 이벤트가 발생합니다 (VCL 컨트롤을 업데이트하거나 공유 리소스에 액세스해야 하는 경우, 자체 동기화 방법을 구현해야 합니다).

neNoSync는 다음 경우에 권장됩니다:

1. 매우 짧은 시간 내에 많은 메시지를 처리해야 합니다.
2. 프로젝트가 명령줄용으로 빌드됩니다 (**neNoSync**를 설정하지 않으면 이벤트를 받지 못합니다).
3. 프로젝트가 라이브러리입니다.

그렇지 않은 경우 **neAsynchronous**의 기본 속성 값을 사용할 수 있습니다.

WebSocket Parameters Connection

지원 대상

TsgcWebSocketClient

Java script

새 WebSocket 연결이 설정될 때 클라이언트에서 서버로 매개변수를 전달하는 것이 유용할 때가 있습니다. 서버에 일부 매개변수를 전달해야 하는 경우 다음 속성을 사용할 수 있습니다:

옵션 / 매개변수

기본적으로 '/'로 설정되어 있으며, id=1과 같은 매개변수를 전달해야 하는 경우, 이 속성을 '/?id=1'로 설정할 수 있습니다

Server Side에서 다음 매개변수를 사용하여 클라이언트 매개변수를 처리할 수 있습니다:

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

Javascript를 사용하여, 연결 url을 통해 매개변수를 전달할 수 있습니다, 예:

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

DLL 내부에서 사용

sgcWebSockets와 함께 동적 링크 라이브러리(DLL)(또는 콘솔 애플리케이션)로 작업해야 하는 경우 **NotifyEvents** 속성을 **neNoSync**로 설정해야 합니다.

Windows Service

windows 서비스에서 sgcWebSockets 라이브러리를 실행할 때 **NotifyEvents** 속성은 **neNoSync**로 설정되어야 합니다.

```
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

TsgcWebSocketServer는 즐겨 사용하는 웹 브라우저로 WebSocket Server 연결을 테스트할 수 있는 내장 웹 페이지를 구현합니다.

이 테스트 페이지에 접근하려면 다음 URL을 입력해야 합니다:

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

예: WebSocket 서버를 IP 127.0.0.1에 구성하고 포트 80을 사용하는 경우, 다음을 입력해야 합니다:

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

이 페이지에서 다음 WebSocket 메서드를 테스트할 수 있습니다:

Open
Close
Status
Send

WebBrowser HTML 테스트 페이지를 비활성화하려면 TsgcWebSocketServer.Options.HTMLFiles = false;로 설정하면 됩니다.

사용자 지정 서브프로토콜

클라이언트는 핸드셰이크에 서브프로토콜 이름을 포함하여 서버가 특정 서브프로토콜을 사용하도록 요청할 수 있습니다. 지정되면 서버는 연결이 설정되기 위해 응답에 선택된 서브프로토콜 값 중 하나를 포함해야 합니다.

자신만의 서브프로토콜을 생성하려면 사용자 정의 서브프로토콜을 만들기 위해 `TsgcWSProtocol_Client_Base` 및 `TsgcWSProtocol_Server_Base`에서 상속해야 합니다.



Authentication

지원 대상

TsgcWebSocketServer
TsgcWebSocketHTTPServer
TsgcWebSocketClient

Java script (*URL Authentication만 지원됨)

WebSocket 사양은 어떠한 인증 방법도 정의하지 않으며, 웹 브라우저 구현은 새 WebSocket 연결에서 사용자 정의 헤더를 보내는 것을 허용하지 않습니다.

이 기능을 활성화하려면 다음 속성에 접근해야 합니다.

Authentication/ Enabled

sgcWebSockets는 3가지 유형의 WebSocket 인증을 구현합니다:

Session: 클라이언트는 사용자 이름과 비밀번호를 전달하여 HTTP GET을 수행해야 하며, 인증되면 서버는 Session ID로 응답합니다. 이 Session ID로 클라이언트는 이를 매개변수로 전달하여 WebSocket 연결을 엽니다. 일반 HTTP 요청을 사용하여 사용자와 비밀번호를 매개변수로 전달하여 session id를 가져올 수 있습니다

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

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

이것은 WebSocket 연결을 사용하여 서버에 연결하는 데 사용되는 토큰을 반환합니다:

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

URL: 클라이언트가 사용자 이름과 비밀번호를 매개변수로 전달하여 WebSocket 연결을 엽니다.

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

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

Basic: Basic Access Authentication을 구현하며, VCL Websockets(서버 및 클라이언트)와 HTTP 요청에만 적용됩니다(클라이언트 웹 브라우저는 이러한 유형의 인증을 구현하지 않습니다). 클라이언트가 연결을 시도하면 AUTH BASIC 사양을 사용하여 헤더를 보냅니다.

Authentication/ AuthUsers 속성을 사용하여 인증된 사용자 목록을 정의할 수 있습니다. 이 스키마에 따라 모든 항목을 정의해야 합니다: user=password. 예제:

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

OnAuthentication이라는 이벤트가 있으며, 사용자가 AuthUsers 목록에 없거나 클라이언트가 권한 부여 요청을 보내지 않은 경우 여기서 인증을 처리할 수 있습니다... User와 Password 매개변수를 확인하고 올바르게 Authenticated 변수를 True로 설정하십시오. 예:

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

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


Secure Connections

지원 대상

TsgcWebSocketServer
TsgcWebSocketHTTPServer
TsgcWebSocketClient
웹 브라우저

SSL 지원은 Indy 구현을 기반으로 하므로, 이 기능을 사용하려면 openssl 라이브러리를 배포해야 합니다. TsgcWebSocketClient는 Microsoft SChannel을 지원하므로 WebSocket 클라이언트에서 SChannel 옵션이 선택된 경우 Windows 32 및 64비트용 openssl 라이브러리를 배포할 필요가 없습니다.

Server Side

이 기능을 활성화하려면 다음 속성을 활성화해야 합니다:

SSL/ Enable

정의해야 하는 다른 속성이 있습니다:

SSLOptions/ CertFile/ KeyFile/ RootCertFile: websocket 통신을 암호화하려면 .PEM 형식의 인증서가 필요합니다.

SSLOptions/ Password: 이것은 선택 사항이며 인증서에 비밀번호가 있는 경우에만 필요합니다.

SSLOptions/ Port: SSL 연결에 사용되는 포트.

클라이언트 측

이 기능을 활성화하려면 다음 속성을 활성화해야 합니다:

TLS/ Enable

OpenSSL

기본적으로, Indy 기반 클라이언트 및 서버 구성 요소는 보안 websocket 서버에 연결할 때 openssl 라이브러리를 사용합니다.

Indy는 1.0.2 OpenSSL API만 지원하므로 API 1.1은 지원되지 않습니다. sgcWebSockets를 당사의 사용자 지정 Indy 라이브러리로 컴파일하면 API 1.1을 사용하고 TLS 1.3 버전을 선택할 수 있습니다. OpenSSL_Options 속성에서 사용하려는 OpenSSL API를 선택하기만 하면 됩니다:

- **osIAPI_1_0:** 이것은 기본 Indy API로, openssl 1.0.2 라이브러리와 함께 표준 Indy 패키지를 사용할 수 있습니다.
- **osIAPI_1_1:** 당사의 사용자 지정 Indy 라이브러리(Enterprise Edition)로 sgcWebSockets를 컴파일하는 경우에만 선택하십시오. openssl 1.1.1 라이브러리를 사용합니다.
- **osIAPI_3_0:** sgcWebSockets를 당사의 사용자 지정 Indy 라이브러리(Enterprise Edition)로 컴파일하는 경우에만 선택하십시오. openssl 3.0.0 라이브러리를 사용합니다.
 - **ECDHE:** TLS 1.2에 대해 ECDHE를 활성화할 수 있습니다(더 안전한 연결).

Events

SSL 설정을 사용자 정의하는 데 사용할 수 있는 2개의 이벤트가 있습니다:

OnSSLGetHandler

이 이벤트는 SSL 핸들러가 생성되기 전에 발생합니다. 여기에서 자체 SSL 핸들러(TidServerIOHandlerSSLBase 또는 TidIOHandlerSSLBase에서 상속되어야 함)를 생성하고 필요한 속성을 설정할 수 있습니다

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

OnSSLAfterCreateHandler

사용자 지정 SSL 객체가 생성되지 않은 경우 OpenSSL 핸들러를 사용하여 기본적으로 하나가 생성됩니다. 필요한 경우 SSL 핸들러 속성에 액세스하여 수정할 수 있습니다.

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

Microsoft SChannel

sgcWebSockets 4.2.6부터 openssl 대신 SChannel을 사용할 수 있습니다(Windows 7 이상의 Windows 전용). 이는 openssl 라이브러리를 배포할 필요가 없음을 의미합니다. TLS 1.0은 Windows 7부터 지원되지만, Windows 7에서 TLS 1.2와 같은 더 현대적인 구현이 필요한 경우 Windows 레지스트리에서 TLS 1.1 및 TLS 1.2를 활성화해야 합니다. Delphi 2010 Professional Edition(또는 Delphi 7, 2007 및 2009의 경우 Enterprise Edition)이 필요합니다.

HeartBeat

지원 대상

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

서버 구성 요소에서 x초마다 모든 활성 WebSocket 연결에 자동으로 ping을 보냅니다.

Client 구성 요소에서 x초마다 서버에 ping을 자동으로 보냅니다.

HeartBeat에는 다음 속성이 있습니다:

- **Enabled:** true이면 ping을 보냅니다.
- **Interval:** ping이 전송되는 초 단위 값입니다. 예: 값이 10이면 ping이 10초마다 전송됩니다.
- **Timeout:** 서버로부터 응답을 기다리는 시간입니다. 예: 값이 30이면 연결을 닫기 전에 응답을 받기 위해 30초 동안 대기합니다.
- **HeartBeatType:** HeartBeat가 작동하는 방식을 사용자 정의할 수 있습니다
 - **hbtAlways:** Interval에 정의된 x초마다 ping을 보냅니다.
 - **hbtOnlyIfNoMsgRcvInterval:** Interval 속성에 정의된 최근 x초 동안 메시지가 수신되지 않은 경우에만 x초마다 ping을 보냅니다.

Customize HeartBeat

클라이언트 및 서버 구성 요소는 사용자 지정 ping을 보내고 연결이 여전히 활성 상태인지 확인하기 위해 HeartBeat의 사용자 지정을 허용합니다. **OnBeforeHeartBeat** 이벤트는 정확히 그 용도로 만들어졌습니다. 이를 통해 사용자 지정 메시지를 보내거나 표준 ping을 보내지 않을 수 있습니다.

예제: 30초마다 ping으로 텍스트 메시지를 보냅니다.

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

WatchDog

지원 대상

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

Server

서버 구성 요소에서 예기치 않은 종료 후 서버를 자동으로 다시 시작합니다. 60초마다 서버가 활성 상태인지 확인하려면 다음 속성을 설정하기만 하면 됩니다:

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

WatchDog.Monitor를 사용하면 새 클라이언트가 서버에 연결할 수 있는지 확인할 수 있습니다. 이는 서버에 WebSocket 연결을 열려고 시도하는 내부 클라이언트에 의해 수행되며, 실패하면 서버를 재시작합니다. 시간 초과 10초로 클라이언트가 서버에 연결할 수 있는지 모니터링하려면 다음 속성을 설정하십시오.

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

클라이언트

클라이언트 구성 요소에서, 예기치 않은 연결 해제 후 서버에 자동으로 재연결합니다. 연결 해제 후 10초마다 재연결하려면 다음 속성을 설정하기만 하면 됩니다.

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

Logs

지원 대상

TsgcWebSocketServer
TsgcWebSocketHTTPServer
TsgcWebSocketClient

이것은 WebSocket 연결을 디버깅할 수 있게 해주는 유용한 기능입니다. 이를 활성화하려면 다음 속성에 접근해야 합니다:

LogFile/ Enabled

일단 활성화되면, 새 연결이 설정될 때마다 텍스트 파일에 기록됩니다. 서버 구성 요소에서 파일이 존재하지 않으면 생성되지만, 서버가 닫힐 때까지 액세스할 수 없습니다. 서버가 활성화된 동안 로그 파일을 열려면, 서버를 시작하기 전에 로그 파일을 생성해야 합니다.

예제:

```
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

WebSocket 프레임은 마스킹될 수 있으며, 이는 기록된 메시지를 읽을 수 없음을 의미합니다.

LogFile.UnMaskFrames 속성이 = True인 경우(기본적으로 true임)

- **WebSocket Client**가 보낸 메시지는 **마스킹되지 않은** 상태로 저장됩니다.
- **WebSocket 서버**가 수신한 메시지는 **마스킹된** 상태와 **마스킹되지 않은** 상태로 저장됩니다(이유는 소켓이 버퍼를 읽을 때 아직 메시지의 프로토콜을 알지 못하므로 둘 다 저장하기 때문입니다).

HTTP

지원하는 항목

TsgcWebSocketHTTPServer

TsgcWebSocketHTTPServer는 동일한 포트를 사용하여 WebSocket 및 HTTP 연결을 처리할 수 있는 구성 요소입니다. HTTP 포트(일반적으로 포트 80)만 활성화된 서버를 설정해야 할 때 매우 유용합니다. 이 구성 요소는 모든 **TsgcWebSocketServer** 기능을 지원하며 HTML 페이지를 제공할 수 있습니다.

DocumentRoot 속성을 사용하여 **HTML 페이지를 정적으로 제공할 수** 있습니다. 예: test.html을 "C:\inetpub\wwwroot" 디렉터리에 저장하고 **DocumentRoot**를 "C:\inetpub\wwwroot"로 설정한 경우, 클라이언트가 test.html에 액세스하려고 하면 자동으로 제공됩니다. 예:

http://localhost/test.html

또는 코드로 **HTML 또는 기타 리소스를 동적으로 제공할 수** 있습니다. 이를 위해 클라이언트가 새 HTML 페이지, 이미지, javascript 파일... 을 요청할 때마다 발생하는 **OnCommandGet**이라는 이벤트가 있습니다. 기본적으로 클라이언트가 어떤 문서를 요청하는지 확인하고(ARequestInfo.Document 사용) 클라이언트에 응답을 보내야 합니다(응답 콘텐츠를 보내는 AResponseInfo.ContentText, 응답 유형인 AResponse.ContentType 및 응답 코드 번호가 있는 AResponseInfo.ResponseNo 사용, 일반적으로 200). 예:

```
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;
    }
}
```

브로드캐스트와 채널

지원 컴포넌트

[TsgcWebSocketServer](#)

[TsgcWebSocketHTTPServer](#)

[TsgcWebSocketServer_HTTPAPI](#)

Broadcast 메서드는 기본적으로 연결된 모든 클라이언트에게 메시지를 전송하지만, **channels** 인수를 사용하여 필터링하고 채널을 구독한 클라이언트에게만 메시지를 브로드캐스트할 수 있습니다.

예시: 서버에 desktop과 모바일 장치라는 2가지 유형의 연결된 클라이언트가 있다면, "desktop"과 "mobile"이라는 2개의 채널을 생성할 수 있습니다.

서버의 OnConnect 이벤트에서 클라이언트가 모바일인지 식별할 수 있다면, 다음과 같이 처리할 수 있습니다:

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

먼저 Connection을 TsgcWSConnectionServer로 캐스팅하여 구독 메서드에 접근하고, 필터에 부합하면 desktop 채널을 구독하게 됩니다. 채널 구독은 어떤 이벤트에서도 수행할 수 있습니다. 예를 들어, 클라이언트에게 모바일 여부를 알려달라고 요청하고 클라이언트에서 서버로 클라이언트 정보가 담긴 메시지를 전송하게 할 수 있습니다. 그러면 desktop 연결에만 브로드캐스트할 수 있습니다:

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

연결이 100개이고 그중 30개가 모바일이라면, 메시지는 나머지 70개에만 전송됩니다.

Bindings

지원하는 항목

[TsgcWebSocketServer](#)

[TsgcWebSocketHTTPServer](#)

일반적으로 서버에는 둘 이상의 IP가 있습니다. WebSocket 서버를 활성화하고 수신 포트를 80으로 설정하면, 서버가 시작될 때 모든 IP의 포트 80에서 수신 대기를 시도합니다. 따라서 IP가 3개 있는 경우 각각의 포트 80을 바인딩합니다.

바인딩을 사용하면 서버가 사용하는 정확한 IP와 Port를 정의할 수 있습니다. 예를 들어 IP 127.0.0.1(내부 주소)과 80.254.21.11(공개 주소)에 대해 포트 80에서 수신해야 하는 경우, 서버가 활성화되기 전에 다음과 같이 할 수 있습니다:

```
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";
}
```


대용량 파일 전송(POST)

지원 컴포넌트

TsgcWebSocketHTTPServer

TsgcWebSocketServer_HTTPAPI

HTTP 클라이언트가 **multipart/form-data** 스트림을 전송하면, 해당 스트림은 서버의 메모리에 저장됩니다. 파일이 큰 경우 서버에서 **out of memory** 예외가 발생할 수 있습니다. 이러한 예외를 방지하기 위해 서버에는 HTTPUploadFiles라는 속성이 있으며, 여기서 POST 스트림을 메모리에 저장할지 파일 스트림으로 처리할지를 구성할 수 있습니다. 스트림을 파일 스트림으로 처리하면, 수신된 스트림이 하드 디스크에 직접 저장되므로 메모리 문제를 피할 수 있습니다.

서버가 multipart/form-data 스트림을 파일 스트림으로 저장하도록 구성하려면 다음 단계를 따르십시오.

1. **HTTPUploadFiles.StreamType = pstFileStream** 속성을 설정하십시오. 이 설정을 사용하면 서버가 이러한 스트림을 하드 디스크에 저장합니다.
2. 파일이 파일 스트림으로 저장될 **최소 바이트 크기**를 구성할 수 있습니다. 기본값은 0이며, 이는 모든 스트림이 파일 스트림으로 저장됨을 의미합니다.
3. 스트림이 저장되는 폴더는 **SaveDirectory**를 사용하여 지정하며, 설정하지 않으면 애플리케이션이 있는 동일한 폴더에 저장됩니다.
4. 클라이언트가 multipart/form-data를 전송하면 콘텐츠는 경계(boundary) 안에 인코딩됩니다. **RemoveBoundaries** 속성이 활성화된 경우, 전체 스트림이 수신된 후 경계의 콘텐츠가 자동으로 추출됩니다.

예제 코드

먼저 새 서버 인스턴스를 생성하고 스트림이 파일 스트림으로 저장되도록 설정하십시오.

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

그런 다음 아래 구성으로 새 html 파일을 생성하십시오

```
<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>
```

마지막으로 웹 브라우저로 html 파일을 열고 서버로 파일을 전송하십시오. 서버는 확장자가 ".sgc_ps"인 새 파일 스트림을 생성하며, 스트림이 완전히 수신되면 경계에서 파일을 추출합니다.

이벤트

업로드 파일 흐름을 사용자 지정하는 데 사용할 수 있는 이벤트가 2개 있습니다(HTTPUploadFiles.RemoveBoundaries 속성이 활성화되어 있어야 합니다)

OnHTTPUploadBeforeSaveFile

이 이벤트는 파일이 저장되기 전에 발생하며 수신된 파일의 이름을 사용자 지정할 수 있습니다.

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

OnHTTPUploadAfterSaveFile

이 이벤트는 파일이 저장된 후에 발생하며 저장된 파일의 이름을 알 수 있게 해줍니다.

```
void OnHTTPUploadBeforeSaveFileEvent(TObject *Sender, String aFileName, String aFilePath)
{
}
```

```
    DoLog("File Received: " + aFileName);  
}
```

OnHTTPUploadReadInput

이 이벤트는 디코더가 파일 입력과 다른 입력 값을 읽을 때 발생합니다(예: 폼에 name, date 등의 변수가 있는 경우).

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

압축

지원하는 항목

[TsgcWebSocketServer](#)
[TsgcWebSocketHTTPServer](#)
[TsgcWebSocketClient](#)
Chrome과 같은 웹 브라우저

이것은 일반적으로 바이너리 메시지를 사용하여 많은 데이터를 보내야 할 때 매우 잘 작동하는 기능입니다. Chrome과 같은 일부 브라우저에서 지원하는 "PerMessage_Deflate" 프로토콜을 사용하여 WebSocket 메시지를 압축하기 때문입니다.

이 기능을 활성화하려면, 다음 속성을 활성화해야 합니다:

Extensions/ PerMessage_Deflate / Enabled

클라이언트가 WebSocket 서버에 연결하려고 하고 이 속성이 활성화되면, 이 속성이 활성화된 헤더를 보냅니다. 서버가 이 기능을 활성화한 경우, 이 프로토콜이 활성화된 응답을 클라이언트에 보내고 모든 메시지가 압축됩니다. 서버에 이 기능이 없으면 모든 메시지가 압축 없이 전송됩니다.

웹 브라우저에서는 아무것도 할 필요가 없습니다. 이 확장이 지원되면 자동으로 사용되고, 그렇지 않으면 메시지가 압축 없이 전송됩니다.

WebSocket 메시지가 작은 경우, 메시지를 압축/압축 해제하는 데 CPU 사이클을 소모하므로 이 속성을 활성화하지 않는 것이 좋습니다. 하지만 많은 양의 데이터를 전송하는 경우 메시지 교환 속도가 증가하는 것을 느낄 수 있습니다.

Flash

지원 대상

TsgcWebSocketServer
TsgcWebSocketHTTPServer

WebSocket은 다양한 웹 브라우저에서 기본적으로 지원됩니다(<http://caniuse.com/websockets>를 확인하십시오), 그러나 WebSocket을 구현하지 않는 일부 구버전이 있습니다(예: Internet Explorer 6, 7, 8 또는 9). WebSocket을 구현하지 않는 이러한 모든 브라우저에 대해 **Flash Fallback**을 활성화할 수 있습니다.

거의 모든 다른 또는 오래된 브라우저는 Adobe Flash Player를 설치하여 Flash를 지원합니다. Flash 연결을 지원하려면 Flash가 보안상의 이유로 cross-domain-access를 확인하기 위해 이 포트를 사용하므로 서버에서 **포트 843을 열어야** 합니다. 포트 843에 연결할 수 없으면 3초 동안 기다린 후 서버 기본 포트에 연결을 시도합니다.

Flash는 브라우저가 WebSockets를 네이티브로 지원하지 않는 경우에만 적용됩니다. 따라서 서버 측에서 Flash Fallback을 활성화하고 웹 브라우저가 WebSockets를 네이티브로 지원하면, 여전히 WebSockets를 전송 수단으로 사용합니다.

Flash Fallback을 활성화하려면 서버의 **FallBack / Flash** 속성에 접근하여 **활성화**해야 합니다. 두 가지 속성이 더 있습니다:

1. Domain: flash 연결을 단일/다중 도메인으로 제한해야 하는 경우(기본적으로 모든 도메인이 허용됨). 예: 이것은 도메인 swf.example.com에 대한 접근을 허용합니다

swf.example.com

2. Ports: flash 연결을 단일/다중 포트로 제한해야 하는 경우(기본적으로 모든 포트가 허용됨). 예: 이는 포트 123, 456, 457 및 458에 대한 액세스를 허용합니다

123,456-458

Flash 연결은 텍스트 메시지만 지원하며, 바이너리 메시지는 지원되지 않습니다.

Custom Objects

지원하는 항목

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

새 WebSocket 연결이 설정될 때마다, sgcWebSockets는 식별자, 수신/전송 바이트, 클라이언트 IP 등 일부 속성에 접근할 수 있는 [TsgcWSConnection](#) 클래스를 생성하며, 데이터베이스 접근, 세션 객체 등 객체를 메모리에 저장할 수 있는 **Data**라는 속성이 있습니다.

```
// 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

지원 대상

TsgcWebSocketServer
TsgcWebSocketHTTPServer
TsgcWebSocketServer_HTTPAPI

sgcWebSockets 는 연결된 클라이언트의 지정된 하위 집합에 메시지를 브로드캐스트하는 강력한 메서드를 제공합니다. 그룹은 임의의 수의 클라이언트를 가질 수 있으며, 클라이언트는 임의의 수의 그룹의 멤버가 될 수 있습니다. 그룹을 명시적으로 생성할 필요가 없습니다. 실제로 Groups.Add 호출에서 그룹 이름을 처음 지정할 때 그룹이 자동으로 생성됩니다.

Groups.Add 메서드를 사용하여 그룹에 사용자를 추가하면, 사용자는 현재 연결 기간 동안 해당 그룹으로 보내는 메시지를 받습니다.

사용자 추가 및 제거

그룹에서 사용자를 추가하거나 제거하려면 Add 또는 Remove 메서드를 호출하고 Group Name과 TsgcWSConnection 클래스를 전달하십시오. 연결이 종료될 때 그룹에서 사용자를 수동으로 제거할 필요가 없습니다.

다음 예제는 Groups.Add 메서드를 보여줍니다.

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

그룹에 메시지 보내기

다음 예제와 같이 그룹의 모든 멤버에게 메시지를 보낼 수 있습니다.

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

또는 "Room"으로 시작하는 모든 그룹에 메시지를 보낼 수 있습니다(따라서 Room1, Room2, Room3...이 존재하면 이러한 사용자가 메시지를 받습니다).

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

Events

새 클라이언트가 그룹에 추가되거나 하나가 제거될 때마다 Group과 Client를 처리하는 데 사용할 수 있는 2개의 이벤트가 있습니다:

OnClientAdded
OnClientRemoved

예: 멤버가 그룹을 떠날 때 그룹에 메시지를 보냅니다.

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

IOCP

Supported by

TsgcWebSocketServer
TsgcWebSocketHTTPServer

*사용자 지정 Indy 버전이 필요합니다.

Windows용 IOCP는 Indy가 기본적으로 하는 것처럼 연결당 하나의 스레드를 사용하는 대신 제한된 스레드 풀을 사용하여 수천 개의 연결을 처리할 수 있는 API입니다.

Indy 서버에 대해 IOCP를 활성화하려면 **IOHandlerOptions** 속성으로 이동하여 IOHandler 유형으로 **iohIOCP**를 선택하십시오.

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

IOCPThreads는 IOCP 비동기 요청(오버랩 작업)에 사용되는 스레드입니다. 기본적으로 값은 0이며, 이는 스레드 수가 프로세서 수를 사용하여 계산됨을 의미합니다(cpucount 함수가 지원되지 않아 스레드 수가 32로 설정되는 Delphi 7 및 2007 제외).

WorkOpThreads는 연결이 항상 동일한 스레드에서 처리되기를 원하는 경우에만 활성화해야 합니다. IOCP를 사용할 때, 요청은 스레드 풀에 의해 처리되며, (동일한 연결에 대한) 모든 요청이 다른 스레드에서 처리될 수 있습니다. 모든 연결을 동일한 스레드에서 처리하려면 WorkOpThreads에 이러한 요청을 처리하는 데 사용되는 스레드 수를 설정하십시오. 이는 서버 성능에 영향을 미치며, 이 기능이 필요한 경우에만 0보다 큰 값을 설정하는 것이 권장됩니다.

수천 개의 연결을 처리해야 할 때 windows 서버에 대해 IOCP를 활성화하는 것이 권장됩니다. 서버가 최대 100개의 동시 연결만 처리하는 경우 기본 Indy 스레드 모델을 유지할 수 있습니다.

OnDisconnect 이벤트가 발생하지 않음

IOCP는 기본 indy IOHandler와 다르게 작동합니다. 기본 Indy IOHandler를 사용하면 모든 연결이 스레드에서 실행되고 이러한 스레드는 연결이 활성 상태인지 확인하면서 항상 실행되므로, 연결 해제가 있으면 짧은 시간 내에 알림이 전달됩니다.

IOCP는 다르게 작동합니다. Indy가 기본적으로 하는 1 스레드 = 1 연결 대신, 모든 연결을 처리하는 스레드 풀이 있습니다. IOCP의 경우, 연결이 여전히 활성 상태인지 감지하는 유일한 방법은 소켓에 쓰기를 시도하는 것입니다. 오류가 있으면 연결이 닫혔음을 의미합니다. 연결 해제를 감지하는 데는 2가지 옵션이 있습니다:

1. **TsgcWebSocketClient**를 사용하는 경우 Options 속성에서 **CleanDisconnect := True**로 활성화할 수 있습니다(기본적으로 비활성화됨). 활성화되면 클라이언트가 연결을 끊기 전에 연결 해제에 대해 서버에 알리는 메시지를 보내므로, 서버가 이 메시지를 수신하고 OnDisconnect 이벤트가 발생합니다.
2. 서버 측에서 **heartbeat**를 활성화할 수 있습니다. 예를 들어 60초마다 연결된 모든 클라이언트에 ping을 보내려고 시도하며, 연결이 끊긴 클라이언트가 있으면 OnDisconnect가 호출됩니다.

EPOLL

Supported by

TsgcWebSocketServer
TsgcWebSocketHTTPServer

*sgcWebSockets Enterprise Edition이 필요합니다.

Linux용 EPOLL은 Indy가 기본적으로 하는 것처럼 연결당 하나의 스레드를 사용하는 대신 제한된 스레드 풀을 사용하여 수천 개의 연결을 처리할 수 있는 API입니다.

Indy 서버에 EPOLL을 활성화하려면 **IOHandlerOptions** 속성으로 이동하여 IOHandler Type으로 **iohIEPOLL**을 선택하십시오.

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

EPOLLThreads는 EPOLL 비동기 요청(오버랩 작업)에 사용되는 스레드입니다. 기본적으로 값은 0이며, 이는 스레드 수가 프로세서 수를 사용하여 계산됨을 의미합니다(cpucount 함수가 지원되지 않아 스레드 수가 32로 설정되는 Delphi 7 및 2007 제외). 스레드 수를 수동으로 조정할 수 있습니다.

WorkOpThreads는 연결이 항상 동일한 스레드에서 처리되기를 원하는 경우에만 활성화해야 합니다. EPOLL을 사용할 때 요청은 스레드 풀에 의해 처리되며, (동일한 연결에 대한) 모든 요청이 다른 스레드에서 처리될 수 있습니다. 모든 연결을 동일한 스레드에서 처리하려면 WorkOpThreads에 이러한 요청을 처리하는 데 사용되는 스레드 수를 설정하십시오. 이는 서버 성능에 영향을 미치며, 이 기능이 필요한 경우에만 0보다 큰 값을 설정하는 것이 권장됩니다.

수천 개의 연결을 처리해야 하는 경우 Linux 서버에 EPOLL을 활성화하는 것이 권장됩니다. 서버가 최대 100개의 동시 연결만 처리하는 경우, 기본 Indy 스레드 모델을 유지할 수 있습니다.

OnDisconnect 이벤트가 발생하지 않음

EPOLL은 기본 indy IOHandler와 다르게 작동합니다. 기본 indy IOHandler에서는 모든 연결이 스레드에서 실행되며 이러한 스레드는 항상 실행되어 연결이 활성 상태인지 확인하므로, 연결이 해제되면 짧은 시간 안에 알림을 받습니다.

EPOLL은 다르게 작동합니다. indy가 기본적으로 하는 것처럼 1개 스레드 = 1개 연결이 아니라, 모든 연결을 처리하는 스레드 풀이 있습니다. EPOLL의 경우 연결이 여전히 살아 있는지 감지하는 유일한 방법은 소켓에 쓰기를 시도하는 것이며, 오류가 있으면 연결이 닫혔음을 의미합니다. 연결 해제를 감지하는 두 가지 옵션이 있습니다:

1. **TsgcWebSocketClient**를 사용하는 경우 Options 속성에서 **CleanDisconnect := True**로 활성화할 수 있습니다(기본적으로 비활성화됨). 활성화되면 클라이언트가 연결을 끊기 전에 연결 해제에 대해 서버에 알리는 메시지를 보내므로, 서버가 이 메시지를 수신하고 OnDisconnect 이벤트가 발생합니다.
2. 서버 측에서 **heartbeat**를 활성화할 수 있습니다. 예를 들어 60초마다 연결된 모든 클라이언트에 ping을 보내려고 시도하며, 연결이 끊긴 클라이언트가 있으면 OnDisconnect가 호출됩니다.

Linux Connections Limit

동시 열린 연결 수를 늘리려면 다음 명령을 사용하십시오.

```
ulimit -n 10000
```

이전 명령은 열린 파일 디스크립터의 최대 수를 10000으로 설정합니다

ALPN

지원 대상

TsgcWebSocketServer
TsgcWebSocketHTTPServer
TsgcWebSocketClient

*사용자 지정 Indy 버전이 필요합니다.

Application-Layer Protocol Negotiation (ALPN)은 애플리케이션 계층 프로토콜 협상을 위한 Transport Layer Security (TLS) 확장입니다. ALPN을 사용하면 애플리케이션 계층이 추가 왕복을 피하는 방식으로 보안 연결을 통해 수행되어야 할 프로토콜을 협상할 수 있으며, 이는 애플리케이션 계층 프로토콜과 독립적입니다. 이는 보안 HTTP/2 연결에 필요하며, HTTP/1.x에 비해 웹 페이지의 압축을 개선하고 지연 시간을 줄입니다.

클라이언트

TLSOptions.ALPNProtocols에서 클라이언트가 지원하는 프로토콜을 구성할 수 있습니다. 클라이언트가 서버에 연결하면 이러한 프로토콜이 초기 TLS 핸드셰이크 'Client Hello'에서 전송되며, 클라이언트가 지원하는 프로토콜을 나열하고 서버가 어떤 프로토콜을 사용할지 선택합니다(있는 경우).

TsgcWSConnectionClient의 ALPNProtocol 속성에 액세스하여 서버가 어떤 프로토콜을 선택했는지 얻을 수 있습니다.

Server

새로운 TLS 연결이 있을 때 OnSSLALPNSelect 이벤트가 호출되며, 여기에서 클라이언트가 지원하는 프로토콜 목록에 액세스할 수 있고 서버는 그 중 어떤 것을 지원할지 선택할 수 있습니다.

어떤 프로토콜도 지원하지 않는 경우, aProtocol은 비워 둘 수 있습니다.

```
// 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;
        }
    }
}
```

HTTP 요청 전달

지원 대상

TsgcWebSocketHTTPServer
 TsgcWebSocketServer_HTTPAPI
 TsgcWSHTTPWebBrokerBridgeServer
 TsgcWSHTTP2WebBrokerBridgeServer
 TsgcWSServer_HTTPAPI_WebBrokerBridge

일부 HTTP 요청을 다른 서버로 전달하도록 서버를 구성할 수 있습니다. 이는 둘 이상의 서버가 있고 하나의 서버만 public 주소에서 수신 대기할 때 매우 유용합니다.

Example: 서버를 구성하여 /internal에 대한 모든 요청은 다른 서버로 전달하고, 다른 모든 요청은 sgcWebSockets 서버에서 처리하도록 할 수 있습니다.

OnBeforeForwardHTTP 이벤트를 사용하여 요청된 URL을 전달해야 하는지 확인하고, 전달해야 하는 경우 전달할 URL을 설정하십시오.

예제: 문서 "/internal"에 대한 모든 요청을 서버 "localhost:8080"으로 전달하려면 다음을 수행하십시오:

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

Other Options

HTTP 요청을 전달하려면 다음 추가 옵션이 있습니다:

1. 기본적으로 요청은 원본 문서를 사용하여 전달됩니다. **예제:** http://localhost:8080/internal 요청을 내부 서버 http://localhost:5555로 전달하면, 전달된 URL은 http://localhost:5555/internal이 됩니다. 그러나 Forward 객체의 **Document** 속성을 사용하여 Document를 수정할 수 있습니다(기본적으로 원본 요청과 동일하게 사용됨).

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

2. 보안 HTTP 연결(HTTPs)을 전달하는 경우, Forward 객체의 **TLSEOptions** 속성에서 SSL/TLS 옵션을 사용자 정의할 수 있습니다. **예제:** TLS 버전을 설정합니다

```
aForward.TLSEOptions.Version = tls1_2
```

3. 다음 속성을 사용하여 HTTP 요청을 사용자 정의할 수 있습니다:

- **QueryParams:** 문서 뒤의 매개변수입니다. 예: 'id=1&user=2'.
- **Host:** 요청이 전송되는 서버의 host 및 포트 번호를 지정합니다. 예: www.esegece.com:443
- **Origin:** 요청을 발생시킨 origin(스킴, 호스트 이름 및 포트)입니다. 예: https://www.esegece.com/document.
- **LogFilename:** 요청/응답이 저장될 파일 이름입니다.
- **NoCache:** 요청이 웹 브라우저 캐시를 사용해서는 안 되는 경우이며, 기본적으로 활성화되어 있습니다.
- **CustomHeaders:** 요청에 추가될 사용자 정의 헤더 목록. 예: CustomHeaders.Add('X-ReverseProxy-Host: http://127.0.0.1:8888/test');

Quality Of Service

지원하는 항목

TsgcWSPServer_sgc
TsgcWSPClient_sgc
TsgcWSPClient_MQTT
Java script

SGC Default Protocol 및 MQTT는 메시지 전달을 위한 QoS(Quality of Service)를 구현합니다. 3가지 유형이 있습니다:

Level 0: "At most once"는 기본 TCP/IP 네트워크의 최선의 노력에 따라 메시지가 전달되는 것을 의미합니다. 메시지 손실 또는 중복이 발생할 수 있습니다. 이 레벨은 예를 들어 개별 측정값이 손실되어도 다음 측정값이 곧 게시되므로 중요하지 않은 주변 센서 데이터에 사용될 수 있습니다.

Level 1: "최소 한 번"으로, 메시지가 도착하는 것은 보장되지만 중복이 발생할 수 있습니다.

Level 2: "Exactly once", 메시지가 정확히 한 번 도착하도록 보장됩니다. 이 수준은 예를 들어 중복되거나 손실된 메시지가 잘못된 청구로 이어질 수 있는 청구 시스템에서 사용될 수 있습니다.

Level 0

메시지는 기본 TCP/IP 네트워크의 최선의 노력에 따라 전달됩니다. 응답이 예상되지 않으며 프로토콜에 재시도 의미가 정의되어 있지 않습니다. 메시지는 서버에 한 번 도착하거나 전혀 도착하지 않습니다.

아래 표는 QoS 레벨 0 프로토콜 흐름을 보여줍니다.

Client	Message and direction	서버
QoS = 0	PUBLISH ----->	Action: 구독자에게 메시지 게시

Level 1

서버의 메시지 수신은 ACKNOWLEDGEMENT 메시지로 확인됩니다. 통신 링크나 송신 장치에 식별된 장애가 있거나 지정된 시간 후에도 확인 메시지가 수신되지 않으면 송신자가 메시지를 다시 보냅니다. 메시지는 서버에 적어도 한 번 도착합니다.

QoS level 1의 메시지에는 메시지에 Message ID가 있습니다.

아래 표는 QoS 레벨 1 프로토콜 흐름을 보여줍니다.

Client	Message and direction	서버
QoS = 1 Message ID = x Action: 메시지 저장	PUBLISH ----->	Actions: • 메시지 저장 • 구독자에게 메시지 게시 • 메시지 삭제
Action: 메시지 폐기	ACKNOWLEDGEMENT <-----	

클라이언트가 ACKNOWLEDGMENT 메시지를 수신하지 못하면(애플리케이션에서 정의한 시간 기간 내에, 또는 실패가 감지되고 통신 세션이 다시 시작되는 경우), 클라이언트는 PUBLISH 메시지를 다시 보낼 수 있습니다.

Level 2

QoS level 1 이상의 추가 프로토콜 흐름은 중복 메시지가 수신 애플리케이션에 전달되지 않도록 보장합니다. 이는 가장 높은 수준의 전달이며, 중복 메시지가 허용되지 않는 경우에 사용됩니다. 네트워크 트래픽이 증가하지만, 메시지 내용의 중요성 때문에 일반적으로 허용됩니다.

QoS 레벨 2의 메시지에는 메시지에 Message ID가 있습니다.

아래 표는 QoS level 2 프로토콜 흐름을 보여줍니다. PUBLISH 흐름이 수신자에 의해 어떻게 처리되어야 하는지에 대한 두 가지 의미가 있습니다.

Client	Message and direction	서버
QoS = 2 Message ID = x Action: 메시지 저장	PUBLISH ----->	Action: 메시지 저장
	PUBREC <-----	Message ID = x
Message ID = x	PUBREL ----->	Actions: • 구독자에게 메시지 게시 • 메시지 삭제
Action: 메시지 폐기	ACKNOWLEDGEMENT <-----	Message ID = x

실패가 감지되거나 정의된 시간 기간 후에 마지막으로 확인되지 않은 프로토콜 메시지에서 프로토콜 흐름이 재시도됩니다. 추가 프로토콜 흐름은 메시지가 구독자에게 정확히 한 번 전달되도록 보장합니다.

Queues

지원하는 항목

[TsgcWSPServer_sgc](#)

[TsgcWSPClient_sgc](#)

Java script

[SGC Default Protocol](#) 은 게시된 메시지에 연속성을 추가하기 위해 Queue를 구현합니다(**Published messages**에 대해서만 사용 가능)

Level 0: 메시지가 서버에 큐에 들어가지 않습니다

Level 1: 마지막 메시지만 서버에 큐에 들어가며, 클라이언트가 새 채널을 구독하거나 서버에 연결할 때마다 전송됩니다.

Level 2: 모든 메시지가 Server에 큐에 추가되며, 클라이언트가 새 채널을 구독하거나 서버에 연결할 때마다 전송됩니다.

Level 0

메시지가 서버에 의해 큐잉되지 않습니다

아래 표는 Queue 수준 0 프로토콜 흐름을 보여줍니다.

Client	Message and direction	서버
Queue = 0	PUBLISH ----->	Action: 구독자에게 메시지 게시

Level 1

Queue level 1의 메시지는 서버에 저장되며, 이 채널에 저장된 다른 메시지가 있으면 삭제됩니다.

아래 표는 Queue level 1 프로토콜 흐름을 보여줍니다.

Client	Message and direction	서버
Queue = 1	PUBLISH ----->	작업: - 이 채널의 모든 메시지를 삭제합니다. - 채널별로 마지막 메시지를 저장합니다.
Action: Process message	NOTIFY <-----	Action: 새 클라이언트가 이 채널을 구독할 때마다 마지막 메시지가 전송됩니다.

이는 게시자가 "report by exception" 방식으로 메시지를 보내는 경우에 유용하며, 메시지 사이에 시간이 걸릴 수 있습니다. 이를 통해 새 구독자는 retained 또는 Last Known Good 값으로 즉시 데이터를 받을 수 있습니다.

Level 2

Queue 레벨 2의 모든 메시지는 서버에 저장됩니다.

아래 표는 Queue level 2 프로토콜 흐름을 보여줍니다.

Client	Message and direction	서버
Queue = 2	PUBLISH ----->	Action: 메시지 저장
Action: Process message	NOTIFY <-----	Action: 새 클라이언트가 이 채널을 구독할 때마다, 모든 메시지가 전송됩니다.

Transactions

지원하는 항목

[TsgcWSPServer_sgc](#)

[TsgcWSPClient_sgc](#)

Java script

sgcWebSockets SGC Protocol은 트랜잭션 메시징을 지원하며, 클라이언트가 트랜잭션을 커밋하면 클라이언트가 보낸 모든 메시지가 서버 측에서 처리됩니다. 클라이언트가 호출하는 3개의 메서드가 있습니다:

StartTransaction

서버 측에서 새 Transaction을 생성하며, 이 메서드 이후 클라이언트에서 서버로 전송되는 모든 메시지는 클라이언트가 Commit 또는 Rollback을 호출할 때까지 서버 측에서 큐에 보관됩니다.

Client	Message and direction	서버
Channel = X	STARTTRANSACTION ----->	Action: 지정된 채널의 모든 메시지를 저장하기 위한 새 Queue를 생성합니다
Channel = X	PUBLISH ----->	Action: 메시지가 서버 측에 저장됩니다.
Action: 클라이언트가 메시지 전송 확인을 받습니다.	ACKNOWLEDGEMENT < -----	Action: 메시지가 저장되었기 때문에 서버가 클라이언트에 Acknowledgement를 반환합니다.
....		

Commit

클라이언트가 Commit을 호출하면 서버가 큐에 넣은 모든 메시지가 처리됩니다.

Client	Message and direction	서버
Channel = X	COMMIT ----->	Action: Transaction이 큐에 넣은 모든 메시지 처리

RollBack

클라이언트가 RollBack을 호출하면, 서버가 대기열에 넣은 모든 메시지가 삭제되고 서버 측에서 처리되지 않습니다.

Client	Message and direction	서버
Channel = X	ROLLBACK ----->	Action: Transaction으로 큐에 대기된 모든 메시지 삭제

TCP Connections

지원 대상

TsgcWebSocketServer
TsgcWebSocketHTTPServer
TsgcWebSocketClient

기본적으로 sgcWebSocket은 WebSocket을 프로토콜로 사용하지만, 클라이언트 및 서버 구성 요소에서 일반 TCP 프로토콜을 사용할 수 있습니다.

Client 구성 요소

WebSocket 프로토콜을 비활성화합니다.

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

Server Component

OnUnknownProtocol 이벤트를 처리하고 Transport를 trpTCP로 설정한 후 연결을 Accept하십시오.

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

그러면 클라이언트가 서버에 연결할 때, 이 연결은 TCP로 정의되며 WebSockets 대신 일반 TCP 프로토콜을 사용합니다. 일반 TCP 연결은 텍스트와 바이너리 메시지를 구분하지 않으므로, 수신되는 모든 메시지는 OnBinary 이벤트에 의해 처리됩니다.

End of Message

메시지가 큰 경우 때때로 조각화되어 수신될 수 있습니다. 어떤 바이트를 찾을지 지정하여 메시지의 끝을 감지하는 메서드가 있습니다. 예: STOMP 프로토콜에서 모든 메시지는 바이트 0과 10으로 끝납니다.

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

SubProtocol

지원 대상

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

WebSocket은 간단한 서브프로토콜 협상을 제공합니다. 기본적으로 요청이 지원하는 프로토콜 이름이 담긴 헤더를 추가합니다. 이러한 프로토콜이 수신되고, 수신자가 그중 하나를 지원하면 지원되는 서브프로토콜과 함께 응답을 보냅니다.

sgcWebSockets는 여러 SubProtocol을 지원합니다: [MQTT](#), [WAMP](#)... 등이 있습니다. 매우 쉬운 방법으로 자체 서브프로토콜을 구현할 수 있습니다. RegisterProtocol을 호출하고 SubProtocol Name을 인수로 보내기만 하면 됩니다.

예제: 서브프로토콜 "Test 1.0"을 구현하는 서버에 연결해야 합니다

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

단일 연결에서 둘 이상의 프로토콜을 사용하려면 **Broker Protocol**(서버 및 클라이언트) 구성 요소를 사용하여 처리할 수 있습니다. 클라이언트/서버와 프로토콜 사이에 Broker를 두기만 하면 됩니다. **예:** 단일 연결을 사용하여 SGC 및 Files 프로토콜을 사용합니다.

```
// ... 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;
```

Server/Client와 프로토콜 사이에 브로커 프로토콜이 연결되면, **OnConnect** 및 **OnDisconnect** 이벤트가 (Server 또는 Client 구성 요소 대신) Broker 구성 요소에서 발생합니다.

Throttle

지원하는 항목

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

대역폭 조절(Bandwidth Throttling)은 서버 및 클라이언트 구성 요소에서 지원되며, 활성화되면 소켓이 초당 송수신하는 비트 수를 제한할 수 있습니다. Indy는 블로킹 방법을 사용하므로, 클라이언트가 읽기를 제한하면 읽지 않은 데이터가 클라이언트 소켓 내부에 남아 있고 서버는 클라이언트에 새 데이터를 쓰는 것이 차단됩니다. 클라이언트가 데이터를 더 느리게 읽을수록 서버는 새 데이터를 더 느리게 쓸 수 있습니다.

Server-sent Events (Push Notifications)

지원 대상

TsGcWebSocketServer
TsGcWebSocketHTTPServer
Java script

SSE는 WebSockets의 일부가 아니며, 서버로부터 푸시 알림을 수신하기 위해 HTTP 연결을 여는 API를 정의합니다.

SSE는 기존 HTTP를 통해 전송됩니다. 즉, 작동하기 위해 특별한 프로토콜이나 서버 구현이 필요하지 않습니다. 또한 Server-Sent Events에는 자동 재연결, 이벤트 ID 및 임의의 이벤트를 보내는 기능 등 WebSocket이 설계상 부족한 다양한 기능이 있습니다.

Events

- **Open:** 새 SSE 연결이 열릴 때.
- **Message:** 클라이언트가 새 메시지를 받을 때.
- **Error:** 연결 끊김과 같은 연결 오류가 있을 때.

JavaScript API

이벤트 스트림을 구독하려면 EventSource 객체를 생성하고 스트림의 URL을 전달하십시오:

```
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);
```

서버에서 업데이트가 푸시되면 onmessage 핸들러가 발생하고 e.data 속성에서 새 데이터를 사용할 수 있습니다. 연결이 닫히면 브라우저는 약 3초 후에 소스에 자동으로 다시 연결됩니다(이것은 기본 재시도 간격이며, 서버 측에서 변경할 수 있습니다).

필드

사양에 의해 다음 필드 이름이 정의됩니다:

event

이벤트의 유형입니다. 이것이 지정되면, 지정된 이벤트 이름에 대한 리스너에게 브라우저에서 이벤트가 디스패치됩니다. 웹 사이트는 addEventListener()를 사용하여 명명된 이벤트를 수신 대기합니다. 메시지에 이벤트 이름이 지정되지 않은 경우 onmessage 처리기가 호출됩니다.

data

메시지의 data 필드입니다. EventSource가 data:로 시작하는 여러 연속된 줄을 수신하면, 각 줄 사이에 줄 바꿈 문자를 삽입하여 연결합니다. 후행 줄 바꿈은 제거됩니다.

id

EventSource 객체의 last event ID 값을 설정할 이벤트 ID입니다.

retry

이벤트를 보내려고 시도할 때 사용할 재연결 시간입니다. 이것은 재연결 시간을 밀리초 단위로 지정하는 정수여야 합니다. 정수가 아닌 값이 지정되면 필드가 무시됩니다.

다른 모든 필드 이름은 무시됩니다.

여러 줄 문자열의 경우 line feed로 #10을 사용하십시오.

사용 예제:

클라이언트에 메시지를 보내야 하는 경우, WriteData 메서드를 사용하기만 하면 됩니다.

```
// 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

지원하는 항목

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

로드 밸런싱을 사용하면 여러 백엔드 서버 간에 작업을 분배할 수 있습니다. 새 클라이언트가 연결을 요청할 때마다 (백엔드 서버에 연결된) 로드 밸런서 서버에 연결되고, 호스트, 포트 등에 대한 정보가 포함된 연결 문자열을 반환합니다. 이것은 클라이언트가 서버에 연결하는 데 사용됩니다. 예를 들어 4개의 서버가 있는 경우, 이 방법으로 모든 서버가 대략 동일한 수의 연결을 가지며 작업 부하가 유사해집니다.

클라이언트가 모든 서버의 모든 클라이언트에 메시지를 보내려면, Broadcast 메서드를 사용하기만 하면 메시지가 Load Balancer 서버에 연결된 모든 서버로 브로드캐스트됩니다.

이 기능을 활성화하려면:

1. [TsgcWebSocketLoadBalancerServer](#) 구성 요소를 드롭하고, 수신 포트를 설정하고, active를 True로 설정하십시오.
2. 서버 및 클라이언트 구성 요소에는 LoadBalancer라는 속성이 있으며, 여기에 Load Balancer Server의 host와 port를 설정하고 Enabled를 True로 설정해야 합니다.

이 구성 요소는 **WebSocket** 및 **HTTP** 프로토콜의 로드 밸런싱을 허용합니다.

Files

지원하는 항목

[TsgcWSPServer_sgc](#)

[TsgcWSPClient_sgc](#)

이 프로토콜은 클라이언트에서 서버로, 서버에서 클라이언트로 파일을 쉽게 보낼 수 있도록 합니다. 적은 메모리 사용량으로 아주 작은 파일부터 큰 파일까지 보낼 수 있습니다. 다음을 설정할 수 있습니다:

1. 패킷 크기(바이트)입니다.
2. 사용자 정의 채널을 사용하여 구독한 클라이언트에만 파일을 보냅니다.
3. 파일 송신 및 수신의 진행 상황.
4. 수신된 파일의 권한 부여.
5. 전송된 패킷의 확인 응답.

Proxy

지원하는 항목

[TsgcWebSocketClient](#)

클라이언트 WebSocket 구성 요소는 HTTP 프록시를 통한 WebSocket 연결을 지원합니다. 프록시 연결을 활성화하려면 다음 속성을 활성화해야 합니다:

Proxy / Enabled

True로 설정하면 다음을 설정할 수 있습니다:

Host: 프록시 서버 주소

Port: 프록시 서버 포트

UserName/Password: 프록시에 연결하기 위한 인증입니다. 필요한 경우에만 사용합니다.

ProxyType: 다음 프록시가 지원됩니다:

- HTTP
- Socks4
- Socks4A
- Socks5

SOCKS 속성에 액세스하고 Enabled를 True로 설정하여 SOCKS 프록시를 구성할 수 있습니다.

Fragmented Messages

지원 대상

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

기본적으로 sgcWebSockets 라이브러리를 사용하여 스트림을 보낼 때 모든 데이터를 단일 패킷으로 보내거나 모든 패킷을 버퍼링하고 마지막 패킷이 수신되면 OnBinary 메시지 이벤트가 호출됩니다.

이 동작은 **Options.FragmmentedMessages** 속성으로 사용자 정의할 수 있으며, 다음 값을 허용합니다:

1. frgOnlyBuffer: 이것이 기본값입니다. 패킷 메시지가 버퍼링되고 전체 스트림이 수신될 때만 OnBinary 메시지 이벤트가 호출됨을 의미합니다.
2. frgOnlyFragmented: 이는 수신된 모든 패킷에 대해 OnFragmented 이벤트만 호출됨을 의미합니다.
3. frgAll: 이는 수신된 모든 패킷과 전체 스트림이 수신될 때 OnFragmented 이벤트가 호출됨을 의미합니다.

OnFragmented 이벤트는 큰 스트림을 보내야 하고 수신자가 전송 진행 상황을 표시해야 할 때 유용합니다.

예: 클라이언트가 크기 1.000.000바이트의 스트림을 서버로 보내야 하고 서버는 수신된 1000바이트마다 진행 상황을 표시하려고 합니다.

클라이언트는 1000 크기의 패킷에 대해 writedata 메서드를 사용하여 스트림을 보냅니다

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

서버는 Options.FragmmentedMessages := frgAll을 설정하고 스트림의 진행 상황을 수신하기 위해 OnFragmented 이벤트를 처리합니다

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

TsgcWebSocketClient

클라이언트 WebSocket 구성 요소 — 모든 RFC 6455 WebSocket 서버에 연결하여 텍스트 및 바이너리 메시지를 송수신합니다.

소개

TsgcWebSocketClient는 클라이언트 WebSocket 구성 요소를 구현하며 WebSocket 서버에 연결할 수 있습니다. 이 구성 요소를 구성하려면 아래 단계를 따르십시오:

1. form에 **TsgcWebSocketClient** 구성 요소를 놓으십시오.
2. 사용 가능한 WebSocket 서버에 연결하려면 **Host**와 **Port**(기본값은 80)를 설정하십시오. **URL** 속성을 설정하면 Host, Port, Parameters... 가 URL로부터 업데이트됩니다. **예제**: wss://127.0.0.1:8080/ws/ 는 다음과 같은 결과가 됩니다:

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

3. require **TLS**(보안 연결) 여부를 선택할 수 있으며, 기본적으로 활성화되어 있지 않습니다.

4. HTTP Proxy Server를 통해 연결할 수 있습니다. 프록시 속성을 정의해야 합니다:

Host: 프록시 서버 호스트 이름입니다.

Port: 프록시 서버 포트 번호입니다.

Username: 인증을 위한 사용자 이름, 익명의 경우 비워 두십시오.

Password: 인증을 위한 비밀번호, 익명의 경우 비워 두십시오.

5. 서버가 **압축**을 지원하는 경우, 압축을 활성화하여 전송되는 메시지를 압축할 수 있습니다.

6. 허용되는 **Specifications**를 설정하십시오. 기본적으로 모든 사양이 활성화됩니다.

RFC6455: 표준이며 권장되는 WebSocket 사양입니다.

Hixie76: 항상 false입니다

7. 새 WebSocket 연결을 시작하려면 `Active = true`로 설정하십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 23개의 속성.
- 메서드 — 22개의 public 메서드.
- 이벤트 — 처리할 수 있는 16개의 이벤트.

Guides

Most common uses

- **Connection**
 - [How to Connect to a WebSocket Server](#)
 - [Open a Client Connection](#)
 - [Close a Client Connection](#)
 - [Keep Connection active](#)
 - [Dropped Disconnections](#)
 - [Connect TCP Server](#)
 - [WebSocket Redirections](#)
- **Secure Servers**
 - [Connect Secure Server](#)
 - [Certificates OpenSSL](#)
 - [Certificates SChannel](#)

- **SChannel Get Connection Info**
- 메시지 전송
 - 텍스트 메시지 전송
 - 바이너리 메시지 전송
- 메시지 수신
 - 텍스트 메시지 수신
 - 바이너리 메시지 수신
- 인증
 - **Client Authentication**
- Other
 - **Client Exceptions**
 - **Client WebSocket HandShake**
 - **Client Register Protocol**
 - **Client Proxies**

TsgcWebSocketClient | Connect WebSocket Server

URL 속성

WebSocket 서버에 연결하는 가장 쉬운 방법은 **URL** 속성을 사용하고 **Active = true**로 설정하는 것입니다.

예제: 보안 연결을 사용하여 www.esegece.com에 연결합니다.

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

호스트, 포트 및 매개변수

Host 및 port 속성을 사용하여 WebSocket 서버에 연결할 수 있습니다.

예제: 보안 연결을 사용하여 www.esegece.com에 연결

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

TsgcWebSocketClient | Client Open Connection

클라이언트가 서버에 연결하도록 구성되면 새 연결을 여는 3가지 다른 옵션이 있습니다.

Active 속성

새 연결을 여는 가장 쉬운 방법은 Active 속성을 true로 설정하는 것입니다. 이것은 구성 요소 구성을 사용하여 서버에 연결을 시도합니다.

Active 속성을 false로 설정하면 연결이 활성 상태인 경우 연결을 닫습니다.

이 메서드는 호출자와 동일한 스레드에서 실행됩니다. 따라서 Main Thread에서 호출하면 메서드는 애플리케이션의 Main Thread에서 실행됩니다.

Open Connection

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

Active = true를 호출할 때, 클라이언트가 아직 연결 중일 수 있으므로 여전히 서버로 데이터를 보낼 수 없습니다. 먼저 OnConnect 이벤트가 발생할 때까지 기다린 다음 서버로 메시지를 보내기 시작할 수 있습니다.

Close Connection

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

Active = false를 호출할 때, 이 코드 직후에 연결이 이미 닫혔는지 확인할 수 없으므로, OnDisconnect 이벤트가 발생할 때까지 기다려야 합니다.

Start/Stop 메서드

서버에 연결/연결 해제하기 위해 Start() 또는 Stop()을 호출하면, 호출은 보조 스레드에서 실행되므로 호출되는 스레드를 차단하지 않습니다. 서버에 연결하고 아래 코드를 계속 진행하려면 이 메서드를 사용하십시오.

Open Connection

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

Start()를 호출할 때 클라이언트가 아직 연결 중일 수 있으므로 여전히 서버에 데이터를 보낼 수 없습니다. 먼저 OnConnect 이벤트가 발생할 때까지 기다린 다음 서버에 메시지를 보내기 시작할 수 있습니다.

Close Connection

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

Stop()을 호출할 때, 이 코드 직후에 연결이 이미 닫혔는지 확인할 수 없으므로, OnDisconnect 이벤트가 발생할 때까지 기다려야 합니다.

Connect/Disconnect 메서드

Connect() 또는 Disconnect()를 호출하여 서버에 대한 연결을 열거나 닫을 때, 호출은 호출된 동일한 스레드에서 실행되지만 프로세스가 완료될 때까지 기다립니다. 프로세스가 완료될 때까지 기다리는 최대 시간을 정의하려면 Timeout을 설정해야 합니다(기본값 10초).

예제: 서버에 연결하고 최대 5초 대기

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

Connect() 메서드가 성공적인 결과를 반환하면 연결이 활성 상태이므로 이미 서버에 메시지를 전송할 수 있습니다.

예제: 서버에서 연결을 해제하고 최대 10초 대기

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

Disconnect() 메서드가 성공적인 결과를 반환하면, 이는 연결이 이미 닫혔음을 의미합니다.

OnBeforeConnect 이벤트는 클라이언트가 연결을 시도하기 전에 서버 연결 속성을 사용자 지정하는 데 사용할 수 있습니다.

TsgcWebSocketClient | 클라이언트 연결 종료

연결은 Active 속성, Stop 또는 Disconnect 메서드를 사용하여 닫을 수 있습니다. 자세한 내용은 [클라이언트 연결 열기](#)에서 확인하십시오.

CleanDisconnect

연결이 닫히면 close 메시지를 보내 다른 피어에게 연결이 닫히고 있음을 알릴 수 있습니다. 이 기능을 활성화하려면 Options.CleanDisconnect 속성을 true로 설정하십시오.

이 속성이 활성화되면, 연결이 닫히기 전에 클라이언트가 연결을 닫고 있음을 서버에 알리기 위해 Close 메시지가 전송됩니다.

Disconnect

[TsgcWSConnection](#)에는 소켓 수준에서 연결을 끊을 수 있는 Disconnect()라는 메서드가 있습니다. 이 메서드를 호출하면 서버로부터의 어떤 응답도 기다리지 않고 소켓이 직접 연결 해제됩니다. 이 메서드와 함께 Close Code를 보낼 수 있습니다.

Close

[TsgcWSConnection](#)에는 Close()라는 메서드가 있으며, 이를 통해 연결을 닫도록 요청하는 메시지를 서버에 보낼 수 있습니다. 서버가 이 메시지를 수신하면 연결을 닫아야 하며 클라이언트는 연결이 닫혔다는 알림을 수신합니다. 이 메서드와 함께 Close Code를 보낼 수 있습니다.

TsgcWebSocketClient | Client Keep Connection Open

클라이언트가 서버에 연결한 후, 때때로 신호 불량, 연결 오류 등으로 인해 연결이 닫힐 수 있습니다. 연결을 활성 상태로 유지하는 데 도움이 되는 2개의 속성이 있습니다.

HeartBeat

HeartBeat 속성을 사용하면 **연결을 활성 상태로 유지하기 위해 X 초마다 Ping을 보낼** 수 있습니다. 일부 서버는 피어 간에 데이터 교환이 없으면 TCP 연결을 닫습니다. HeartBeat는 특정 간격으로 ping을 보내 이 문제를 해결합니다. 일반적으로 이것으로 연결을 활성 상태로 유지하기에 충분하지만, X초 후에 서버로부터 응답을 받지 못할 때 연결을 닫으려면 TimeOut 간격을 설정할 수 있습니다.

예제: 30초마다 ping 보내기

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

OnBeforeHeartBeat이라는 이벤트가 있어 HeartBeat 동작을 사용자 정의할 수 있습니다. 기본적으로 HeartBeat이 활성화되어 있으면, 클라이언트는 HeartBeat.Interval 속성으로 설정된 대로 X초마다 WebSocket ping을 보냅니다.

OnBeforeHeartBeat에는 Handled라는 매개변수가 있으며, 기본적으로 false입니다. 이는 흐름이 TsgcWebSocketClient 구성 요소에 의해 제어됨을 의미합니다. 값을 True로 설정하면 ping이 전송되지 않으며, Connection 클래스를 사용하여 사용자 정의 메시지를 보낼 수 있습니다.

WatchDog

WatchDog가 활성화된 경우, 클라이언트가 연결 끊김을 감지하면 WatchDog는 연결이 다시 활성화될 때까지 X초마다 재연결을 시도합니다.

예: 연결 해제 후 무제한 시도로 10초마다 재연결합니다.

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

클라이언트가 연결을 시도할 서버를 변경하기 위해 **OnBeforeWatchDog** 이벤트를 사용할 수 있습니다. **예:** 3번 재시도 후 클라이언트가 서버에 연결할 수 없으면 보조 서버에 연결을 시도합니다.

Handled 속성이 True로 설정되면 클라이언트가 재연결을 시도하지 않음을 의미합니다.

TsgcWebSocketClient | Dropped Disconnections

연결이 설정된 후 피어가 데이터를 보내지 않으면 네트워크를 통해 패킷이 전송되지 않습니다. TCP는 유휴 프로토콜이므로 연결이 여전히 활성 상태라고 가정합니다.

연결 끊김 사유

- **애플리케이션 닫힘:** 프로세스가 완료되면 일반적으로 FIN 패킷을 전송하여 연결이 닫혔음을 다른 피어에 확인합니다. 그러나 프로세스가 충돌하면 이 패킷이 다른 피어에 전송된다는 보장이 없습니다.
- **Device Closes:** 디바이스가 닫히면, 대부분의 경우 이에 대한 알림이 없을 것입니다.
- **네트워크 케이블 분리:** 네트워크 케이블이 분리된 경우 이는 라우터가 닫히는 것과 동일합니다. 전송되는 데이터가 없으므로 연결이 닫히지 않습니다.
- **라우터로부터의 신호 손실:** 애플리케이션이 라우터로부터 신호를 잃어도, 연결은 여전히 활성 상태로 유지됩니다.

Half-Open 연결 끊김 감지

다음 메서드를 사용하여 연결 끊김을 감지하려고 시도할 수 있습니다

두 번째 연결

두 번째 연결을 열고 연결을 시도할 수 있지만, 여기에는 몇 가지 단점이 있습니다: 더 많은 리소스를 소비하고, 새 스레드를 생성하는 등. 또한 다른 피어가 재부팅된 경우 두 번째 연결은 작동하지만 첫 번째 연결은 작동하지 않습니다.

다른 피어 ping

half-open 연결로 ping 또는 어떤 메시지를 보내려고 하면 아무런 오류도 발생하지 않는 것을 볼 수 있습니다.

TCP Socket 수준에서 KeepAlive 활성화

TCP keep-alive 패킷은 단순히 시퀀스 번호가 연결의 현재 시퀀스 번호보다 1 작게 설정된 ACK입니다. 이러한 ACK 중 하나를 수신하는 호스트는 현재 시퀀스 번호에 대한 ACK로 응답합니다. Keep-alive는 연결의 원격 끝에 있는 컴퓨터가 여전히 사용 가능한지 확인하는 데 사용할 수 있습니다. TCP keep-alive는 다른 데이터나 상위 수준 keep-alive가 TCP 연결을 통해 전달되지 않은 경우 TCPKeepAlive.Time(기본값 7,200,000밀리초 또는 2시간)마다 한 번씩 전송될 수 있습니다. keep-alive에 대한 응답이 없으면 TCPKeepAlive.Interval초마다 한 번씩 반복됩니다. KeepAliveInterval은 기본적으로 1000밀리초입니다.

연결별 KeepAlive를 활성화하고 TCP 프로토콜이 연결이 활성 상태인지 여부를 확인하도록 허용할 수 있습니다. 끊긴 연결을 감지하려는 경우 (예: 네트워크 케이블을 뽑을 때) 이것이 선호되는 방법입니다.

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

TsgcWebSocketClient | Connect TCP Server

TsgcWebSocketClient는 WebSocket 서버에 연결할 수 있지만, 일반 TCP 서버에도 연결할 수 있습니다.

URL 속성

TCP 서버에 연결하는 가장 쉬운 방법은 **URL** 속성을 사용하고 **Active = true**로 설정하는 것입니다.

예제: 127.0.0.1 포트 5555에 연결

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

호스트, 포트 및 매개변수

Host 및 port 속성을 사용하여 TCP 서버에 연결할 수 있습니다.

예제: 127.0.0.1 포트 5555에 연결

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

TsgcWebSocketClient | Connections

TIME_WAIT

클라이언트가 서버로부터 연결 해제를 시작하면, 연결 해제 상태를 전달하기 위해 클라이언트와 서버 간에 교환이 이루어집니다. 프로세스가 완료되면, 클라이언트 소켓 연결은 가변 기간 동안 TIME_WAIT 상태에 들어갑니다. 이것은 정상적인 동작입니다. Windows 운영 체제에서 이 시간은 기본적으로 약 4분입니다.

다음 대안을 사용하여 이 동작을 줄이거나 제거할 수 있지만, 주의해서 수행하십시오.

REGEDIT

Windows Regedit를 사용하여 TIME_WAIT 값을 줄일 수 있습니다

1. Regedit를 열고 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\ Services\TCP/IP\Parameters 레지스트리 하위 키에 접근하십시오.
2. **TcpTimedWaitDelay**라는 이름의 새 REG_DWORD 값을 생성하십시오.
3. 값을 초 단위로 설정하십시오. 예: 값을 5로 설정하면 TIME_WAIT가 최대 5초까지 대기함을 의미합니다.
4. 저장하고 시스템을 다시 시작하십시오.

LINGER

TIME_WAIT 상태를 피하는 또 다른 옵션은 소켓 옵션 SO_LINGER를 사용하는 것입니다. 활성화되면 연결을 정상적으로 닫는 대신 클라이언트가 연결을 재설정하여 TIME_WAIT 상태를 피합니다.

LingerState 속성을 사용하여 이 옵션을 활성화할 수 있으며, 기본적으로 값은 -1입니다. 값을 0으로 설정하면 시간 초과 없이 소켓에서 연결을 끊을 때 연결이 리셋됩니다.

이 옵션은 아마도 가장 권장되지 않으며 최후의 수단으로만 사용해야 합니다.

TsgcWebSocketClient | WebSocket Redirections

클라이언트가 WebSocket 서버에 연결하면, 서버는 HTTP Response Code 30x를 반환할 수 있습니다. 응답 코드가 301이면 위치가 영구적으로 이동되었음을 의미하며, 새 URL이 Location HTTP Header에 제공됩니다.

WebSocket 클라이언트는 리디렉션을 자동으로 처리하므로, 서버 응답에 리디렉션이 포함되어 있음을 감지하면 현재 연결을 끊고 새 Location URL에 연결을 시도합니다.

예제

1. 클라이언트는 먼저 url ws://127.0.0.1:5000에 연결하려고 시도합니다
2. 서버는 301의 Response Code를 반환하며 ws://80.50.1.2:3000 값을 가진 Header Location을 포함합니다
3. 클라이언트는 서버로부터 Response를 읽고, 리디렉션임을 감지하고 Location을 읽습니다
 1. 먼저 현재 연결을 끊습니다.
 2. Location Header의 값(ws://80.50.1.2:3000)으로 URL 속성을 업데이트합니다
 3. 새 서버에 연결합니다.

TsgcWebSocketClient | Connect Secure Server

TsgcWebSocketClient는 보안 및 비보안 연결을 사용하여 WebSocket 서버에 연결할 수 있습니다.

URL 속성 또는 Host / Port 속성을 사용하여 보안 연결을 구성할 수 있습니다. [Connect to WebSocket Server](#)를 참조하십시오.

TLSOptions

TLSOptions 속성에는 보안 연결을 사용자 지정하는 속성이 있습니다. 가장 중요한 속성은 **version**으로, **TLS 프로토콜의 버전**을 지정합니다. 일반적으로 **TLS 속성을 true로** 설정하고 **TLSOptions.Version을 tlsUndefined로** 설정하면 대다수의 WebSocket 서버에 충분합니다.

TLSOptions.Version을 사용하면 서버에 연결하는 데 사용되는 TLS 버전을 설정하거나 클라이언트가 사용 가능한 모든 버전 중에서 TLS 버전을 협상하도록 할 수 있습니다 (값이 **tlsUndefined**일 때 기본값임).

TLS 프로토콜에 대해 서버에 연결하려고 시도할 때 오류가 발생하면, 대부분 서버가 설정한 것보다 최신 TLS 버전을 요구하는 경우일 가능성이 높습니다.

TLSOptions.IOHandler가 **iohOpenSSL**로 설정된 경우, **OpenSSL 라이브러리를 배포**해야 합니다(이 라이브러리들이 모든 TLS 작업을 처리하는 라이브러리입니다). [OpenSSL](#)에 관한 다음 문서를 확인하십시오.

TLSOptions.IOHandler가 **iohSChannel**로 설정된 경우 라이브러리를 배포할 필요가 없습니다(Windows만 지원됨).

TsgcWebSocketClient | Certificates

OpenSSL

서버가 클라이언트가 SSL 인증서를 사용하여 연결하도록 요구하는 경우, TsgcWebSocketClient의 TLSOptions 속성을 사용하여 인증서 파일을 설정하십시오. 인증서는 PEM 형식이어야 하므로, 인증서가 다른 형식인 경우 먼저 PEM으로 변환해야 합니다.

OpenSSL 라이브러리를 통한 연결은 TLSOptions.IOHandler = iohOpenSSL일 것을 요구합니다.

다음 속성을 구성하십시오:

- **CertFile:** PEM 형식의 인증서 경로입니다.
- **KeyFile:** 인증서의 개인 키 경로입니다.
- **RootCertFile:** 인증서 루트의 경로입니다.
- **Password:** 인증서가 비밀번호로 보호되는 경우 여기에 secret을 설정하십시오.

TsgcWebSocketClient | Certificates

SChannel

서버가 클라이언트에게 SSL 인증서를 사용하여 연결하도록 요구하는 경우, TsgcWebSocketClient의 TLSOptions 속성을 사용하여 인증서 파일을 설정하십시오.

SChannel을 통한 연결은 TLSOptions.IOHandler = iohSChannel을 요구합니다.

SChannel은 2가지 유형의 인증서 인증을 지원합니다:

1. **PFX 인증서** 사용
2. Windows 시스템에 이미 설치된 인증서의 **Hash Certificate**를 설정합니다.

PFX Certificate

PFX 인증서는 인증서와 개인 키를 포함하는 파일입니다. PEM 형식의 인증서가 있는 경우가 있으므로, 사용하기 전에 PFX로 변환해야 합니다.

PEM 인증서를 PFX로 변환하려면 다음 openssl 명령을 사용하십시오

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

인증서가 PFX 형식이 되면, 인증서를 배포하고 **TLSOptions.CertFile** 속성을 해당 경로로 설정하기만 하면 됩니다.

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

Hash Certificate

인증서가 이미 Windows 인증서 저장소에 설치되어 있는 경우, 인증서 지문만 알고 이를 TLSOptions.SChannel_Options 속성에 설정하면 됩니다.

인증서의 해시를 찾는 것은 **powershell**에서 인증서 컨테이너에 **dir** 명령을 실행하는 것만큼 쉽습니다.

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

해시는 16진수 **Thumbprint** 값입니다.

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

Thumbprint 값을 얻으면 **TLSOptions.SChannel_Options** 속성에 해시와 인증서 위치를 설정해야 합니다.

```
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

클라이언트가 보안 서버에 연결되면, 사용 중인 TLS 버전(TLS 1.2, TLS 1.3 등), 사용된 cipher, 강도 등에 대한 정보를 요청할 수 있습니다.

이 정보에 액세스하려면 SChannel Handler의 **GetInfo** 함수를 호출하십시오. SChannel Handler가 생성된 후 호출되는 **OnSSLAfterCreateHandler** 메서드를 사용하여 SSL Handler에 액세스할 수 있습니다. 클라이언트가 서버에 연결된 후 SSL Handler가 할당되어 있으면 **GetInfo** 함수를 호출하고, 성공하면 연결 데이터를 반환합니다.

```
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 | Client Send Text Message

클라이언트가 서버에 연결되면 텍스트 메시지를 보낼 수 있습니다. 텍스트 메시지를 보내려면 `WriteData()` 메서드를 호출하기만 하면 됩니다.

텍스트 메시지 보내기

텍스트 메시지를 보내려면 **`WriteData()`** 메서드를 호출하십시오. 이 메서드는 호출된 **동일한 스레드**에서 실행됩니다.

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

`QueueOptions.Text`가 **`qmNone`**과 다른 값을 가지면, 호출된 동일한 스레드에서 처리되는 대신 보조 스레드에서 처리됩니다. 기본적으로 이 옵션은 비활성화되어 있습니다.

텍스트 메시지를 보내고 응답 대기

`WriteAndWaitData()` 메서드를 호출하여 텍스트 메시지를 보내고 서버로부터 응답을 기다립니다. 함수는 수신된 텍스트 메시지를 반환합니다.

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

TsgcWebSocketClient | Client Send Binary Message

클라이언트가 서버에 연결되면 바이너리 메시지를 보낼 수 있습니다. 바이너리 메시지를 보내려면 `WriteData()` 메서드를 호출하기만 하면 됩니다.

바이너리 메시지 보내기

바이너리 메시지를 보내려면 **`WriteData()`** 메서드를 호출하십시오. 이 메서드는 호출된 **동일한 스레드**에서 실행됩니다.

```
TMemoryStream oStream = new TMemoryStream();  
...  
TsgcWebSocketClient1->WriteData(oStream);  
delete oStream;
```

`QueueOptions.Binary` 가 **`qmNone`**과 다른 값을 가진 경우, 호출되는 동일한 스레드에서 처리되는 대신 보조 스레드에서 처리됩니다. 기본적으로 이 옵션은 비활성화되어 있습니다.

바이너리 메시지를 전송하고 응답을 기다리기

`WriteAndWaitData()` 메서드를 호출하여 바이너리 메시지를 보내고 서버로부터의 응답을 기다리십시오. 이 함수는 수신된 바이너리 메시지를 반환합니다.

```
TsgcWebSocketClient1->WriteAndWaitDataData(oStream);
```

TsgcWebSocketClient | 클라이언트가 텍스트 및 바이너리 메시지 보내기

WebSocket 프로토콜은 텍스트 또는 바이너리 두 가지 유형의 메시지만 허용합니다. 그러나 동일한 메시지에서 바이너리 데이터를 텍스트와 함께 보낼 수는 없습니다.

이를 해결하는 한 가지 방법은 바이너리 메시지를 보내기 전에 헤더를 추가하고 수신될 때 이 바이너리 메시지를 디코딩하는 것입니다.

sgcWebSocket_Helpers에는 바이너리 패킷에 대한 짧은 설명을 설정하는 데 사용할 수 있는 2개의 함수가 있습니다. 이는 기본적으로 스트림에 헤더를 추가하며, 바이너리 패킷을 식별하는 데 사용됩니다.

바이너리 메시지를 전송하기 전에 스트림을 인코딩하는 메서드를 호출하십시오.

```
sgcWSStreamWrite("00001", oStream);
TsgcWebSocketClient1->WriteData(oStream);
```

바이너리 메시지가 수신되면, 스트림을 디코딩하기 위해 메서드를 호출하십시오.

```
sgcWSStreamRead(oStream, vID);
```

유일한 제한 사항은 바이너리 메시지를 식별하는 데 사용되는 텍스트의 최대 길이가 10자라는 것입니다(소스 코드에 액세스할 수 있는 경우 수정할 수 있음).

TsgcWebSocketClient | 텍스트 메시지 수신

클라이언트가 텍스트 메시지를 수신하면 **OnMessage** 이벤트가 발생합니다. 수신된 메시지의 문자열을 검색하려면 Text 매개변수를 읽으십시오.

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

기본적으로 클라이언트는 OnMessage 이벤트를 디스패치하기 위해 **neAsynchronous** 메서드를 사용합니다. 이는 이 이벤트가 **메인 스레드의 컨텍스트에서 실행됨**을 의미하므로, 예를 들어 폼의 컨트롤을 업데이트하는 것이 스레드 안전합니다.

클라이언트가 많은 메시지를 받거나 다른 스레드와의 동기화를 제어해야 하는 경우 NotifyEvents 속성을 **neNoSync**로 설정하십시오. 이는 OnMessage 이벤트가 **연결 스레드의 컨텍스트에서 실행됨**을 의미하므로, 폼의 컨트롤을 업데이트하거나 공유 객체에 액세스해야 하는 경우 자체 동기화 메서드를 구현해야 합니다.

TsgcWebSocketClient | Receive Binary Messages

클라이언트가 바이너리 메시지를 수신하면 **OnBinary** 이벤트가 발생합니다. 수신된 바이너리 메시지를 검색하려면 Data 매개변수를 읽으십시오.

```
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;
}
```

기본적으로 클라이언트는 OnMessage 이벤트를 디스패치하기 위해 **neAsynchronous** 방식을 사용하며, 이는 이 이벤트가 메인 스레드의 컨텍스트에서 실행됨을 의미하므로, 예를 들어 폼의 컨트롤을 업데이트하는 것이 스레드로부터 안전합니다.

클라이언트가 많은 메시지를 수신하거나 다른 스레드와의 동기화를 제어해야 하는 경우, NotifyEvents 속성을 **neNoSync**로 설정하십시오. 이는 OnMessage 이벤트가 **e연결 스레드의 컨텍스트에서 실행됨**을 의미하므로, 폼의 컨트롤을 업데이트하거나 공유 객체에 액세스해야 하는 경우 사용자가 직접 동기화 방법을 구현해야 합니다.

TsgcWebSocketClient | 클라이언트 인증

TsgcWebSocket 클라이언트는 4가지 유형의 인증을 지원합니다:

- **Basic:** WebSocket HandShake 중에 User와 Password를 Basic Authentication으로 인코딩한 HTTP 헤더를 보냅니다.
- **Token:** WebSocket 핸드셰이크 중에 HTTP 헤더로 토큰을 전송합니다. 서버가 요구하는 대로 필요한 토큰을 Authentication.Token.AuthToken에 설정하십시오.
- **Session:** 먼저 클라이언트가 서버에서 HTTP 세션을 요청하고 서버가 세션을 반환하면, WebSocket HandShake의 GET HTTP 헤더로 전달됩니다. (* sgcWebSockets 라이브러리 전용 인증 메서드).
- **URL:** 클라이언트가 WebSocket HandShake의 GET HTTP 헤더를 사용하여 권한 부여를 요청합니다. (* sgcWebSockets 라이브러리의 자체 인증 방법).

Authentication Basic

이것은 사용자와 비밀번호가 인코딩되어 HTTP Header로 전달되는 간단한 권한 부여 방법입니다. User와 Password를 설정하고 Basic Authorization 유형만 활성화하여 이 방법을 사용하십시오.

```
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;
```

인증 토큰

JWT를 사용하여 권한을 부여할 수 있습니다. 이를 위해서는 외부 도구(예: HTTP 연결, OAuth2 등)를 사용하여 토큰을 얻어야 합니다.

OAuth2 구성 요소를 연결하면 이 토큰을 자동으로 얻을 수 있습니다. [OAuth2](#)에 대해 자세히 알아보십시오.

AuthToken을 설정하고 Token Authentication을 활성화해야 합니다.

```
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;
```

인증 세션

먼저 클라이언트는 새 Session을 요청하는 HTTP 연결을 사용하여 서버에 연결합니다. 성공하면 서버는 SessionId를 반환하고 클라이언트는 WebSocket HandShake의 GET HTTP Header에 이 SessionId를 보냅니다. UserName과 Password를 설정하고 Session Authentication을 활성화해야 합니다.

```
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

이 인증 방법은 WebSocket HandShake의 GET HTTP Header에 사용자 이름과 비밀번호를 전달합니다.

```
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 | Client Exceptions

때때로 통신에서 오류가 발생합니다: 서버는 권한이 없거나 메시지 형식이 올바르지 않다는 이유로 연결을 끊을 수 있습니다. 오류가 포착되는 2개의 이벤트가 있습니다.

OnError

이 이벤트는 잘못된 메시지 유형, 잘못된 utf8 문자열 등과 같은 WebSocket 프로토콜 오류가 있을 때마다 발생합니다.

```
void OnError(TsgcWSConnection *Connection, const string Error)
{
    WriteLn("#error: " + Error);
}
```

OnException

이 이벤트는 활성 상태가 아닌 소켓에 쓰거나 존재하지 않는 객체에 액세스하는 등 예외가 있을 때마다 발생합니다.

```
void OnException(TsgcWSConnection *Connection, Exception *E)
{
    WriteLn("#exception: " + E->Message);
}
```

기본적으로 서버에 의해 연결이 닫히면, 예외가 발생합니다. 이러한 예외가 발생하지 않도록 하려면, **Options.RaiseDisconnectExceptions**를 비활성화하십시오.

TsgcWebSocketClient | WebSocket HandShake

WebSocket 프로토콜은 HTTP HandShake를 사용하여 HTTP 프로토콜에서 WebSocket 프로토콜로 업그레이드합니다. 이 핸드셰이크는 TsgcWebSocket Client 구성 요소에 의해 내부적으로 처리되지만, 서버가 추가 HTTP 헤더 정보를 요구하는 경우 자체 사용자 지정 HTTP 헤더를 추가할 수 있습니다.

Example: 이 HTTP 헤더 "Client: sgcWebSockets"를 추가해야 하는 경우

```
void OnHandshake(TsgcWSConnection *Connection, ref TStringList *Headers)
{
    Headers->Add("Client: sgcWebSockets");
}
```

TsgcWebSocketClient | Client Register Protocol

기본적으로 TsgcWebSocketClient는 SubProtocol을 사용하지 않습니다. WebSocket 서브프로토콜은 WebSocket 프로토콜 위에 구축되며 사용자 지정 메시지 프로토콜을 정의합니다. WebSocket 서브프로토콜의 예로는 MQTT, STOMP 등이 있습니다.

WebSocket SubProtocol 이름은 WebSocket HandShake에서 HTTP Header로 전송됩니다. 이 헤더는 서버에서 처리되며, 서버가 이 서브프로토콜을 지원하면 연결을 수락합니다. 지원되지 않으면 연결이 자동으로 닫힙니다.

예: SubProtocol 이름이 'myprotocol'인 websocket 서버에 연결합니다.

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

TsgcWebSocketClient | 클라이언트 프록시

TsgcWebSocket 클라이언트는 프록시를 통한 연결을 지원합니다. 프록시 연결을 구성하려면 TsgcWebSocket 클라이언트의 **Proxy** 속성을 채우기만 하면 됩니다.

```
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*는 일반 TCP 서버에 연결할 수 있는 TCP 클라이언트 구성 요소입니다.

소개

TsgcTCPClient는 일반 TCP 서버에 연결할 수 있는 TCP 클라이언트 구성 요소입니다. TLS/SSL 연결, 프록시 서버, 하트비트 모니터링, 자동 재연결(WatchDog) 및 TCP keep-alive를 지원합니다.

1. **TsgcTCPClient** 구성 요소를 폼에 놓습니다.
2. 사용 가능한 TCP Server에 연결하기 위해 **Host**와 **Port**(기본값은 80)를 설정하십시오.
3. 보안 연결이 필요한 경우 **TLS**를 True로 설정하십시오(기본값은 False).
4. 프록시 서버를 통해 연결하는 경우 선택적으로 **Proxy**를 구성하십시오.
5. 이벤트 처리: **OnConnect**, **OnDisconnect**, **OnMessage**, **OnBinary**, **OnError**, **OnException**.
6. 연결을 열려면 **Active**를 True로 설정하십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 속성 23개입니다.
- 메서드 — 7개의 public 메서드.
- 이벤트 — 처리할 수 있는 12개의 이벤트.

TsgcTCPClient

*TsgcTCPClient*는 일반 TCP 서버에 연결할 수 있는 TCP 클라이언트 구성 요소입니다.

소개

TsgcTCPClient는 일반 TCP 서버에 연결할 수 있는 TCP 클라이언트 구성 요소입니다. TLS/SSL 연결, 프록시 서버, 하트비트 모니터링, 자동 재연결(WatchDog) 및 TCP keep-alive를 지원합니다.

1. **TsgcTCPClient** 구성 요소를 폼에 놓습니다.
2. 사용 가능한 TCP Server에 연결하기 위해 **Host**와 **Port**(기본값은 80)를 설정하십시오.
3. 보안 연결이 필요한 경우 **TLS**를 True로 설정하십시오(기본값은 False).
4. 프록시 서버를 통해 연결하는 경우 선택적으로 **Proxy**를 구성하십시오.
5. 이벤트 처리: **OnConnect**, **OnDisconnect**, **OnMessage**, **OnBinary**, **OnError**, **OnException**.
6. 연결을 열려면 **Active**를 True로 설정하십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 속성 23개입니다.
- 메서드 — 7개의 public 메서드.
- 이벤트 — 처리할 수 있는 12개의 이벤트.

TsgcWebSocketServer

TsgcWebSocketServer는 서버 WebSocket 구성 요소를 구현하며 여러 스레드 클라이언트 연결을 처리할 수 있습니다.

소개

TsgcWebSocketServer는 서버 WebSocket 구성 요소를 구현하며 여러 스레드화된 클라이언트 연결을 처리할 수 있습니다. 이 구성 요소를 구성하려면 아래 단계를 따르십시오:

1. 폼에 TsgcWebSocketServer 구성 요소를 놓으십시오
2. Port를 설정합니다(기본값은 80). 방화벽 뒤에 있는 경우 구성해야 할 수 있습니다.
3. 허용되는 Specifications를 설정하십시오. 기본적으로 모든 specification이 허용됩니다.

RFC6455: 표준이며 권장되는 WebSocket 사양입니다.

Hixie76: 이것은 초안이며 Safari 4.2와 같은 오래된 브라우저에 대한 지원을 제공하려는 경우에만 Hixie76 연결을 설정하는 것이 권장됩니다

```
void __fastcall TForm1::OnTCPConnectEvent(TsgcWSConnection *aConnection, bool &Accept)
{
    aConnection->Transport = trpTCP;
    Accept = true;
}
```

5. 프로시저를 생성하고 Active = True 속성을 설정하십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 27개의 속성.
- 메서드 — 7개의 public 메서드.
- 이벤트 — 처리할 수 있는 22개의 이벤트.

Guides

- 시작
 - [Server Start](#)
 - [Server Bindings](#)
 - [Server Startup - Shutdown](#)
 - [Server Keep Active](#)
- Connections
 - [Server Keep Connections Alive](#)
 - [Server Plain TCP](#)
 - [Server Close Connection](#)
 - [Client Connections](#)
- 인증
 - [서버 인증](#)

- **Send Messages**
 - [Server Send Text Message](#)
 - [Server Send Binary Message](#)
- **메시지 수신**
 - [서버 텍스트 메시지 수신](#)
 - [서버 바이너리 메시지 수신](#)
- **SSL**
 - [Server SSL](#)
 - [Server SSL SChannel](#)

TsgcWebSocketServer | Start Server

서버를 시작할 때 먼저 설정해야 하는 것은 수신 대기 포트입니다. 기본적으로 포트 80으로 설정되어 있지만 임의의 포트도 변경할 수 있습니다. 포트가 설정되면, 서버를 시작하는 2가지 방법이 있습니다.

Active 속성

Active 속성을 true로 설정하면 서버는 구성된 포트에서 들어오는 모든 연결에 대한 수신 대기를 시작합니다.

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

Active 속성을 false로 설정하면 서버가 중지되고 모든 활성 연결을 닫습니다.

```
oServer->Active = false;
```

Start / Stop 메서드

Active 속성을 설정하면 동일한 스레드에서 서버가 시작/중지되는 반면, Start 및 Stop 메서드는 보조 스레드에서 실행됩니다.

```
oServer = new TsgcWebSocketServer();
oServer->Port = 80;
oServer->Start();
```

Stop() 메서드를 호출하면 서버가 중지되고 모든 활성 연결이 닫힙니다.

```
oServer->Stop();
```

보조 스레드에서 서버를 중지하고 시작하려면 **ReStart** 메서드를 사용할 수 있습니다.

서버를 닫은 후 Port를 변경하여 다른 포트에서 수신을 시작하려면, 서버를 닫은 후 **Bindings.Clear()** 메서드를 호출하여 이전 바인딩을 모두 삭제하십시오. 그렇지 않으면 서버가 이전 바인딩에 바인딩하려고 시도합니다.

TsgcWebSocketServer | Server Bindings

기본적으로 **Port** 속성만 채우면 서버는 모든 IP에 수신 포트를 바인딩합니다. 따라서 예를 들어 127.0.0.1, 80.54.11.22 및 12.55.41.17의 3개 IP가 있는 경우 서버는 3개 IP 모두에서 이 포트를 바인딩합니다.

일반적으로 필요한 IP에만 바인딩하는 것이 권장됩니다. 여기에서 Bindings 속성을 사용할 수 있습니다.

Port 속성을 사용하는 대신 Bindings 속성을 사용하여 필요한 IP와 Port를 입력하십시오.

예: Port 5555를 IP 127.0.0.1 및 IP 80.58.25.40에 바인딩합니다

```
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;
```

서버를 닫은 후 다른 포트에서 수신 대기를 시작하기 위해 Port를 변경하면, 서버를 닫은 후 **Bindings.Clear()** 메서드를 호출하여 모든 이전 바인딩을 삭제하십시오. 그렇지 않으면 서버가 이전 바인딩에 바인딩을 시도합니다.

TsgcWebSocketServer | Server Startup Shutdown

서버의 필요한 모든 구성을 설정하면, 서버가 시작되었을 때와 중지되었을 때를 알 수 있는 2개의 유용한 이벤트가 있습니다.

OnStartup

이 이벤트는 서버가 시작되어 새 연결을 처리할 수 있을 때 발생합니다.

```
void OnStartup(TObject *Sender)
{
    WriteLn("#server started");
}
```

OnShutdown

이 이벤트는 서버가 중지되고 더 이상 연결이 수락되지 않은 후에 발생합니다.

```
void OnShutdown(TObject *Sender)
{
    WriteLn("#server stopped");
}
```

TsgcWebSocketServer | Server Keep Active

서버가 시작된 후, 때때로 어떤 이유로든 중지될 수 있습니다. 예기치 않은 종료 후 서버를 재시작하려면 WatchDog 속성을 사용할 수 있습니다.

WatchDog

WatchDog가 활성화된 경우 서버가 종료를 감지하면 WatchDog는 서버가 다시 활성화될 때까지 X초마다 다시 시작하려고 시도합니다.

예제: 예기치 않은 중지 후 무제한 시도로 10초마다 다시 시작합니다.

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

TsgcWebSocketServer | Server SSL

서버는 **SSL Certificates**를 사용하도록 구성할 수 있습니다. 서버 인증서가 있는 프로덕션 서버를 얻으려면 **잘 알려진 공급자**: Namecheap, GoDaddy, Thawte 등으로부터 인증서를 **구매**해야 합니다. **테스트 목적**으로는 **자체 서명 인증서**를 사용할 수 있습니다(자체 서명 인증서를 사용하는 Demos/Chat 예제 확인).

인증서는 **PEM 형식**이어야 합니다. PEM(Privacy Enhanced Mail에서 유래)은 RFC 1421~1424에 정의되어 있으며, 이것은 공개 인증서만 포함할 수 있는(Apache 설치 및 CA 인증서 파일 /etc/ssl/certs와 같이) 컨테이너 형식이거나, 공개 키, 개인 키 및 루트 인증서를 포함한 전체 인증서 체인을 포함할 수 있습니다. 단일 pem 인증서를 생성하려면 개인 키 파일을 열고, 콘텐츠를 복사하여 인증서 파일에 붙여넣으십시오.

예제:

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-----
```

SSL을 활성화하려면 **SSL 속성을 활성화**하고 **CertFile**, **KeyFile** 및 **RootFile** 경로를 구성하십시오. 인증서에 전체 인증서 체인(공개 키, 비공개 키 등)이 포함된 경우, 모든 경로를 동일한 인증서로 설정하기만 하면 됩니다.

설정해야 하는 또 다른 속성은 **SSLOptions.Port**입니다. 이것은 보안 연결에 사용되는 포트입니다.

간단한 SSL 구성

예: IP 127.0.0.1과 Port 443에서 SSL을 구성합니다.

```
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

서버가 둘 이상의 IP 및 port에서 수신 대기하도록 구성할 수 있습니다. 작동 방식을 설명하는 [Binding 글](#)을 확인하십시오. 서버는 SSL 연결과 비-SSL 연결을 동시에 허용하도록 구성할 수 있습니다(물론 서로 다른 port에서 수신 대기). 두 개의 서로 다른 port에 바인딩하고 한 port는 SSL 연결용으로, 다른 port는 비-SSL 연결용으로 구성하기만 하면 됩니다.

예제: IP 127.0.0.1, 포트 80(비암호화) 및 443(SSL)에서 서버 구성

```
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

Indy 기반 서버 구성 요소(**TsgcWebSocketServer**, **TsgcWebsocketHTTPServer**)는 OpenSSL 대신 **Windows SChannel**(Secure Channel)을 TLS 공급자로 사용할 수 있습니다. SChannel은 네이티브 Windows TLS 구현이므로 외부 DLL이 필요하지 않습니다.

작동 방식

서버 구성 요소에서 SSL이 활성화되어 있고 IOHandler가 **iohSChannel**로 설정된 경우, 서버는 Windows SChannel API를 사용하여 모든 TLS 작업을 처리하는 **TsgcIdServerIOHandlerSSLSChannel** 인스턴스를 생성합니다. 들어오는 모든 클라이언트 연결에 대해 서버는 SChannel 공급자를 통해 TLS 핸드셰이크를 수행하여 프로토콜 버전, 암호화 스위트, 그리고 구성된 인증서 바인딩을 협상합니다.

SChannel은 **Windows Certificate Store** 또는 **PFX 파일**(.pfx / .p12)에서 인증서를 읽습니다. PEM 파일이 필요 없고 OpenSSL 라이브러리를 배포할 필요가 없습니다.

구성

Indy 서버에서 SChannel을 활성화하려면 다음 속성을 구성하십시오:

1. **SSL** 속성을 **True**로 설정하십시오.
2. **SSLOptions.IOHandler**를 **iohSChannel**로 설정하십시오.
3. **SSLOptions.Version**을 원하는 TLS 버전(tls1_2, tls1_3, ...)으로 설정하십시오.
4. **SSLOptions.Port**를 보안 연결에 사용되는 포트로 설정하십시오.
5. 아래에 설명된 두 가지 방법 중 하나를 사용하여 인증서를 구성하십시오.

SChannel_Options 속성

SSLOptions.SChannel_Options 하위 속성은 SChannel 전용 설정을 노출합니다:

속성	Description
CertHash	Windows Certificate Store에 설치된 인증서의 지문(16진수 해시)입니다.
CertStoreName	검색할 인증서 저장소: scsnMY (Personal), scsnRoot , scsnTrust , scsnCA .
CertStorePath	저장소 위치: scspStoreLocalMachine 또는 scspStoreCurrentUser .
CipherList	선택적 콜론으로 구분된 cipher 알고리즘 목록입니다(예: CALG_AES_256:CALG_AES_128). 비어 있으면 시스템 기본값을 사용할 것을 의미합니다.
UseLegacyCredentials	True인 경우, SCH_CREDENTIALS 대신 레거시 SCHANNEL_CRED 구조체를 사용합니다. 최신 API를 지원하지 않는 이전 Windows 버전의 경우 이를 활성화하십시오.

또한, 일반 **SSLOptions** 속성 **CertFile** 및 **Password** 는 PFX 파일에서 인증서를 로드할 때 사용됩니다.

Windows Store의 인증서

인증서가 이미 Windows Certificate Store에 설치된 경우, 인증서 **thumbprint**를 제공하고 위치를 표시하십시오. 지문을 찾으려면 **PowerShell**을 열고 다음을 실행하십시오:

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

Thumbprint 열은 필요한 16진수 해시를 보여줍니다.

```
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;
```

PFX 파일의 인증서

PFX (.pfx 또는 .p12) 인증서 파일이 있는 경우 SSLOptions에서 **CertFile** 과 **Password** 속성을 설정하십시오. SChannel 은 시작 시 인증서를 가져옵니다.

```
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;
```

PEM 인증서와 개인 키가 있는 경우, 먼저 OpenSSL을 사용하여 이를 PFX 형식으로 변환하십시오.

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

TLS 버전

서버가 허용하는 TLS 버전을 제어하려면 **SSLOptions.Version** 속성을 사용하십시오:

Value	Description
tls1_0	TLS 1.0 (권장되지 않음)
tls1_1	TLS 1.1
tls1_2	TLS 1.2 (권장)
tls1_3	TLS 1.3
tlsUndefined	TLS 1.0, 1.1 및 1.2 수락

Cipher List

기본적으로 SChannel은 시스템 암호 구성을 사용합니다. **SChannel_Options.CipherList**를 콜론으로 구분된 알고리즘 이름 목록으로 설정하여 허용되는 암호를 제한할 수 있습니다. 예:

```
CALG_AES_256:CALG_AES_128
```

Windows 기본값을 사용하려면 이 속성을 비워 두십시오.

Legacy Credentials

Windows Server 2019 이전 버전은 최신 **SCH_CREDENTIALS** API를 지원하지 않을 수 있습니다. 이전 Windows 버전에서 서버 시작에 실패하면 레거시 **SCHANNEL_CRED** 구조를 대신 사용하도록 **SChannel_Options.UseLegacyCredentials**를 **True**로 설정하십시오. 구성 요소는 대부분의 경우 Windows 버전을 자동으로 감지하지만, 필요한 경우 legacy 모드를 강제할 수 있습니다.

OpenSSL 대비 장점

- **외부 DLL 불필요:** SChannel 은 Windows에 내장되어 있으므로 libeay32.dll / ssleay32.dll 또는 libcrypto / libssl을 배포할 필요가 없습니다.
- **Windows Certificate Store 통합:** 운영 체제에서 이미 설치되고 관리되는 인증서를 사용하십시오.
- **자동 업데이트:** TLS 개선 사항 및 보안 패치가 Windows Update를 통해 적용됩니다.

Notes

- SChannel은 **Windows에서만** 사용할 수 있습니다. 크로스 플랫폼 서버의 경우 OpenSSL(iohOpenSSL)을 사용하십시오.
- 서버에는 인증서와 연결된 **개인 키**가 있어야 합니다. Windows Store 방법을 사용할 때 인증서는 개인 키와 함께 가져와야 합니다.
- Certificate Store 방법을 사용하는 프로덕션 서버의 경우, **Local Machine** 저장소 (scspStoreLocalMachine)를 사용하는 것이 권장되므로 서비스를 실행하는 사용자 계정에 관계없이 인증서를 사용할 수 있습니다.
- **PFX** (.pfx / .p12) 인증서 파일만 지원됩니다. PEM 파일이 있는 경우 먼저 PFX 형식으로 변환하십시오.

TsgcWebSocketServer | Server Verify Certificate

기본적으로 서버는 피어 인증서를 검증하지 않습니다. 서버가 클라이언트 인증서를 검증하도록 구성하려면 다음 단계를 구현하십시오:

1. `SSLOptions.VerifyCertificate = true` 속성을 설정하십시오

`OnSSLVerifyPeer` 이벤트를 처리하고 다음 코드를 구현하여 클라이언트가 인증서로 연결할 때마다 알림을 받으십시오.

```
void OnSSLSSLVerifyPeer(System::TObject* Sender, TIdX509* Certificate, bool &Accept)
{
    // ... validate the certificate
    if (Certificate_OK)
        Accept = true;
    else
        Accept = false;
}
```

`OnSSLVerifyPeer` 이벤트는 클라이언트가 인증서를 제공하는 경우에만 호출되며, 클라이언트가 인증서를 제공하지 않으면 이벤트가 발생하지 않습니다.

인증서를 사용하는 **SSL 연결만 허용**하도록 서버를 구성할 수 있습니다. 이렇게 하려면 다음 속성을 설정하십시오:

- `SSLOptions.VerifyCertificate_Options.FailIfNoCertificate = true`

클라이언트가 인증서를 제공하지 않으면 SSL 핸드셰이크에서 연결이 닫힙니다.

TsgcWebSocketServer | Server Keep Connections Alive

클라이언트가 서버에 연결되면, 때때로 신호 불량, 연결 오류 등으로 인해 연결이 닫힐 수 있습니다. HeartBeat를 사용하여 연결을 활성 상태로 유지하십시오.

HeartBeat

HeartBeat 속성은 **연결을 유지하기 위해 X 초마다 Ping을 보낼 수** 있게 합니다. 일부 클라이언트는 피어 간에 데이터가 교환되지 않으면 TCP 연결을 닫습니다. HeartBeat은 특정 간격으로 ping을 보내 이 문제를 해결합니다. 일반적으로 이것으로 연결을 활성 상태로 유지하기에 충분하지만, X초 후에 클라이언트로부터 응답을 받지 못하면 연결을 닫으려는 경우 TimeOut 간격을 설정할 수 있습니다.

예제: 30초마다 연결된 모든 클라이언트에 ping 보내기

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

TsgcWebSocketServer | Server Plain TCP

WebSocket 서버는 WebSocket, HTTP, SSE 및 기타 프로토콜을 수락하지만 일반 TCP 연결과도 함께 작동할 수 있습니다. [TCP 연결](#)에 대해 자세히 알아보십시오.

TCP 연결을 처리하는 데 사용할 수 있는 2개의 이벤트가 있습니다.

OnTCPConnect

이 이벤트는 클라이언트가 서버에 연결한 후 클라이언트와 서버 간 핸드셰이크 전에 호출됩니다. OnConnect 이벤트는 클라이언트가 메시지를 보낸 후에만 발생합니다(서버가 어떤 프로토콜이 사용되고 있는지 감지할 수 있도록).

이 이벤트를 통해 새 클라이언트가 서버에 연결하려고 한다는 것을 알 수 있으며, 서버는 연결을 수락하거나 거부할 수 있습니다. 기본적으로 서버는 항상 연결을 수락합니다.

OnUnknownProtocol

이 이벤트는 서버가 클라이언트로부터 첫 번째 메시지를 수신했지만 알려진 프로토콜 중 어느 것인지 감지할 수 없을 때 호출됩니다. 이 이벤트에서 서버는 프로토콜을 수락하거나 거부할 수 있습니다.

OnConnect

이 이벤트는 성공적이고 완전한 연결 후에 발생합니다. 연결이 일반 TCP인 경우, OnUnknownProtocol 이벤트에서 프로토콜이 수락된 후에 발생합니다.

TsgcWebSocketServer | 서버 연결 종료

단일 Connection은 Close 또는 Disconnect 메서드를 사용하여 닫을 수 있습니다.

Disconnect

[TsgcWSConnection](#)에는 Disconnect()라는 메서드가 있어 소켓 수준에서 연결을 끊을 수 있습니다. 이 메서드를 호출하면 소켓이 클라이언트로부터 어떤 응답도 기다리지 않고 직접 끊어집니다. 이 메서드와 함께 Close Code를 전송할 수 있습니다.

Close

[TsgcWSConnection](#)에는 Close()라는 메서드가 있어, 클라이언트에게 연결을 닫도록 요청하는 메시지를 보낼 수 있습니다. 클라이언트가 이 메시지를 수신하면 연결을 닫아야 하며 서버는 연결이 닫혔다는 알림을 받습니다. 이 메서드로 Close Code를 보낼 수 있습니다.

DisconnectAll

모든 활성 연결을 끊습니다. 이 메서드는 서버가 수신 대기를 중지하기 전에 자동으로 호출되지만, 언제든지 이 메서드를 호출할 수 있습니다.

TsgcWebSocketServer | Client Connections

활성 클라이언트 연결에 액세스하려면 Connections 속성을 사용하여 목록을 반복하고 클라이언트 연결 클래스에 액세스할 수 있습니다. Connections 속성은 스레드된 목록에 액세스하므로, 먼저 목록을 잠그고 완료되면 목록을 잠금 해제하십시오.

```
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

TsgcWebSocket 서버는 3가지 유형의 인증을 지원합니다:

- **Basic:** WebSocket HandShake 중에 Basic Authorization으로 인코딩된 User와 Password가 포함된 HTTP Header를 읽습니다.
- **Session:** 먼저 클라이언트가 서버로부터 HTTP 세션을 요청하고 서버가 세션을 반환하면, 이는 WebSocket HandShake의 GET HTTP Header에 전달됩니다. (* sgcWebSockets 라이브러리 자체의 권한 부여 방법).
- **URL:** WebSocket HandShake의 GET HTTP Header를 사용하여 요청 권한 부여를 읽습니다. (* sgcWebSockets 라이브러리 고유의 권한 부여 방법).

AuthUsers 속성을 사용하여 인증된 사용자 목록을 설정할 수 있습니다. 다음 형식으로 사용자를 설정하기만 하면 됩니다:
user=password

OnAuthentication

서버가 클라이언트로부터 Authentication Request를 수신할 때마다, 이 이벤트가 호출되어 사용자가 인증되었는지 여부를 나타냅니다.

연결을 수락할지 여부를 결정하려면 Authenticated 매개변수를 사용하십시오.

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

OnUnknownAuthentication

기본적으로 지원되지 않는 인증 유형(예: JWT)이라도, 이 이벤트를 사용하여 연결을 수락하거나 거부할 수 있습니다. 매개변수를 읽고 연결을 수락할지 결정하기만 하면 됩니다.

```
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

클라이언트가 서버에 연결되면, 서버는 텍스트 메시지를 보낼 수 있습니다. 텍스트 메시지를 보내려면, 단일 클라이언트에게 메시지를 보내기 위해 WriteData() 메서드를 호출하거나, 모든 클라이언트에게 메시지를 보내기 위해 Broadcast를 사용하십시오.

텍스트 메시지 보내기

텍스트 메시지를 보내려면 **WriteData()** 메서드를 호출하십시오.

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

QueueOptions.Text가 **qmNone**과 다른 값을 가지면, 호출된 동일한 스레드에서 처리되는 대신 보조 스레드에서 처리됩니다. 기본적으로 이 옵션은 비활성화되어 있습니다.

QueueOptions는 **IOHandlerOptions.IOHandlerType = iohIOCP** 속성에서는 작동하지 않습니다(IOCP 아키텍처로 인해 이 기능은 지원되지 않습니다).

TsgcWSConnection에서도 WriteData() 메서드를 호출할 수 있습니다. **예제:** 클라이언트가 서버에 연결할 때 클라이언트에 메시지를 보냅니다.

```
void OnConnect(TsgcWSConnection *Connection)
{
    Connection->WriteData("Hello From Server");
}
```

연결된 모든 클라이언트에 메시지를 보냅니다.

연결된 모든 클라이언트에 텍스트 메시지를 보내려면 **Broadcast()** 메서드를 호출하십시오.

```
TsgcWebSocketServer1->Broadcast("Hello From Server");
```

TsgcWebSocketServer | Server Send Binary Message

클라이언트가 서버에 연결되면, 서버는 바이너리 메시지를 보낼 수 있습니다. 바이너리 메시지를 보내려면 WriteData() 메서드를 호출하여 단일 클라이언트에 메시지를 보내거나, Broadcast를 사용하여 모든 클라이언트에 메시지를 보내십시오.

바이너리 메시지 보내기

바이너리 메시지를 보내려면 **WriteData()** 메서드를 호출하십시오.

```
TsgcWebSocketServer1->WriteData("guid", new TMemoryStream());
```

QueueOptions.Binary에 **qmNone**과 다른 값이 있으면, 호출되는 동일한 스레드에서 처리되는 대신 보조 스레드에서 처리됩니다. 기본적으로 이 옵션은 비활성화되어 있습니다.

속성 **IOHandlerOptions.IOHandlerType = iohIOCP**인 경우 **QueueOptions**는 작동하지 않습니다(**IOCP 아키텍처로 인해 이 기능은 지원되지 않습니다**).

TsgcWSConnection에서도 WriteData() 메서드를 호출할 수 있습니다. **예제:** 클라이언트가 서버에 연결할 때 클라이언트에 메시지를 보냅니다.

```
void OnConnect(TsgcWSConnection *Connection)
{
    Connection->WriteData(new TMemoryStream());
}
```

연결된 모든 클라이언트에 메시지를 보냅니다.

Broadcast() 메서드를 호출하여 연결된 모든 클라이언트에 바이너리 메시지를 보내십시오.

```
TsgcWebSocketServer1->Broadcast(new TMemoryStream());
```

TsgcWebSocketServer | Server Receive Text Message

서버가 텍스트 메시지를 수신하면 **OnMessage** 이벤트가 발생합니다. 수신된 메시지의 문자열을 검색하려면 Text 매개변수를 읽으십시오.

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

기본적으로 서버는 OnMessage 이벤트를 디스패치하기 위해 **neAsynchronous** 방법을 사용합니다. 이는 이 이벤트가 **Main Thread의 컨텍스트에서 실행됨**을 의미하므로, 예를 들어 폼의 컨트롤을 업데이트하는 것이 스레드 안전합니다.

서버가 많은 메시지를 수신하거나 다른 스레드와의 동기화를 제어해야 하는 경우, NotifyEvents 속성을 **neNoSync**로 설정하십시오. 이는 OnMessage 이벤트가 **연결 스레드의 컨텍스트에서 실행됨**을 의미하므로, 폼의 컨트롤을 업데이트하거나 공유 객체에 접근해야 하는 경우 자체 동기화 메서드를 구현해야 합니다.

TsgcWebSocketServer | Server Receive Binary Message

서버가 바이너리 메시지를 수신하면 **OnBinary** 이벤트가 발생합니다. 수신된 바이너리 메시지를 검색하려면 Data 매개변수를 읽으십시오.

```
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();
}
```

기본적으로 서버는 OnMessage 이벤트를 디스패치하기 위해 **neAsynchronous** 방법을 사용합니다. 이는 이 이벤트가 **Main Thread의 컨텍스트에서 실행됨**을 의미하므로, 예를 들어 폼의 컨트롤을 업데이트하는 것이 스레드 안전합니다.

서버가 많은 메시지를 수신하거나 다른 스레드와의 동기화를 제어해야 하는 경우, NotifyEvents 속성을 **neNoSync**로 설정하십시오. 이는 OnMessage 이벤트가 **연결 스레드의 컨텍스트에서 실행됨**을 의미하므로, 폼의 컨트롤을 업데이트하거나 공유 객체에 접근해야 하는 경우 자체 동기화 메서드를 구현해야 합니다.

TsgcWebSocketServer | Server Read Headers from Client

WebSocket 서버에 클라이언트가 연결되면, 클라이언트 연결에 대한 정보가 있는 **헤더** 목록을 보냅니다. 이러한 클라이언트 헤더를 읽으려면, 서버가 클라이언트로부터 헤더를 수신하고 응답을 보내기 전에 호출되는 서버 구성 요소의 **OnHandshake** 이벤트를 사용할 수 있습니다.

클라이언트 헤더는 **TsgcWSConnectionServer**의 **HeadersRequest** 속성에 저장됩니다.

```
void OnServerHandshake(TsgcWSConnection *Connection, TStringList *Headers)
{
    ShowMessage(dynamic_cast<TsgcWSConnectionServer*>(Connection)->HeadersRequest->Text);
}
```

TsgcWebSocketHTTPServer

AResponseInfo.

소개

TsgcWebSocketHTTPServer는 서버 WebSocket 구성 요소를 구현하며 [TsgcWebSocketServer](#)처럼 여러 스레드 클라이언트 연결을 처리할 수 있고, 내장 HTTP 서버를 사용하여 HTML 페이지를 제공할 수 있으며, WebSocket 연결과 HTTP 요청에 동일한 포트를 공유합니다.

이 구성 요소를 구성하려면 아래 단계를 따르십시오:

1. Form에 TsgcWebSocketHTTPServer 구성 요소를 놓으십시오
2. Port를 설정합니다(기본값은 80). 방화벽 뒤에 있는 경우 구성해야 할 수 있습니다.
3. 허용되는 Specifications를 설정하십시오. 기본적으로 모든 specification이 허용됩니다.

RFC6455: 표준이며 권장되는 WebSocket 사양입니다.

Hixie76: 이것은 초안이며 Safari 4.2와 같은 오래된 브라우저에 대한 지원을 제공하려는 경우에만 Hixie76 연결을 설정하는 것이 권장됩니다

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

OnHTTPUploadBeforeCreatePostStream: 이 이벤트는 헤더를 읽은 후 그리고 post stream이 생성되기 전에 호출됩니다.

* 경우에 따라, 요청되지 않은 연결로 인해 CPU 소비가 높아질 수 있습니다. 이러한 경우, HTTP 요청이면 오류 500을 반환하거나 Unknown Protocol 요청이면 연결을 닫으십시오.

5. 프로시저를 만들고 Active = true 속성을 설정하십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 33개의 속성.
- 메서드 — 9개의 public 메서드.
- 이벤트 — 처리할 수 있는 36개의 이벤트.

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](#)
- **Other**
 - [HTTP Server Sessions](#)

TsgcWebSocketHTTPServer | HTTP Server Requests

OnCommandGet을 사용하여 HTTP 클라이언트 요청을 처리하십시오. 다음 매개변수를 사용하십시오:

- **RequestInfo:** HTTP 요청 정보를 포함합니다.
- **ResponseInfo:** HTTP Request에 대한 HTTP 응답입니다.
 - **ContentText:** 텍스트 형식의 응답입니다.
 - **ContentType:** Content-Type의 유형입니다.
 - **ResponseNo:** HTTP 응답 번호입니다. 예: 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

이 이벤트를 사용하여 HTTP 응답을 사용자 지정하십시오. 예를 들어, 일부 엔드포인트는 권한 부여 체계를 사용하고 다른 엔드포인트는 권한 부여 없이 액세스할 수 있도록 하려면 options 매개변수를 사용하여 허용하거나 비활성화하십시오. 아래는 Authorization Basic이 활성화되어 있지만 사용자가 /public 엔드포인트를 요청할 때 권한 부여가 필요하지 않은 예입니다.

```
void __fastcall TForm1::OnBeforeCommand(TsgcWSConnection* aConnection, TIdHTTPRequestInfo* ARequestInfo, TIdHTTPF
{
  if (ARequestInfo->Document == "/public")
    aOptions << hcoAuthorizedBasic;
}
```

TsgcWebSocketHTTPServer | HTTP Dispatch Files

클라이언트가 파일을 요청하면 **OnCommandGet** 이벤트가 발생하지만, **DocumentRoot** 속성을 사용하여 파일을 자동으로 디스패치할 수 있습니다.

예: **DocumentRoot**를 **c:/www/files**로 설정하면, 새 파일이 요청될 때마다 서버는 이 폴더에서 검색합니다. 파일이 존재하면 자동으로 디스패치됩니다.

TsgcWebSocketHTTPServer | HTTP/2 Server

sgcWebSockets HTTP 서버는 HTTP/1.1 및 HTTP/2.0 요청을 처리할 수 있게 합니다. 서버의 HTTP2Options를 사용하여 HTTP/2 프로토콜을 활성화할 수 있습니다.

서버가 HTTP/2 프로토콜 요청을 수락할 수 있도록 **HTTP2Options.Enabled = true** 로 설정하십시오. 요청은 HTTP/1.1 프로토콜과 정확히 동일하게 사용자가 처리할 수 있습니다, [자세히 읽기](#).

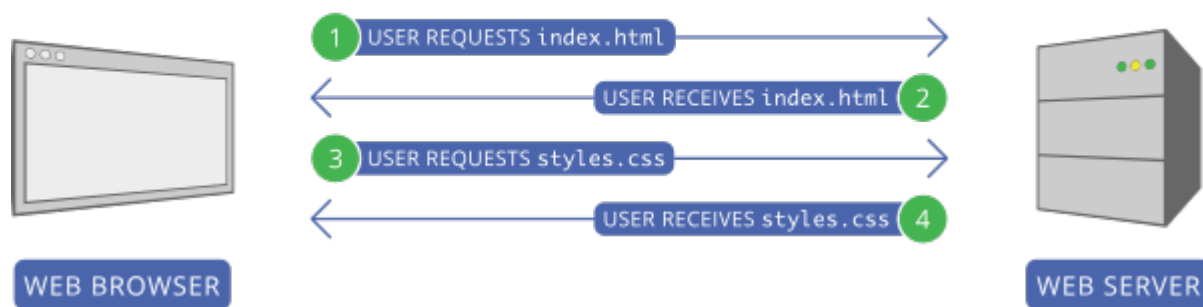
HTTP/2 프로토콜이 활성화되어 있어도 서버는 여전히 HTTP/1.1 요청을 지원합니다.

기본적으로 새 HTTP/2 연결이 있을 때 OnConnect 및 OnDisconnect 이벤트는 호출되지 않지만, HTTP2Options.Events 속성에 액세스하여 이를 수정할 수 있습니다. 여기에서 새 HTTP/2 연결 및/또는 연결 해제가 있을 때마다 알림을 받을지 여부를 사용자 지정할 수 있습니다.

TsgcWebSocketHTTPServer | HTTP/2 Server Push

HTTP는 일반적으로 요청/응답 패턴으로 작동합니다. 클라이언트가 리소스에 대한 REQUEST를 SERVER로 보내면, SERVER는 요청된 리소스나 오류와 함께 RESPONSE를 보냅니다. 일반적으로 브라우저와 같은 클라이언트는 서버가 제공하는 자산에 대해 일련의 요청을 합니다.

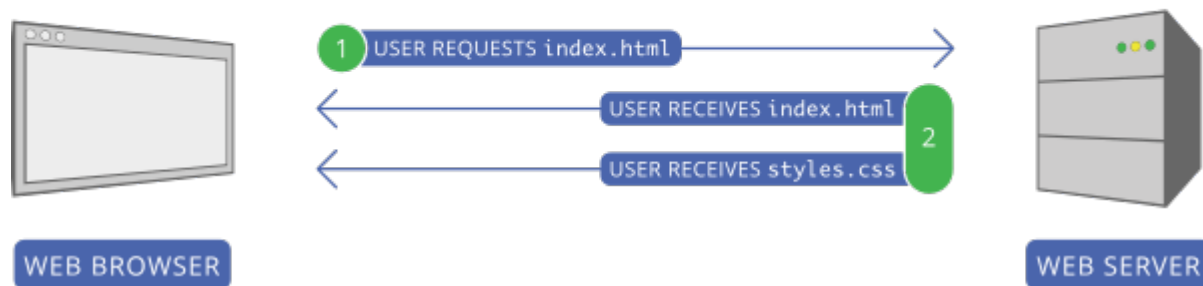
TYPICAL WEB SERVER COMMUNICATION



이 접근 방식의 주요 문제는 클라이언트가 먼저 리소스를 가져오는 요청을 보내고(예: index.html), 서버가 응답을 보낼 때까지 기다리고, 콘텐츠를 읽은 다음, 다른 모든 요청(예: styles.css)을 해야 한다는 것입니다.

HTTP/2 서버 푸시는 이 문제를 해결하려고 합니다. 클라이언트가 파일을 요청할 때 서버가 이 파일에 추가 파일이 필요하다고 판단하면 해당 파일이 자동으로 클라이언트에 PUSH됩니다.

WEB SERVER COMMUNICATION WITH HTTP/2 SERVER PUSH



위 스크린샷에서 클라이언트는 먼저 index.html을 요청합니다. 서버는 이 요청을 읽고 응답으로 2개의 파일을 보냅니다: index.html과 styles.css. 따라서 styles.css를 얻기 위한 두 번째 요청을 피합니다.

Server Push 구성

위의 스크린샷을 따라, /index.html 파일에 대한 새 요청이 있을 때마다 서버가 index.html과 styles.css를 보내도록 서버를 구성할 수 있습니다.

각 요청을 push promise 목록과 연결하려면 **PushPromiseAddPreLoadLinks** 메서드를 사용하십시오.

```

TsgcWebSocketHTTPServer *server = new TsgcWebSocketHTTPServer(this);
TStringList *oLinks = new TStringList();
try
{
    oLinks->Add("/styles.css");
    server->PushPromiseAddPreLoadLinks("/index.html", oLinks);
}
  
```

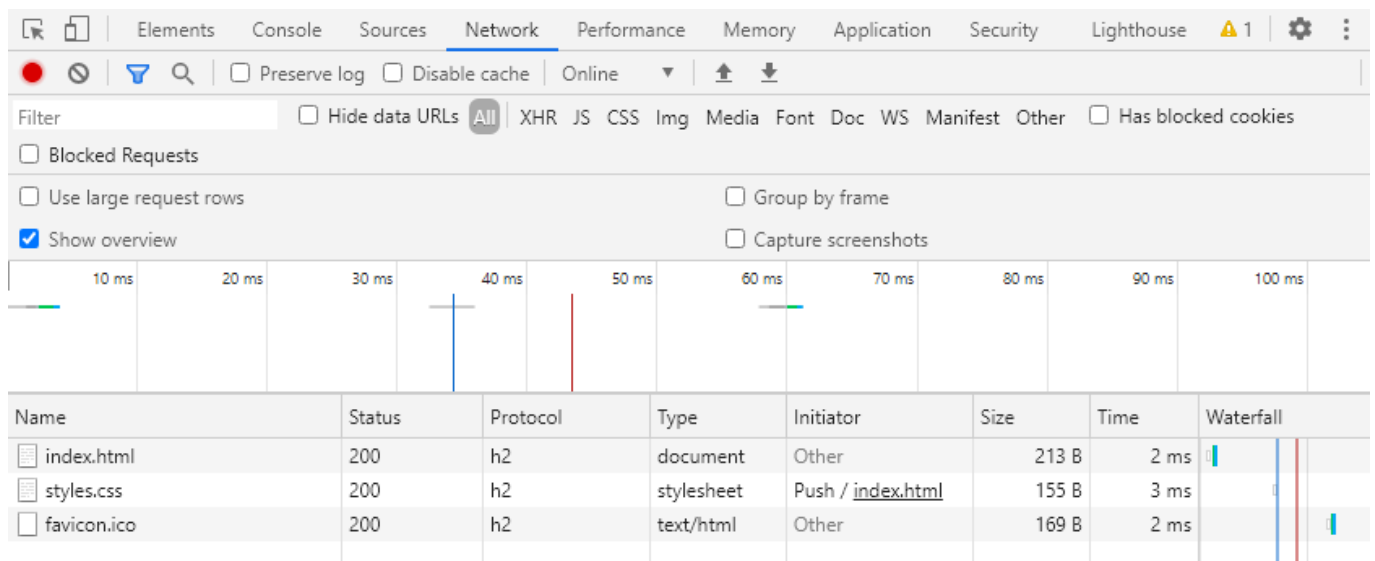
```

__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;
    }
}

```

Chrome 개발자 도구를 사용하여 styles.css 파일이 클라이언트로 어떻게 푸시되는지 볼 수 있습니다.



TsgcWebSocketHTTPServer | HTTP/2 Alternate Service

Alt-Svc HTTP header는 동일한 리소스를 다른 서비스 또는 프로토콜에서 도달할 수 있음을 클라이언트에 알리는 데 사용됩니다. 이는 예를 들어 서버가 HTTP/2를 지원함을 HTTP 클라이언트에 알리려는 경우 유용합니다.

예제: 서버가 로컬 IP 127.0.0.1에서 실행 중이고 2개의 포트에서 수신 대기 중인 경우: 80(비암호화) 및 443(암호화). 다음 HTTP 헤더를 사용하여 포트 443에서 HTTP/2가 지원됨을 클라이언트에 알릴 수 있습니다

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

HTTP/2가 활성화된 경우, 연결이 HTTP/2 프로토콜에서 실행되고 있지 않으면 이 헤더가 자동으로 추가됩니다.

HTTP2Options.AltSvc 속성을 사용하여 이 기능을 활성화하거나 비활성화할 수 있습니다.

TsgcWebSocketHTTPServer | HTTP/2 서버 스레드

HTTP 1.1과 HTTP 2.0의 차이점은 아래를 참조하십시오.

HTTP 1.1

기존 HTTP 동작에서는 동일한 연결을 통해 여러 요청을 할 때 클라이언트가 다음 요청을 보내기 전에 각 요청의 응답을 기다려야 합니다. 이러한 순차적 접근 방식은 웹사이트 리소스의 로드 시간을 크게 증가시킵니다. 이 문제를 해결하기 위해 HTTP/1.1은 파이프라이닝이라는 기능을 도입하여 클라이언트가 서버의 응답을 기다리지 않고 여러 요청을 보낼 수 있게 합니다. 이에 따라 서버는 요청을 받은 순서와 동일한 순서로 클라이언트에 응답합니다.

pipelining이 해결책으로 보였지만, 다음과 같은 어려움에 직면했습니다:

- **서버 무시 또는 응답 손상:** 일부 서버는 파이프라인된 요청을 무시하거나 응답을 손상시켜 신뢰할 수 없는 통신을 초래했습니다.
- **Head-of-Line Blocking:** 파이프라인의 첫 번째 요청이 후속 요청을 차단하여, 다른 요청의 처리가 지연될 수 있습니다. head-of-line blocking으로 알려진 이 현상은 페이지 로딩 시간을 느리게 만들었습니다.

HTTP/1.1을 지원하는 서버에서 페이지 로딩을 최적화하기 위해 웹 브라우저는 해결책을 구현했습니다. 서버에 6~8개의 병렬 연결을 열어 여러 요청을 동시에 전송할 수 있게 합니다. 이러한 병렬 처리는 파이프라이닝과 관련된 문제를 완화하고 전체 페이지 로드 시간을 개선하는 것을 목표로 합니다.

웹 브라우저가 6개에서 8개의 병렬 연결을 선택하는 것은 최적화 고려 사항에 근거합니다. 이 숫자를 선택하는 구체적인 이유에는 리소스 활용도, 네트워크 효율성, 잠재적 병목 현상 방지 사이의 절충이 포함될 수 있습니다.

HTTP 2.0

파이프라이닝에서 직면한 제약에 대응하여, HTTP/2는 multiplexing이라는 기능을 도입했습니다. **Multiplexing**은 단일 연결을 통해 여러 요청과 응답의 동시 전송을 가능하게 함으로써 클라이언트와 서버 간의 더 효율적인 통신을 허용합니다.

HTTP/2는 바이너리 프레이밍 메커니즘을 활용하며, 이는 HTTP 메시지가 프레임이라고 하는 더 작은 독립적인 단위로 분해됨을 의미합니다. 이러한 프레임은 인터리브되어 서로 독립적으로 연결을 통해 전송될 수 있습니다. 수신 측에서는 프레임이 다시 조립되어 원래 HTTP 메시지를 재구성합니다.

이 바이너리 프레이밍 메커니즘은 HTTP/2에서 다중화를 달성하는 데 기본이 됩니다. 이를 통해 브라우저는 차단 문제를 겪지 않고 동일한 연결을 통해 여러 요청을 보낼 수 있습니다. 결과적으로 Chrome과 같은 브라우저는 HTTP/2 요청에 동일한 연결 ID를 활용하여 클라이언트와 서버 간의 효율적이고 중단 없는 통신을 가능하게 합니다.

본질적으로, 바이너리 프레이밍 메커니즘으로 활성화된 HTTP/2의 멀티플렉싱 기능은 단일 연결을 통해 여러 요청과 응답의 동시 전송을 용이하게 하여 클라이언트와 서버 간의 데이터 교환의 효율성과 속도를 향상시킵니다.

TsgcWebSocketHTTPServer

HTTP/2 프로토콜의 성능을 향상시키기 위해, 서버가 새 HTTP/2 요청을 수신할 때마다 요청이 기본적으로 스레드 풀(기본값 32)에서 디스패치됩니다. 이는 단일 연결이 이 스레드 풀이 없으면 (연결 스레드 컨텍스트에서) 순차 처리를 요구하는 많은 동시 요청을 보낼 때 발생하는 대기를 방지합니다.

스레드 풀의 동작은 다음 속성으로 구성할 수 있습니다.

- **HTTP2Options.PoolOfThreads.Enabled:** (기본값 false) HTTP/2 요청을 연결 스레드 대신 스레드 풀에서 디스패치하려면 이것을 활성화하십시오.

- **HTTP2Options.Threads:** (기본값 32) HTTP/2 요청을 처리하는 데 사용되는 스레드 수입입니다. 서버의 프로세서 수에 따라 숫자를 설정하십시오.

요청을 미세 조정하여 어떤 요청을 스레드 풀에서 처리해야 하는지(시간이 많이 소요되기 때문에) 선택하고 다른 요청은 연결 스레드에서 처리할 수 있도록 하려면 **OnHttp2BeforeAsyncRequest** 이벤트를 사용할 수 있습니다. 이 이벤트는 스레드 풀에 요청을 큐에 넣기 전에 발생합니다. 요청을 스레드 처리할지 여부를 설정하려면 **Async** 매개변수를 사용하십시오.

```
void __fastcall TForm1::OnHTTP2BeforeAsyncRequest(TObject *Sender, TsgcWConnection *Connection, const TIdHTTPRec
{
    if (ARequestInfo.Document == "/fast-request")
        Async = false;
}
```


TsgcWebSocketHTTPServer | 응답 본문이 없는 404 오류

기본적으로 Indy 라이브러리는 `ContentText`나 `ContentStream`이 할당되지 않은 경우 HTTP 응답에 콘텐츠 본문을 추가합니다. 빈 응답 본문을 반환하려는 경우(404 오류 등), 다음 접근 방식을 사용할 수 있습니다.

콘텐츠 없이 새 `TStringStream`을 생성하고 이를 HTTP Response의 `ContentStream` 속성에 할당하십시오. 이렇게 하면 HTTP 응답이 기본 HTML 태그 없이 전송됩니다.

예제

```
private void OnCommandGet(TIdContext *AContext, TIdHTTPRequestInfo *ARequestInfo,
    TIdHTTPResponseInfo *AResponseInfo)
{
    AResponseInfo->ContentStream = new TStringStream("");
    AResponseInfo->ContentType = "text/html";
    AResponseInfo->ResponseNo = 404;
}
```

TsgcWebSocketHTTPServer | Sessions

HTTP는 무상태 프로토콜입니다(적어도 HTTP 1.1까지). 따라서 클라이언트가 파일을 요청하면 서버가 응답을 보내고 연결이 닫힙니다(연결이 즉시 닫히지 않도록 keep-alive를 활성화할 수 있지만 이는 이 문서의 범위를 벗어납니다). 세션을 사용하면 클라이언트에 대한 정보를 저장할 수 있으며, 예를 들어 클라이언트 로그인 중에 사용할 수 있습니다. 고유한 세션 ID를 사용하고 세션 목록을 검색하여 이미 존재하는지 확인한 다음, 없으면 새 세션을 생성할 수 있습니다. 세션은 비활성 기간 후 또는 클라이언트 로그아웃 후 수동으로 소멸될 수 있습니다.

구성

TsgcWebSocketHTTPServer에는 서버 구성 요소에서 세션을 활성화/비활성화하는 일부 속성이 있습니다. 가장 중요한 것은 다음과 같습니다:

속성	설명
SessionState	이것은 Session을 사용하기 위해 활성화해야 하는 첫 번째 속성입니다. 이 속성을 활성화하지 않으면 동작하지 않습니다
SessionTimeout	여기에 세션이 활성화될 최대 시간에 대해 0보다 큰 값(밀리초)을 설정해야 합니다.
AutoStartSession	세션은 자동으로(AutoStartSession = true) 또는 수동으로(AutoStartSession = false) 생성할 수 있습니다. 세션이 자동으로 생성되면, 서버는 RemoteIP를 고유 식별자로 사용하여 저장된 활성 세션이 있는지 확인합니다.

```
TsgcWebSocketHTTPServer1->SessionState = true;
TsgcWebSocketHTTPServer1->SessionTimeout = 600000;
AutoStartSession = false;
```

세션 생성

새 세션을 생성하려면 **고유한 새 session ID**를 생성해야 합니다. 어떤 값이든 사용할 수 있습니다. **예:** 클라이언트가 인증 중인 경우, user + password + remoteip를 session ID로 사용할 수 있습니다. 그런 다음 세션 목록을 검색하여 이미 존재하는지 확인하십시오. 존재하지 않으면 새로 생성하십시오.

새 세션이 생성되면 **OnSessionStart** 이벤트가 호출되고, 세션이 닫히면 **OnSessionEnd** 이벤트가 발생합니다.

```
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 = TsgcWebSocketHTTPServer1->SessionList->GetSession(vID, ARequestInfo->RemoteIP);
```

```
// create new session if not exists
if (oSession == NULL)
{
    oSession = TsgcWebSocketHTTPServer1->SessionList->CreateSession(ARequestInfo->RemoteIP, VID);
}

AResponseInfo->ContentText = "<html><head></head><body>Authenticated</body></html>";
AResponseInfo->ResponseNo = 200;
}
}
```

TsgcWebSocketHTTPServer | Stream Video

서버를 사용하여 비디오 파일을 스트리밍하려면 IndyStreamFileVideo 함수를 사용하여 chunked transfer encoding으로 파일을 스트리밍할 수 있습니다.

함수는 다음 매개변수를 받습니다:

- **AContext:** OnCommandGet 이벤트에서 가져옵니다.
- **AResponseInfo:** OnCommandGet 이벤트에서 가져옵니다.
- **aFileName:** 스트리밍할 비디오의 전체 파일 이름입니다.
- **ContentType:** 기본값은 "video/mpeg"입니다.
- **aBufferSize:** 기본값은 1024바이트입니다.

Example: 사용자가 문서 /video.mp4로 이동할 때 비디오를 스트리밍하며 이 비디오는 c:\videos\video.mp4 폴더에 있습니다

```
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

Indy 기반 서버 구성 요소(**TsgcWebSocketServer**, **TsgcWebSocketHTTPServer**)는 OpenSSL 대신 **Windows SChannel**(Secure Channel)을 TLS 공급자로 사용할 수 있습니다. SChannel은 네이티브 Windows TLS 구현이므로 외부 DLL이 필요하지 않습니다.

작동 방식

서버 구성 요소에서 SSL이 활성화되어 있고 IOHandler가 **iohSChannel**로 설정된 경우, 서버는 Windows SChannel API를 사용하여 모든 TLS 작업을 처리하는 **TsgcIdServerIOHandlerSSLSChannel** 인스턴스를 생성합니다. 들어오는 모든 클라이언트 연결에 대해 서버는 SChannel 공급자를 통해 TLS 핸드셰이크를 수행하여 프로토콜 버전, 암호화 스위트, 그리고 구성된 인증서 바인딩을 협상합니다.

SChannel은 **Windows Certificate Store** 또는 **PFX 파일**(.pfx / .p12)에서 인증서를 읽습니다. PEM 파일이 필요 없고 OpenSSL 라이브러리를 배포할 필요가 없습니다.

구성

Indy 서버에서 SChannel을 활성화하려면 다음 속성을 구성하십시오:

1. **SSL** 속성을 **True**로 설정하십시오.
2. **SSLOptions.IOHandler**를 **iohSChannel**로 설정하십시오.
3. **SSLOptions.Version**을 원하는 TLS 버전(tls1_2, tls1_3, ...)으로 설정하십시오.
4. **SSLOptions.Port**를 보안 연결에 사용되는 포트로 설정하십시오.
5. 아래에 설명된 두 가지 방법 중 하나를 사용하여 인증서를 구성하십시오.

SChannel_Options 속성

SSLOptions.SChannel_Options 하위 속성은 SChannel 전용 설정을 노출합니다:

속성	Description
CertHash	Windows Certificate Store에 설치된 인증서의 지문(16진수 해시)입니다.
CertStoreName	검색할 인증서 저장소: scsnMY (Personal), scsnRoot , scsnTrust , scsnCA .
CertStorePath	저장소 위치: scspStoreLocalMachine 또는 scspStoreCurrentUser .
CipherList	선택적 콜론으로 구분된 cipher 알고리즘 목록입니다(예: CALG_AES_256:CALG_AES_128). 비어 있으면 시스템 기본값을 사용함을 의미합니다.
UseLegacyCredentials	True인 경우, SCH_CREDENTIALS 대신 레거시 SCHANNEL_CRED 구조체를 사용합니다. 최신 API를 지원하지 않는 이전 Windows 버전의 경우 이를 활성화하십시오.

또한, 일반 **SSLOptions** 속성 **CertFile** 및 **Password** 는 PFX 파일에서 인증서를 로드할 때 사용됩니다.

Windows Store의 인증서

인증서가 이미 Windows Certificate Store에 설치된 경우, 인증서 **thumbprint**를 제공하고 위치를 표시하십시오. 지문을 찾으려면 **PowerShell**을 열고 다음을 실행하십시오:

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

Thumbprint 열은 필요한 16진수 해시를 보여줍니다.

```
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;
```

PFX 파일의 인증서

PFX (.pfx 또는 .p12) 인증서 파일이 있는 경우 SSLOptions에서 **CertFile** 과 **Password** 속성을 설정하십시오. SChannel 은 시작 시 인증서를 가져옵니다.

```
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;
```

PEM 인증서와 개인 키가 있는 경우, 먼저 OpenSSL을 사용하여 이를 PFX 형식으로 변환하십시오.

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

TLS 버전

서버가 허용하는 TLS 버전을 제어하려면 **SSLOptions.Version** 속성을 사용하십시오:

Value	Description
tls1_0	TLS 1.0 (권장되지 않음)
tls1_1	TLS 1.1
tls1_2	TLS 1.2 (권장)
tls1_3	TLS 1.3
tlsUndefined	TLS 1.0, 1.1 및 1.2 수락

Cipher List

기본적으로 SChannel은 시스템 암호 구성을 사용합니다. **SChannel_Options.CipherList**를 콜론으로 구분된 알고리즘 이름 목록으로 설정하여 허용되는 암호를 제한할 수 있습니다. 예:

```
CALG_AES_256:CALG_AES_128
```

Windows 기본값을 사용하려면 이 속성을 비워 두십시오.

Legacy Credentials

Windows Server 2019 이전 버전은 최신 **SCH_CREDENTIALS** API를 지원하지 않을 수 있습니다. 이전 Windows 버전에서 서버 시작에 실패하면 레거시 **SCHANNEL_CRED** 구조를 대신 사용하도록 **SChannel_Options.UseLegacyCredentials**를 **True**로 설정하십시오.

구성 요소는 대부분의 경우 Windows 버전을 자동으로 감지하지만, 필요한 경우 legacy 모드를 강제할 수 있습니다.

OpenSSL 대비 장점

- **외부 DLL 불필요:** SChannel 은 Windows에 내장되어 있으므로 libeay32.dll / ssleay32.dll 또는 libcrypto / libssl을 배포할 필요가 없습니다.
- **Windows Certificate Store 통합:** 운영 체제에서 이미 설치되고 관리되는 인증서를 사용하십시오.
- **자동 업데이트:** TLS 개선 사항 및 보안 패치가 Windows Update를 통해 적용됩니다.

Notes

- SChannel은 **Windows에서만** 사용할 수 있습니다. 크로스 플랫폼 서버의 경우 OpenSSL(iohOpenSSL)을 사용하십시오.
- 서버에는 인증서와 연결된 **개인 키**가 있어야 합니다. Windows Store 방법을 사용할 때 인증서는 개인 키와 함께 가져와야 합니다.
- Certificate Store 방법을 사용하는 프로덕션 서버의 경우, **Local Machine** 저장소 (scspStoreLocalMachine)를 사용하는 것이 권장되므로 서비스를 실행하는 사용자 계정에 관계없이 인증서를 사용할 수 있습니다.
- **PFX** (.pfx / .p12) 인증서 파일만 지원됩니다. PEM 파일이 있는 경우 먼저 PFX 형식으로 변환하십시오.

TsgcWebSocketServer_HTTPAPI

HTTP Server API를 사용하면 애플리케이션이 Microsoft Internet Information Server(IIS)를 사용하지 않고 HTTP를 통해 통신할 수 있습니다.

소개

HTTP Server API는 Microsoft Internet Information Server(IIS) 없이 애플리케이션이 HTTP를 통해 통신할 수 있도록 합니다. 애플리케이션은 특정 URL에 대한 HTTP 요청을 받고, WebSocket 요청을 받고, WebSocket 응답을 보내도록 등록할 수 있습니다. HTTP Server API는 SSL 지원을 포함하므로 애플리케이션이 IIS 없이 보안 HTTP 연결을 통해 데이터를 교환할 수 있습니다. 또한 I/O completion port와 함께 작동하도록 설계되었습니다.

서버는 다음 프로토콜을 지원합니다:

- WebSockets(Windows 8 이상 필요)
- HTTP 1.1
- HTTP/2 (Windows 2016+ 또는 Windows 10+ 필요).

기본적으로 이 구성 요소는 URL 등록을 위해 애플리케이션이 관리자 모드로 실행되어야 합니다. URL이 netsh와 같은 외부 도구를 사용하여 이미 등록된 경우, 관리자 권한 없이 실행할 수 있습니다. 관리자 권한 없이 애플리케이션을 시작할 수 있도록 **BindingOptions.ConfigureSSLCertificate** 속성을 비활성화하십시오. **FastMM4/FastMM5**를 프로젝트의 첫 번째 unit으로 설정하십시오.

이 구성 요소를 구성하려면 아래 단계를 따르십시오:

1. 폼에 TsgcWebSocketServer_HTTPAPI 구성 요소를 놓으십시오
2. 수신 대기 주소와 포트를 정의합니다:

```
Server->Host = "127.0.0.1";
Server->Port = 80;
```

3. 허용되는 Specifications를 설정하십시오. 기본적으로 모든 specification이 허용됩니다.

RFC6455: 표준이며 권장되는 WebSocket 사양입니다.

Hixie76: 이것은 초안이며 Safari 4.2와 같은 오래된 브라우저에 대한 지원을 제공하려는 경우에만 Hixie76 연결을 설정하는 것이 권장됩니다

5. 프로시저를 생성하고 Active = true 속성을 설정합니다.

URL Reservation

HTTP.SYS 서버는 URL 예약을 사용하여 HTTP.SYS 서버가 사용할 URL 엔드포인트를 할당합니다.

Basic URL Reservation

이것은 서버를 구성하는 가장 쉽고 간단한 모드로, 기본적으로 HTTP.SYS 서버가 처리할 Host와 Port만 설정하면 됩니다. 예제: 서버가 IP 127.0.0.1 및 포트 80에서 실행되는 경우, 다음 속성을 설정하기만 하면 됩니다

```
Server->Host = "127.0.0.1";
Server->Port = 80;
```


서버가 둘 이상의 IP에서 실행되고 여러 IP에 바인딩하려면 **NewBinding** 메서드를 사용하십시오. 먼저 Host 및 Bindings 속성을 지운 다음 NewBinding 메서드를 사용하여 모든 Server Binding을 정의하십시오.

```
Server->Host = "";
Server->Bindings->Clear;
Server->Bindings->NewBinding("127.0.0.1", 80, "");
Server->Bindings->NewBinding("80.50.55.11", 80, "");
```

서버가 SSL 연결을 요구하는 경우, SSL 연결을 처리하는 데 사용할 Host와 Port를 정의하려면 다음을 수행하십시오.

```
Server->Host = "127.0.0.1";
Server->Port = 443;
Server->SSL = true;
Server->SSLOptions->Hash = "CERTIFICATE_HASH";
```

서버가 여러 IP Addresses로 SSL 연결을 요구하는 경우, 먼저 Host와 Bindings 속성을 지운 다음 새 Bindings를 등록하십시오.

```
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");
```

참조

- 속성 — 구성 요소에서 사용 가능한 27개의 속성.
- 메서드 — 9개의 public 메서드.
- 이벤트 — 처리할 수 있는 23개의 이벤트.

Guides

- 구성
 - [URL Reservation](#)
- Connection
 - [OnDisconnect not fired](#)
- SSL
 - [HTTPAPI Server SSL](#)
 - 자체 서명 인증서
- HTTP
 - 사용자 정의 헤더
 - 텍스트 응답 전송
 - 파일 응답 전송
 - 큰 파일 게시
- HTTP/2
 - [Disable HTTP/2](#)

HTTPAPI | URL Reservation

HTTP.SYS URL 예약은 Windows 운영 체제의 기능으로, 사용자가 자신의 애플리케이션이나 서비스를 위해 특정 URL(Uniform Resource Locator)을 예약할 수 있게 합니다. HTTP.SYS를 사용하여 URL을 예약하면 운영 체제는 해당 URL에 대한 들어오는 HTTP 요청을 가로채 지정된 애플리케이션이나 서비스로 라우팅합니다.

HTTP.SYS를 사용하여 URL을 예약하려면 애플리케이션 또는 서비스가 먼저 HTTP API를 호출하여 URL을 HTTP.SYS 드라이버에 등록해야 합니다. 애플리케이션 또는 서비스는 URL, HTTP 메서드(예: GET, POST) 및 인증 요구 사항과 같은 추가 설정을 지정합니다.

URL이 등록되면, HTTP.SYS는 해당 URL에 대한 들어오는 HTTP 요청을 가로채고 URL과 메서드를 기준으로 등록된 애플리케이션 또는 서비스를 조회합니다. 일치하는 애플리케이션 또는 서비스가 발견되면, HTTP.SYS 드라이버는 처리를 위해 해당 애플리케이션 또는 서비스로 요청을 전달합니다.

NETSH Commands

URL 등록

이 예제에서는 URL `http://example.com:80/` 이 사용자 `DOMAIN\user`에 대해 등록됩니다. 이를 원하는 URL과 사용자로 대체할 수 있습니다.

```
netsh http add urlacl url=http://example.com:80/ user=DOMAIN\user
```

URL 삭제

이 예제에서는 URL `http://example.com:80/`이 삭제됩니다. 이를 삭제하려는 URL로 대체할 수 있습니다.

```
netsh http delete urlacl url=http://example.com:80/
```

Show All URLs

이 명령은 시스템에 등록된 모든 URL 예약 목록을 표시합니다.

```
netsh http show urlacl
```

TsgcWebSocketServer_HTTPAPI

HTTP.SYS 서버는 시작될 때 URL을 자동으로 등록합니다. 이는 다음 매개변수와 메서드를 사용하여 수행됩니다.

- **Host and Port:** Host가 비어 있지 않고 Port가 0과 다르면 서버는 URL을 등록하려고 시도합니다. 예제: URL `https://127.0.0.1:5000`은 다음 속성을 사용하여 등록됩니다
 - Host = '127.0.0.1';
 - Port = 5000
 - SSL = True
- **NewBinding:** 하나 또는 여러 URL을 등록하려면 이 메서드를 사용하십시오.
 - url `https://127.0.0.1:5000` 등록 --> `NewBinding('127.0.0.1', 5000, '/', True)`
 - url `http://+:5000/ws/` 등록 --> `NewBinding('+', 5000, '/ws/')`

URL 등록은 다음 경우에 관리자 권한이 필요합니다:

- 포트 번호가 1024 미만임
- 호스트는 ip 주소 대신 와일드카드 "+"입니다.

서버의 모든 IP 주소에 대해 포트 443을 등록하고 엔드포인트 "/ws/"에서만 수신 대기하되 서버를 관리자 권한으로 실행하고 싶지 않다면 다음 단계를 수행하십시오:

- netsh를 사용하여 URL 등록
 - netsh http add urlacl url=https://+:443/ws/ user=DOMAIN\user
- 다음 바인딩으로 서버를 구성합니다.
 - NewBinding('+', 443, '/ws/', True);
- ConfigureSSLCertificate 속성을 비활성화하십시오
 - TsgcWebSocketServer_HTTPAPI.BindingOptions.ConfigureSSLCertificate = false;
- SSL 인증서 구성
 - [HTTPAPI Server SSL](#)

HTTPAPI | FineTune

TsgcWSServer_HTTPAPI는 `TsgcServerHTTPAPI_FineTune` 유형의 `FineTune`이라는 이름의 `published` 속성을 노출합니다. 이는 Windows HTTP Server API(`http.sys`)가 요청을 대기열에 넣고, 디스패치하고, 완료하는 방식에 영향을 미치는 저수준 커널 모드 설정을 그룹화합니다.

QueueLength

`HttpServerQueueLengthProperty` 커널 설정을 래핑합니다. 서버가 큐에서 빼기 전에 `http.sys`가 커널 모드 큐에 보유할 대기 중인 요청의 최대 수를 제어합니다. 큐가 가득 차면 커널은 사용자 모드에 도달하지 않고 새 연결에 **503 Service Unavailable**로 직접 응답합니다.

Type: ULONG.

기본값: 1000(Windows 커널 기본값).

Range: 최대 65535개의 요청까지입니다.

개선하는 점: 이 값을 높이면 버스트 워크로드에서 커널이 정당한 트래픽을 거부하는 것을 방지합니다. 네트워크 끊김 후 다시 연결되는 IoT 플릿, 공급자 재설정 후 다시 연결되는 시장 데이터 브로드캐스트 클라이언트, 또는 공유 cron 틱에서 모두 연결되는 예약된 작업 워커 등입니다.

SkipIOCPOnSuccess

`True`로 설정되면 서버는 `FILE_SKIP_COMPLETION_PORT_ON_SUCCESS` 및 `FILE_SKIP_SET_EVENT_ON_HANDLE` 플래그로 요청 큐 핸들에서 `SetFileCompletionNotificationModes`를 호출합니다. 동기적으로 완료되는 요청 큐의 중첩 I/O 작업은 더 이상 I/O 완료 포트에 추가 완료 패킷을 게시하지 않습니다.

Type: Boolean.

Default: `False`(플래그가 설정되지 않아 현재 동작이 유지됨).

개선되는 점: 호출이 동기적으로 `NO_ERROR`를 반환할 때 핫 요청 경로에서 커널-사용자 모드 홉을 제거합니다. 워커는 IOCP 패킷을 기다리는 대신 호출 스레드에서 인라인으로 완료를 디스패치합니다. 이것은 Microsoft의 참조 HTTP Server High Performance 샘플에서 사용되는 패턴입니다. 이 플래그는 `OperatingMode = ompHighPerf`와 함께 사용하도록 의도되었습니다.

API는 런타임에 `kernel32`의 `GetProcAddress`를 통해 확인됩니다. Windows XP 및 이전 버전(Delphi 7 런타임 대상)에서는 진입점이 존재하지 않으므로, 래퍼는 `False`를 반환하고 서버는 플래그가 설정되지 않은 채로 실행됩니다.

OperatingMode

두 가지 `accept/dispatch` 아키텍처 중 하나를 선택합니다.

Type: `TsgcHTTPAPIOperatingMode = (ompClassic, ompHighPerf)`.

기본값: `ompClassic`.

- **ompClassic** - 단일 `acceptor` 스레드가 `HttpReceiveHttpRequest`를 동기적으로 호출하고 `PostQueuedCompletionStatus`를 통해 각 요청을 IOCP 워커 풀로 직접 디스패치합니다. 이것이 기존 동작입니다.
- **ompHighPerf** - Microsoft의 참조 HTTP Server High Performance 패턴을 구현합니다. `acceptor` 스레드가 제거됩니다. 시작 시 서버는 큐에 바인딩된 IOCP에 `ThreadPoolSize x HighPerfAcceptsPerWorker`개의 비동기 수신을 사전 게시합니다(기본값 $32 \times 4 = 128$ 개의 동시 미해결 수신). 각 작업자는 완료를 인라인으로 처리하고 또 다른 비동기 수신을 다시 게시하여 종료될 때까지 슬라이딩 윈도우를 유지합니다.

개선 사항: `ompHighPerf`는 서버가 깊은 단일 스트림 파이프라인(대형 프레임 업로드/다운로드) 또는 많은 동시 클라이언트(수백에서 수천)를 처리할 때 효과를 발휘합니다. 사전 게시 수신 창은 연결당 할당 없이 버스트를 흡수하고, 인라인 디스패치는 `acceptor` 핸드오프 병목 현상을 제거합니다. 저트래픽 API와 개발 환경에서는 기본값 `ompClassic`을 유지하십시오 — 가벼운 워크로드에서는 128개의 사전 게시 컨텍스트를 유지하는 오버헤드가 절약되는 것보다 더 많은 비용이 듭니다. 모드는 서버가 활성화되기 전에만 변경할 수 있습니다.

HighPerfAcceptsPerWorker

OperatingMode = ompHighPerf일 때 각 IOCP 작업자가 미리 게시하는 비동기 수신 수를 제어합니다. 이 값은 ompClassic 모드에서 무시됩니다. 서버가 유지하는 동시 미해결 수신 총 수는 ThreadPoolSize x HighPerfAcceptsPerWorker와 같습니다.

Type: Integer.

기본값: 4.

개선되는 점: 워커당 더 깊은 윈도우를 사용하면 서버가 핫 경로에서 새 컨텍스트를 할당하지 않고도 더 큰 수신 요청 버스트를 흡수할 수 있습니다. 고동시성 배포(IoT 플릿, 시장 데이터 배포, 팬아웃 브로커)에서는 이 값을 높이십시오. 트레이드오프는 메모리입니다. 사전 게시된 각 수신은 완료될 때까지 예약된 요청 버퍼(약 16 KB)를 보유합니다. 기본값 4는 MSDN HP 샘플에 대해 검증된 보수적인 중간값입니다.

예제

다음 스니펫은 고동시성 IoT 백엔드를 위한 HTTP.sys 서버를 구성합니다: 재연결 폭주를 흡수하기 위한 큰 커널 큐, 넓혀진 사전 게시 수신 창이 있는 HighPerf 디스패치, 그리고 인라인 완료 디스패치 활성화.

```
#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;
```

일반적인 내부 또는 저트래픽 API의 경우 모든 FineTune 속성을 기본값으로 두십시오:

```
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 서버 SSL

서버는 **SSL 인증서**를 사용하도록 구성할 수 있습니다. 서버 인증서가 있는 프로덕션 서버를 얻으려면 **잘 알려진 공급자**에서 인증서를 **구매**해야 합니다: Namecheap, GoDaddy, Thawte 등. **테스트 목적으로는 자체 서명 인증서**를 사용할 수 있습니다 (자체 서명 인증서를 사용하는 Demos/Chat 예제를 확인하십시오). 다음 문서 [자체 서명 인증서를 만드는 방법](#)을 읽으십시오.

인증서를 보유하면 연결을 암호화하는 데 사용할 인증서를 지정하도록 서버를 구성해야 합니다.

인증서 해시

먼저 인증서의 해시를 알아야 합니다. 인증서의 해시를 찾는 것은 **powershell**에서 인증서 컨테이너에 대해 **dir** 명령을 실행하는 것만큼 쉽습니다.

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

해시는 16진수 **Thumbprint** 값입니다.

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

Thumbprint 값을 얻으면, **TsgcWebSocketServer_HTTPAPI.TLSOptions.Hash** 속성에 설정하기만 하면 됩니다.

해시를 설정하면 **TsgcWebSocketServer_HTTPAPI.SSL = true**로 설정하기만 하면 서버가 시작할 준비가 됩니다.

netsh를 사용하여 인증서를 수동으로 등록하려면 다음 명령을 사용하십시오:

```
netsh http add sslcert iport=<IP>:<PORT> certhash=<THUMBPRINT> appid="{<GUID>}"
```

<IP>: 바인딩의 로컬 IP 주소를 지정합니다. 와일드카드 바인딩을 사용하지 마십시오. 유효한 IP 주소를 사용하십시오.

<PORT>: 바인딩의 포트를 지정합니다.

<THUMBPRINT>: X.509 인증서 thumbprint.

<GUID>: 정보 목적으로 앱을 나타내기 위해 개발자가 생성한 GUID.

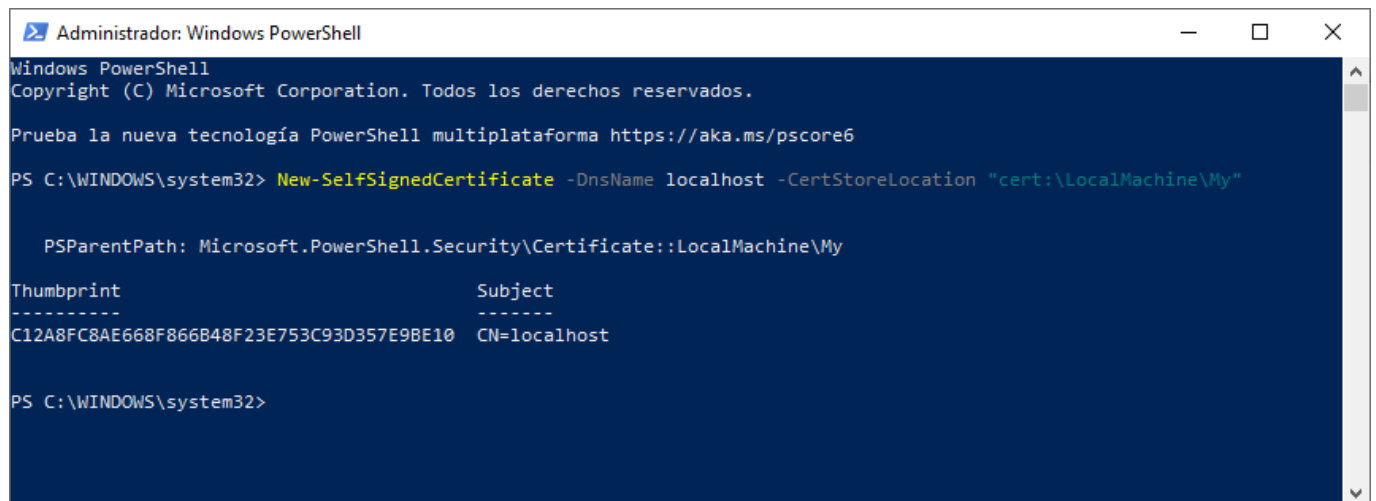
TsgcWebSocketServer_HTTPAPI | Self-Signed Certificates

자체 테스트를 위한 인증서가 필요한 경우, 테스트 머신에서 자체 서명된 인증서를 생성할 수 있습니다. 다음 단계를 따르십시오:

1. Powershell을 관리자 권한으로 실행하십시오.
2. 인증서를 생성하려면 다음 명령을 실행하십시오:

```
New-SelfSignedCertificate -DnsName localhost -CertStoreLocation "cert:\LocalMachine\My"
```

성공하면 새로 생성된 인증서에 대한 확인 메시지를 받게 됩니다. Thumbprint를 복사하여 **TsgcWebSocketServer_HTTPAPI.TLSOptions.Hash** 속성에 붙여넣기만 하면 됩니다.



```

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. 선택 사항, 자체 서명된 인증서를 신뢰할 수 있는 인증 기관으로 추가할 수 있습니다

```
Run MMC -32 as administrator
```

- 3.1. File / Add or Remove Snap-in을 선택하십시오

- 3.2. Certificate를 선택한 다음 Add를 클릭하십시오.

- 3.2. 컴퓨터 계정을 선택하고 다음을 누르십시오.

- 3.4. Local computer를 선택하고 Ok를 누르십시오. 이제 인증서가 표시됩니다.

- 4.5. Personal / Certificates에서 인증서를 선택하고 Trusted Root Certificates Authorities / Certificates에 붙여넣으십시오.

TsgcWebsocketServer_HTTPAPI | HTTP/2 비활성화

HTTP/2 프로토콜은 **Server 2016+** 및 **Windows 10+** OS에서 기본적으로 활성화되어 있습니다. 일부 오래된 브라우저나 HTTP 클라이언트에서는 프로토콜이 완전히 지원되지 않기 때문에 오류가 발생할 수 있습니다. HTTP/2 프로토콜을 비활성화하여 이러한 오류를 방지할 수 있습니다.

HTTP/2 비활성화 방법

- Windows 레지스트리 편집기 열기
- 다음 레지스트리 키로 이동하십시오:

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\HTTP\Parameters

- 다음 DWORD 값을 추가하고 두 값을 모두 0으로 설정하십시오.
 - EnableHttp2Tls
 - EnableHttp2Cleartext
- 컴퓨터를 재부팅합니다.

TsgcWebSocketServer_HTTPAPI | Custom Headers

응답 객체의 **CustomHeaders** 속성을 사용하여 HTTP.SYS 서버의 응답을 사용자 정의할 수 있습니다.

Header Name과 Header Value를 개행 문자로 구분하여 CustomHeaders 값을 설정하십시오.

예제: 다음 헤더를 추가하려면 아래 샘플 코드를 확인하십시오

```
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

HTTP 요청을 처리하려면 **OnHttpRequest** 이벤트를 사용하십시오.

THttpRequest 클래스는 HTTP Request 데이터를 포함합니다.

- **Document:** 피어가 액세스하려는 Document입니다.
- **Method:** HTTP 메서드('GET', 'POST' 등)
- **Headers:** HTTP 요청의 Headers.
- **AcceptEncoding:** accept encoding 변수, 예: "gzip, deflate, br".
- **ContentType:** 예: "text/html"
- **Content:** 존재하는 경우 요청의 콘텐츠입니다.
- **QueryParams:** 쿼리 매개변수입니다.
- **Cookies:** 존재하는 경우 쿠키입니다.
- **ContentLength:** 콘텐츠의 크기입니다.
- **AuthExists, AuthUsername, AuthPassword:** 인증 요청 데이터입니다.
- **Stream:** http 요청에 본문이 있는 경우, 이것은 본문의 스트림입니다.

THttpResponse 클래스는 HTTP 응답 데이터를 포함합니다.

- **ContentText:** 텍스트 형태의 응답입니다.
- **ContentType:** 예: "text/html". ContentText를 UTF8로 인코딩하려면 charset='utf-8'을 설정하십시오. 예: text/html; charset=utf-8
- **CustomHeaders:** 자체 헤더를 보내야 하는 경우 이 변수를 사용하십시오
- **AuthRealm:** 서버가 인증을 요구하면 이 변수를 설정하십시오.
- **ResponseNo:** HTTP 응답 번호입니다. 예: 200은 응답이 성공적임을 의미합니다.
- **ContentStream:** 응답에 스트림이 포함된 경우 여기에 설정하십시오(스트림을 해제하지 마십시오. 자동으로 해제됩니다).
- **FileName:** 응답이 파일 이름인 경우 여기에 파일 이름의 전체 경로를 설정하십시오.
- **Date, Expires, LastModified:** 응답의 datetime 변수입니다.
- **CacheControl:** 캐시 동작을 사용자 지정할 수 있습니다.

예제: 서버가 "/test.html" 문서에 대한 GET 요청을 받으면 OK 응답을 보내고, 그렇지 않으면 다른 문서에 대한 GET 요청인 경우 404를, 다른 method인 경우 오류 500을 보냅니다.

```
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

HTTP 요청에 대한 응답으로 파일을 보내려면 **THttpRequestResponse** 객체의 **FileName** 속성을 사용하십시오.

```
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;
    }
}
```

Resumable Downloads

HTTP 206 Partial Content 응답은 전체 파일을 보내는 대신 서버가 리소스의 특정 부분(범위)에 대한 요청을 이행할 때 사용됩니다. 이는 일반적으로 재개 가능한 다운로드, 미디어 스트리밍, 대용량 파일 전송에 사용됩니다.

작동 방식:

클라이언트가 부분 리소스를 요청함: 클라이언트(브라우저, 다운로더 또는 미디어 플레이어)는 원하는 바이트 범위를 지정하는 Range 헤더를 보냅니다. 요청 예:

```
GET /video.mp4 HTTP/1.1
Host: example.com
Range: bytes=1000-5000
```

이것은 video.mp4의 1000~5000 바이트를 요청합니다.

서버가 HTTP 206으로 응답: 서버가 범위 요청을 지원하면 206 Partial Content로 응답하고 Content-Range 헤더를 포함합니다. 예제 응답:

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 1000-5000/1000000
Content-Length: 4001
Content-Type: video/mp4
```

Content-Range 헤더는 다음을 보여줍니다:

제공되는 범위(1000-5000)
파일의 총 크기(1000000바이트)입니다.
Content-Length 헤더는 반환된 부분의 크기입니다(4001바이트).

Client Can Request More Chunks:

클라이언트는 서로 다른 부분에 대해 여러 요청을 보낼 수 있습니다.
이는 재개 가능한 다운로드와 효율적인 스트리밍을 가능하게 합니다.

```

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

HTTPAPI Server를 처음 사용할 때 클라이언트가 연결을 닫을 때 OnDisconnect 이벤트가 즉시 발생하지 않는 것을 보는 것은 매우 일반적입니다. 그 이유는 HTTPAPI Server가 Indy와 같은 다른 서버와 약간 다르게 작동하기 때문입니다. **Indy 서버**에는 **모든 연결에 대한 스레드**가 있으며 이 스레드는 x 밀리초마다 **연결이 활성 상태인지** 확인합니다. **HTTPAPI Server**는 모든 연결을 처리하는 **스레드 풀**을 사용하며 모든 연결에 대해 활성 상태인지 여부를 **확인하지 않습니다**.

클라이언트가 연결을 닫을 때 알림을 받으려면, 다음 구성을 수행하십시오:

1. [TsgcWebSocketClient](#)를 사용하는 경우, **Options.CleanDisconnect := True**로 설정하십시오. 이는 연결이 닫히기 전에 클라이언트가 연결이 닫힐 것이라는 알림을 서버에 보내려고 시도함을 의미합니다. 서버가 이 메시지를 수신하면 OnDisconnect 이벤트가 호출됩니다.
2. 다른 연결 끊김의 경우 유일한 해결책은 소켓에 무언가를 쓰는 것입니다. 실패하면 연결이 끊어집니다. HTTPAPI 서버에서 **HeartBeat**를 **활성화**하고, 예를 들어 60초의 간격과 0의 시간 초과를 설정하십시오. 이 구성은 60초마다 모든 연결에 ping을 보내고, 연결이 끊어진 연결이 있으면 **OnDisconnect** 이벤트가 발생함을 의미합니다. HeartBeat.Interval에 더 낮은 값을 넣을 수 있지만, 너무 낮게 설정하지 마십시오(예를 들어 1초는 너무 낮음). 서버의 성능이 영향을 받기 때문입니다.

TsgcWebSocketClient_WinHTTP

TsgcWebSocketClient_WinHTTP는 WebSocket 서버에 연결할 수 있는 Client VCL WebSocket 구성 요소를 구현합니다.

소개

TsgcWebSocketClient_WinHTTP는 WebSocket Server에 연결할 수 있는 클라이언트 VCL WebSocket 구성 요소를 구현합니다. WinHTTP API 기반이며 Windows 8 이상이 필요합니다. 이 구성 요소를 구성하려면 아래 단계를 따르십시오:

1. TsgcWebSocketClient_WinHTTP 구성 요소를 폼에 놓습니다
2. 사용 가능한 WebSocket Server에 연결하려면 Host와 Port(기본값 80)를 설정하십시오. URL 속성을 설정할 수 있으며 Host, Port, Parameters... 가 URL에서 업데이트됩니다. 예: wss://127.0.0.1:8080/ws/ 는 다음과 같이 됩니다:

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

3. TLS(보안 연결)를 원하는지 여부를 선택할 수 있으며, 기본적으로 비활성화되어 있습니다.
4. 다음 이벤트를 사용하여 websocket 클라이언트 흐름을 사용자 정의할 수 있습니다:
5. 프로시저를 만들고 Active 속성을 True로 설정하십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 16개의 속성입니다.
- 메서드 — 8개의 public 메서드.
- 이벤트 — 처리할 수 있는 9개의 이벤트.

TsgcWebSocketFirewall

TsgcWebSocketFirewall는 다양한 보안 위협으로부터 WebSocket 서버를 보호하는 포괄적인 방화벽 구성 요소를 구현합니다.

소개

TsgcWebSocketFirewall는 광범위한 보안 위협으로부터 WebSocket 서버를 보호하는 포괄적인 방화벽 구성 요소를 구현합니다. IP 블랙리스트/화이트리스트 필터링, 자동 IP ban이 포함된 무차별 대입 방지, SQL injection 감지, XSS(Cross-Site Scripting) 감지, 연결 속도 제한, 메시지 플러드 방지, GeoIP 국가 기반 필터링, IP 기반 봇 감지, 동적 위험 점수 매기기, 경로 순회 감지, 명령 주입 감지, 페이로드 크기 제한, WebSocket 관련 보호(origin 검증, 프레임 크기 제한, 서브프로토콜 필터링), 점진적 ban 단계 확대, 사용자 정의 규칙 엔진 및 실시간 통계를 포함한 16개의 보호 모듈을 제공합니다.

방화벽은 TsgcWebSocketServer, TsgcWebSocketHTTPServer 및 TsgcWebSocketServer_HTTPAPI에서 사용할 수 있는 **Firewall** 속성을 통해 서버 구성 요소와 자동으로 통합됩니다. 일단 할당되고 구성되면, 수동 이벤트 연결 없이 연결과 메시지를 가로챍니다.

빠른 시작

1. 폼에 **TsgcWebSocketFirewall** 구성 요소를 배치합니다.

2. 원하는 보호 모듈을 구성하십시오:

Blacklist: Enabled를 True로 설정하고 IPs 목록에 IP 또는 CIDR 범위를 추가하십시오(예: "10.0.0.0/8", "192.168.1.100").

Whitelist: Enabled를 True로 설정하고 신뢰할 수 있는 IP를 추가하십시오. 화이트리스트에 등록된 IP는 다른 모든 검사를 우회합니다.

BruteForce: Enabled를 True로 설정하십시오. MaxAttempts(기본값 5), TimeWindowSec(기본값 60) 및 BanDurationSec(기본값 300)를 구성하십시오.

SQLInjection: Enabled를 True로 설정하십시오. 메시지에서 일반적인 SQL 주입 패턴을 감지합니다.

XSS: Enabled를 True로 설정하십시오. 메시지에서 크로스 사이트 스크립팅 패턴을 감지합니다.

RateLimit: Enabled를 True로 설정하십시오. IP당 동시 연결을 제한합니다(기본값 60초당 10개).

FloodProtection: Enabled를 True로 설정하십시오. IP당 초당 메시지를 제한합니다(기본값 100).

PayloadLimit: Enabled를 True로 설정하십시오. MaxSizeBytes(기본값 1MB)를 초과하는 메시지를 차단합니다.

PathTraversal: Enabled를 True로 설정하십시오. 디렉터리 traversal 패턴을 감지합니다.

CommandInjection: Enabled를 True로 설정하십시오. shell 명령 인젝션 패턴을 감지합니다.

ThreatScore: Enabled를 True로 설정하십시오. 위반에 따라 IP에 동적 점수를 할당합니다.

BanEscalation: Enabled를 True로 설정하십시오. 반복 위반자에 대한 차단 기간을 단계적으로 늘립니다.

GeoIP: Enabled를 True로 설정하십시오. 국가 코드별로 연결을 차단하거나 허용합니다.

BotDetection: Enabled를 True로 설정하십시오. 연결하는 IP를 검증된 크롤러, 데이터센터 호스트, 의심되는 봇, 차단 목록 또는 사람으로 분류합니다.

WebSocketProtection: Enabled를 True로 설정하십시오. origin, 프레임 크기 및 서브프로토콜을 검증합니다.

CustomRules: Enabled를 True로 설정하십시오. 복잡한 필터링을 위한 구성 가능한 규칙 엔진입니다.

3. 방화벽을 서버 구성 요소에 할당하십시오:

```
sgcWebSocketHTTPServer1.Firewall := sgcWebSocketFirewall1;
```

4. 서버를 시작합니다. 방화벽은 모든 연결과 메시지를 자동으로 보호합니다.

Properties

Property	설명
Enabled	방화벽을 활성화하거나 비활성화합니다. 기본값: True.
Blacklist	IP 블랙리스트 구성입니다. Enabled(Boolean) 및 IPs(TStringList) 속성을 포함합니다. 정확한 IP와 CIDR 표기법(예: "192.168.0.0/16")을 지원합니다.
Whitelist	IP whitelist 구성입니다. Whitelist에 등록된 IP는 모든 보안 검사를 우회합니다. Blacklist와 동일한 구조입니다.
BruteForce	Brute force 보호입니다. 속성: Enabled, MaxAttempts(기본값 5), TimeWindowSec(기본값 60), BanDurationSec(기본값 300). 시도 제한을 초과하는 IP를 자동으로 차단합니다.
SQLInjection	SQL injection 감지입니다. 속성: Enabled, Action (faDeny/faAllow/faLog), CustomPatterns (추가 패턴을 위한 TStringList).
XSS	교차 사이트 스크립팅(Cross-site scripting) 탐지입니다. 속성: Enabled, Action (faDeny/faAllow/faLog).
RateLimit	연결 속도 제한입니다. 속성: Enabled, MaxConnectionsPerIP(기본값 10), TimeWindowSec(기본값 60).
FloodProtection	메시지 flood 보호. 속성: Enabled, MaxMessagesPerSec(기본값 100), Action(faDeny/faAllow/faLog).
PayloadLimit	메시지 크기 제한. 속성: Enabled, MaxSizeBytes(기본값 1048576), Action(faDeny/faAllow/faLog).
PathTraversal	경로 순회(Path traversal) 탐지입니다. 속성: Enabled, Action (faDeny/faAllow/faLog).
CommandInjection	명령 인젝션 감지입니다. 속성: Enabled, Action(faDeny/faAllow/faLog), CustomPatterns(추가 패턴용 TStringList).
ThreatScore	동적 위협 점수 산정입니다. 속성: Enabled, AutoBanThreshold(기본값 80), DecayPerHour(기본값 5), 그리고 점수 계산을 미세 조정하기 위한 위반 유형별 가중치입니다.
BanEscalation	점진적 차단 단계화입니다. 속성: Enabled, Levels(초 단위 기간의 TStringList), ResetAfterSec(기본값 86400). 후속 차단은 각각 다음 수준 기간을 사용합니다. 수준 값 0은 영구 차단을 의미합니다.

GeolIP	지리적 IP 필터링입니다. 속성: Enabled, Mode(gmBlockList/gmAllowList), Countries(ISO 3166-1 alpha-2 국가 코드의 TStringList), DatabaseFile(GeolIP CSV 데이터베이스 경로).
BotDetection	IP 기반 봇 탐지입니다. 속성: Enabled, KnownBotsFile, CacheDurationSec(기본값 3600), DNSTimeoutMS(기본값 2000) 및 ReverseDNS, Datacenter, DNSBL 하위 객체입니다. 분류 전용: 차단 없이 IP를 검증된 크롤러, 데이터센터 호스트, 의심되는 봇, 차단 목록에 있는 것 또는 사람으로 분류합니다.
WebSocketProtection	WebSocket 전용 보호 기능입니다. 속성: Enabled, AllowedOrigins(TStringList), MaxFrameSize(Integer), AllowedSubprotocols(TStringList).
CustomRules	사용자 정의 규칙 엔진. 속성: Enabled, Rules(TsgcFirewallRuleItem의 TCollection). 각 규칙은 복잡한 필터링 로직을 위한 조건과 액션을 정의합니다.

Events

OnFiltered: 연결 또는 메시지가 차단될 때 발생합니다. *Allow* 매개변수(var Boolean)를 사용하면 방화벽 결정을 재정의할 수 있습니다.

OnViolation: 보안 위반이 감지될 때 발생합니다. 로깅 목적으로 IP 주소, 위반 유형 및 세부 정보를 제공합니다.

OnResolveCountry: GeolIP가 IP를 국가 코드로 확인해야 할 때 발생합니다. 사용자 정의 국가 확인을 제공하려면 핸들러를 할당하거나, 자동 조회를 위해 GeolIP 데이터베이스 파일을 로드하십시오.

OnBotDetected: bot 감지가 IP를 분류할 때 발생합니다. 로깅 또는 분석을 위해 IP 주소, TsgcBotClassification 판정 및 일치하는 bot 이름을 제공합니다.

OnResolveBot: 봇 탐지가 IP를 분류해야 할 때 발생합니다. 내장 리졸버 대신 사용자 정의 판정을 제공하려면 핸들러를 할당하십시오.

OnThreatScoreChanged: IP의 위협 점수가 변경될 때 발생합니다. 이전 및 새 점수 값과 함께 IP 주소를 제공합니다.

Public Methods

Method	설명
IsConnectionAllowed(IP)	주어진 IP의 연결이 허용되는지 확인합니다. 서버와 통합될 때 자동으로 호출됩니다.
IsMessageAllowed(IP, Message)	지정된 IP의 메시지가 모든 메시지 필터를 통과하는지 확인합니다. 서버와 통합될 때 자동으로 호출됩니다.
RegisterConnection(IP)	rate limiting 추적을 위해 새 연결을 등록합니다. 자동으로 호출됩니다.
UnregisterConnection(IP)	추적에서 연결을 제거합니다. 연결 끊김 시 자동으로 호출됩니다.
RegisterFailedAttempt(IP)	무차별 대입 공격 탐지를 위해 실패한 인증 시도를 기록합니다. 인증이 실패할 때 OnAuthentication 이벤트 핸들러에서 이를 호출하십시오.

BanIP(IP, DurationSec)	IP 주소를 수동으로 차단합니다. DurationSec = 0은 영구 차단을 의미합니다.
UnbanIP(IP)	지정된 IP 주소에서 차단을 제거합니다.
IsBanned(IP)	지정된 IP가 현재 차단된 경우 True를 반환합니다.
ClearBans	모든 활성 ban을 제거합니다.
ClearTracking	모든 내부 추적 데이터(연결 카운트, 시도 로그, 메시지 카운트)를 재설정합니다.
SaveBansToFile(FileName)	재시작 전반에 걸친 연속성을 위해 모든 활성 ban을 파일에 저장합니다.
LoadBansFromFile(FileName)	이전에 저장된 파일에서 차단을 복원합니다.
LoadGeoIPDatabase(FileName)	GeoIP CSV 데이터베이스를 로드합니다. 예상 형식: start_ip,end_ip,country_code.
LookupCountry(IP)	IP 주소에 대한 2자리 ISO 국가 코드를 반환합니다.
GetBotClassification(IP)	IP 주소에 대한 TsgcBotClassification을 반환합니다(캐시 우선. 캐시 미스 시 bcUnknown을 반환하고 비동기 조회를 트리거할 수 있음).
GetThreatScore(IP)	IP 주소에 대한 현재 위협 점수(0-100)를 반환합니다.
ResetThreatScore(IP)	IP의 위협 점수를 0으로 재설정합니다.
GetTopThreats(Count)	위협 점수가 가장 높은 IP 목록을 반환합니다.
IsOriginAllowed(Origin)	WebSocket Origin 헤더가 WebSocketProtection 구성에 의해 허용되는지 확인합니다.
IsFrameSizeAllowed(Size)	메시지 크기가 WebSocketProtection.MaxFrameSize에 정의된 한도 내에 있는지 확인합니다.
IsSubprotocolAllowed(Subprotocol)	WebSocket 서브프로토콜이 WebSocketProtection 구성에 의해 허용되는지 확인합니다.
Stats	차단된 연결, 필터링된 메시지, 발급된 차단, 위협 점수 및 모듈별 hit 카운트에 대한 실시간 카운터가 있는 public 읽기 전용 TsgcFirewallStats 객체입니다.

자동 통합

서버의 Firewall 속성에 할당되면, 구성 요소는 자동으로 다음을 수행합니다:

- **새 연결 시:** 블랙리스트, 화이트리스트, 차단, 속도 제한, GeoIP 국가를 확인하고 사용자 정의 규칙을 평가합니다. 차단된 경우 OnTCPConnect 이벤트 전에 연결을 거부합니다.
- **메시지 수신 시:** SQL injection, XSS 패턴, 페이로드 크기, path traversal, command injection, flood 제한을 확인하고 사용자 지정 규칙을 평가합니다. 위협 점수를 업데이트합니다. 위반이 감지되면 클라이언트의 연결을 끊습니다.
- **연결 해제 시:** 내부 추적에서 연결 등록을 해제합니다.

CIDR 지원

blacklist와 whitelist는 IP 범위에 대한 CIDR 표기법을 지원합니다:

- **192.168.1.0/24** — 192.168.1.0부터 192.168.1.255까지 일치
- **10.0.0.0/8** — 모든 10.x.x.x 주소와 일치합니다
- **172.16.0.0/16** — 172.16.x.x 주소와 일치합니다

예제

```
// 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

내장 SQL injection 탐지기는 다음 패턴을 확인합니다(대소문자 구분 안 함):

- ' OR ', ' AND ' — Boolean 인젝션
- UNION SELECT — Union 기반 injection
- '; DROP, '; DELETE, '; INSERT, '; UPDATE — 문 주입(Statement injection)
- -- (SQL 주석)
- EXEC(, EXECUTE(— 명령 실행
- xp_cmdshell — 확장 저장 프로시저
- CAST(, CONVERT(— 유형 변환 남용
- 1=1, 1='1 — 항진 명제 인젝션

추가 사용자 정의 패턴은 SQLInjection.CustomPatterns 속성을 통해 추가할 수 있습니다.

XSS Patterns Detected

- <script — Script injection
- javascript: — 프로토콜 기반 XSS
- onerror=, onload=, onclick=, onmouseover= — 이벤트 핸들러 주입
- eval(— JavaScript 평가
- document.cookie — 쿠키 도용
- <iframe, <object, <embed — 요소 주입(injection)
- <svg onload — SVG 기반 XSS
- expression(— CSS expression 주입

경로 순회 패턴 감지됨

내장 path traversal 감지기는 다음 패턴을 확인합니다:

- ../ 및 ..\ — 상대 디렉터리 순회
- %2e%2e%2f, %2e%2e/, ..%2f — URL-encoded variants
- %00 — Null byte 주입
- /etc/passwd — Unix 민감 파일 액세스
- c:\windows — Windows 시스템 디렉터리 액세스

명령 주입 패턴 감지됨

내장 command injection 감지기는 다음 패턴을 확인합니다:

- ; | && || — Shell 연산자(명령 체이닝)
- `command` — Backtick execution
- \$(command) — Subshell 실행
- rm -rf — 파괴적인 파일 작업
- wget, curl — 원격 파일 다운로드
- /bin/sh, /bin/bash — Unix 셸 호출
- cmd.exe — Windows 명령 인터프리터
- powershell — PowerShell 호출

CommandInjection.CustomPatterns 속성을 통해 추가 사용자 지정 패턴을 추가할 수 있습니다.

스레드 안전성

방화벽 구성 요소는 완전히 스레드 안전합니다. 모든 public 메서드는 추적 데이터에 대한 동시 액세스를 보호하기 위해 내부 임계 구역을 사용합니다. 100,000개의 작업을 수행하는 20개의 동시 스레드로 0개의 오류와 0개의 메모리 누수로 스트레스 테스트되었습니다.

구성 요소는 여러 서버 인스턴스 간에 안전하게 공유할 수 있으며 외부 동기화 없이 모든 스레드(서버 이벤트 핸들러, timer 스레드 등)에서 액세스할 수 있습니다.

참조

- 속성 — 구성 요소에서 사용할 수 있는 16개의 속성.
- 메서드 — 21개의 public 메서드.
- 이벤트 — 처리할 수 있는 6개의 이벤트.

Firewall: 블랙리스트 및 화이트리스트

TsgcWebSocketFirewall는 두 가지 상호 보완적인 메커니즘을 통해 IP 기반 접근 제어를 제공합니다. 특정 IP를 차단하는 블랙리스트와 신뢰할 수 있는 IP에 무조건적인 접근을 허용하는 화이트리스트입니다.

Blacklist

블랙리스트는 지정된 IP 주소 또는 IP 범위로부터의 연결을 방지합니다. 활성화되면 블랙리스트에 있는 IP로부터의 모든 수신 연결은 서버의 연결 이벤트에 도달하기 전에 거부됩니다.

Property	설명
Blacklist.Enabled	블랙리스트 검사를 활성화하거나 비활성화합니다. 기본값: False.
Blacklist.IPs	차단된 IP 주소 또는 CIDR 범위를 포함하는 TStringList입니다.

Adding IPs at Design Time

Object Inspector에서 IPs 속성을 클릭하여 String List 편집기를 엽니다. 한 줄에 하나의 IP 또는 CIDR 범위를 추가하십시오:

```
192.168.1.100
10.0.0.0/8
172.16.0.0/12
```

런타임에 IP 추가

```
sgcWebSocketFirewall1->Blacklist->Enabled = true;
sgcWebSocketFirewall1->Blacklist->IPs->Add("192.168.1.100");
sgcWebSocketFirewall1->Blacklist->IPs->Add("10.0.0.0/8");
```

Whitelist

화이트리스트는 지정된 IP 주소에 무조건적인 액세스를 부여합니다. 화이트리스트에 등록된 IP는 블랙리스트, 무차별 대입 차단, 속도 제한 및 메시지 필터링을 포함한 다른 모든 방화벽 검사를 우회합니다.

Property	설명
Whitelist.Enabled	화이트리스트 검사를 활성화하거나 비활성화합니다. 기본값: False.
Whitelist.IPs	신뢰할 수 있는 IP 주소 또는 CIDR 범위를 포함하는 TStringList입니다.

예제

```
// 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");
```

Priority Order

블랙리스트와 화이트리스트가 모두 활성화된 경우 방화벽은 다음 순서로 평가합니다:

1. IP가 화이트리스트에 있으면 연결이 **즉시 허용**됩니다. 추가 검사가 수행되지 않습니다.
 2. IP가 블랙리스트에 있으면 연결이 **거부**됩니다.
 3. IP가 어느 목록에도 없으면 연결은 다른 검사(무차별 대입, 속도 제한 등)로 진행됩니다.
- 이는 동일한 IP에 대해 화이트리스트 항목이 항상 블랙리스트 항목보다 우선함을 의미합니다.

CIDR 표기법

블랙리스트와 화이트리스트는 모두 IP 범위를 지정하기 위한 CIDR(Classless Inter-Domain Routing) 표기법을 지원합니다.

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

동일한 목록에서 정확한 IP와 CIDR 범위를 혼합할 수 있습니다:

```
sgcWebSocketFirewall1->Blacklist->IPs->Add("203.0.113.50"); // single IP
sgcWebSocketFirewall1->Blacklist->IPs->Add("198.51.100.0/24"); // entire subnet
```

블랙리스트와 화이트리스트 결합

일반적인 패턴은 넓은 범위를 차단하되 해당 범위 내의 특정 IP를 허용하는 것입니다:

```
// 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");
```

이 예제에서는 10.x.x.x 범위의 모든 IP가 차단되지만, 화이트리스트에 등록되어 모든 검사를 우회하는 10.1.1.50은 제외됩니다.

Firewall: Brute Force Protection

무차별 대입 보호 모듈은 동일한 IP 주소에서 반복되는 인증 실패 시도를 감지하고 위반하는 IP를 구성 가능한 기간 동안 자동으로 차단합니다.

작동 방식

방화벽은 슬라이딩 시간 창 내에서 IP 주소별로 실패한 시도를 추적합니다. 실패 횟수가 구성된 임계값을 초과하면 IP가 자동으로 차단됩니다. 차단된 IP는 서버 이벤트가 발생하기 전에 연결 수준에서 거부됩니다.

1. 클라이언트가 연결하고 인증을 시도합니다.
2. 인증이 실패하면, 코드가 **RegisterFailedAttempt(IP)**를 호출하여 방화벽에 알립니다.
3. 방화벽은 현재 시간 창 내에서 해당 IP에 대한 실패 카운터를 증가시킵니다.
4. 카운터가 **MaxAttempts**에 도달하면 IP는 **BanDurationSec**초 동안 차단됩니다.
5. 금지된 IP로부터의 모든 새 연결은 즉시 거부됩니다.
6. ban 기간이 만료된 후 IP는 자동으로 ban 해제되고 실패 카운터가 재설정됩니다.

Properties

Property	Default	설명
BruteForce.Enabled	False	무차별 대입(brute force) 보호를 활성화하거나 비활성화합니다.
BruteForce.MaxAttempts	5	IP가 차단되기 전에 시간 윈도우 내에서 허용되는 최대 실패 시도 횟수입니다.
BruteForce.TimeWindowSec	60	sliding time window의 지속 시간(초)입니다. 이보다 오래된 실패한 시도는 삭제됩니다.
BruteForce.BanDurationSec	300	자동 차단 기간(초). 영구 차단(수동 제거 또는 서버 재시작 시까지)을 위해 0으로 설정하십시오.

RegisterFailedAttempt

방화벽은 어떤 연결이 실패한 로그인을 나타내는 지 알지 못합니다. 로그인 시도가 실패할 때마다 인증 이벤트 핸들러에서 **RegisterFailedAttempt**를 호출해야 합니다:

```
void __fastcall TForm1::ServerAuthentication(TsgcWSConnection *Connection,
String aUser, String aPassword, bool &Authenticated)
{
    Authenticated = CheckCredentials(aUser, aPassword);
    if (!Authenticated)
        sgcWebSocketFirewall1->RegisterFailedAttempt(Connection->IP);
}
```

수동 차단 관리

자동 차단 외에도 차단을 수동으로 관리할 수 있습니다:

Method	설명
BanIP(IP, DurationSec)	IP를 수동으로 ban합니다. 영구 ban의 경우 DurationSec = 0을 사용하십시오.

UnbanIP(IP)	지정된 IP에서 차단을 제거합니다.
IsBanned(IP)	IP가 현재 ban된 경우 True를 반환합니다.
ClearBans	모든 활성 차단(자동 및 수동 모두)을 제거합니다.

예제: 수동 차단

```
// 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");
```

구성 예제

```
// 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;
```

Notes

- 화이트리스트에 등록된 IP는 RegisterFailedAttempt가 호출되더라도 절대 차단되지 않습니다.
- Ban은 기본적으로 메모리에 저장됩니다. 서버 재시작 시에도 ban을 유지하려면 **SaveBansToFile**과 **LoadBansFromFile**을 사용하십시오.
- 시간 윈도우는 고정이지 아니라 슬라이딩입니다. 각 실패한 시도는 개별적으로 추적되며 TimeWindowSec초 후에 만료됩니다.

Firewall: SQL Injection and XSS Detection

TsgcWebSocketFirewall은 WebSocket 메시지에서 SQL injection 및 Cross-Site Scripting(XSS) 공격에 대한 내장 패턴 기반 탐지를 포함합니다. 이 모듈들은 수신 메시지 콘텐츠를 스캔하고 위협 패턴이 감지될 때 조치를 취합니다.

SQL Injection Detection

SQL injection 모듈은 들어오는 모든 WebSocket 메시지에서 일반적인 SQL injection 패턴을 검사합니다. 감지는 대소문자를 구분하지 않습니다.

Properties

Property	설명
SQLInjection.Enabled	SQL injection 감지를 활성화하거나 비활성화합니다. 기본값: False.
SQLInjection.Action	패턴이 감지될 때 취할 동작: faDeny (메시지를 차단하고 연결 해제, 기본값), faLog (OnViolation 이벤트를 발생시키지만 메시지를 허용), faAllow (동작 없음, 감지 비활성화).
SQLInjection.CustomPatterns	확인할 추가 정규식 패턴의 TStringList입니다. 이는 내장 패턴에 추가하여 평가됩니다.

내장 패턴

다음 패턴이 기본적으로 확인됩니다(대소문자 구분 안 함):

Pattern	공격 유형
' OR ', ' AND '	Boolean-based injection
UNION SELECT	Union 기반 injection
'; DROP, '; DELETE, '; INSERT, '; UPDATE	Statement injection / data manipulation
-- (double dash)	SQL 주석 주입
EXEC(, EXECUTE(	명령 실행
xp_cmdshell	Extended stored procedure 남용
CAST(, CONVERT(	Type conversion abuse
1=1, 1='1	항진명제 인젝션

Custom Patterns

CustomPatterns 속성을 사용하여 애플리케이션별 패턴을 추가할 수 있습니다:

```
sgcWebSocketFirewall1->SQLInjection->Enabled = true;
sgcWebSocketFirewall1->SQLInjection->CustomPatterns->Add("WAITFOR DELAY");
```

```
sgcWebSocketFirewall1->SQLInjection->CustomPatterns->Add("BENCHMARK\(");
sgcWebSocketFirewall1->SQLInjection->CustomPatterns->Add("SLEEP\(");
```

XSS Detection

XSS 모듈은 WebSocket 메시지에서 Cross-Site Scripting 패턴을 감지하여, 다른 연결된 클라이언트가 렌더링할 수 있는 악성 스크립트의 주입을 방지합니다.

Properties

Property	설명
XSS.Enabled	XSS 감지를 활성화하거나 비활성화합니다. 기본값: False.
XSS.Action	패턴이 감지될 때 취할 동작: faDeny (차단 및 연결 끊기, 기본값), faLog (OnViolation을 발생시키지만 허용), faAllow (동작 없음).

내장 패턴

다음 XSS 패턴이 감지됩니다(대소문자 구분 없음):

Pattern	공격 유형
<script	Script 태그 주입
javascript:	프로토콜 기반 XSS
onerror=, onload=, onclick=, onmouseover=	이벤트 핸들러 주입
eval(	JavaScript code evaluation
document.cookie	쿠키 탈취
<iframe, <object, <embed	요소 주입
<svg onload	SVG-based XSS
expression(	CSS expression injection

Action Property

SQLInjection 및 XSS 모듈 모두 TsgcFirewallAction 유형의 Action 속성을 지원합니다:

Value	Behavior
faDeny	메시지가 차단되고 클라이언트 연결이 끊깁니다. OnViolation 이벤트가 발생합니다. 이것이 기본값입니다.
faLog	메시지는 통과가 허용되지만 로깅 목적으로 OnViolation 이벤트가 발생합니다.
faAllow	아무 작업도 수행되지 않습니다. Enabled 플래그를 True로 유지하면서 사실상 감지를 비활성화합니다.

OnViolation Event

SQL injection 또는 XSS 패턴이 감지되면(Action이 faDeny 또는 faLog로 설정된 경우), 위반에 대한 세부 정보와 함께 OnViolation 이벤트가 발생합니다:

```
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;
    }
}
```

구성 예제

```
// 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;
```

Notes

- 패턴 감지는 모든 들어오는 텍스트 메시지에 적용됩니다. 바이너리 메시지는 스캔되지 않습니다.
- 화이트리스트에 등록된 IP는 메시지 필터링을 완전히 우회합니다.
- 내장 패턴은 가장 일반적인 공격 벡터를 다룹니다. 애플리케이션별 위협의 경우 SQLInjection.CustomPatterns를 통해 사용자 정의 패턴을 추가하십시오.
- 클라이언트에 영향을 주지 않고 감지를 모니터링하려면 초기 배포 중에 faLog 모드를 사용한 다음, 애플리케이션에 대한 패턴을 검증하면 faDeny로 전환하십시오.

Firewall: Rate Limiting and Flood Protection

TsgcWebSocketFirewall는 트래픽 제어를 위한 두 가지 보완적인 메커니즘을 제공합니다: 연결 속도 제한(IP가 열 수 있는 연결 수 제어)과 메시지 플러드 보호(연결된 클라이언트가 초당 보낼 수 있는 메시지 수 제어).

연결 속도 제한

속도 제한은 단일 IP 주소가 시간 윈도우 내에 설정할 수 있는 연결 수를 제한합니다. 이는 단일 클라이언트가 사용 가능한 모든 서버 연결을 소비하는 것을 방지합니다.

Properties

Property	Default	설명
RateLimit.Enabled	False	연결 rate limiting을 활성화하거나 비활성화합니다.
RateLimit.MaxConnectionsPerIP	10	단일 IP 주소에서 허용되는 최대 동시 연결 수.
RateLimit.TimeWindowSec	60	연결을 추적하는 초 단위 시간 창입니다. 이보다 오래된 연결은 카운트에서 제거됩니다.

작동 방식

1. 새 연결이 도착하면 방화벽은 해당 IP의 활성 연결 수를 계산합니다.
2. count가 MaxConnectionsPerIP 이상이면 새 연결이 거부됩니다.
3. 연결이 끊기면 추적 카운트에서 제거됩니다.
4. TimeWindowSec보다 오래된 연결은 추적에서 자동으로 정리됩니다.

예제

```
// Allow maximum 5 connections per IP
sgcWebSocketFirewall1->RateLimit->Enabled = true;
sgcWebSocketFirewall1->RateLimit->MaxConnectionsPerIP = 5;
sgcWebSocketFirewall1->RateLimit->TimeWindowSec = 60;
```

Message Flood Protection

플러드 보호는 단일 IP가 초당 보낼 수 있는 메시지 수를 제한합니다. 이는 남용하는 클라이언트가 빠른 메시지 버스트로 서버를 압도하는 것을 방지합니다.

Properties

Property	Default	설명
FloodProtection.Enabled	False	메시지 플러드 보호를 활성화하거나 비활성화합니다.
FloodProtection.MaxMessagesPerSec	100	단일 IP 주소로부터 초당 허용되는 최대 메시지 수입니다.

FloodProtection.Action	faDeny	한도를 초과할 때의 작업: faDeny (클라이언트 연결 끊기), faLog (OnViolation을 발생시키지만 허용), faAllow (작업 없음).

작동 방식

1. 방화벽은 매초 재설정되는 IP별 메시지 카운터를 유지합니다.
2. 들어오는 각 메시지는 발신자의 IP에 대한 카운터를 증가시킵니다.
3. 카운터가 MaxMessagesPerSec를 초과하면, 구성된 Action이 취해집니다.
4. faDeny 작업으로, 클라이언트가 즉시 연결 해제됩니다. faLog로, OnViolation 이벤트가 발생하지만 메시지는 정상적으로 처리됩니다.

예제

```
// Limit to 50 messages per second per IP
sgcWebSocketFirewall1->FloodProtection->Enabled = true;
sgcWebSocketFirewall1->FloodProtection->MaxMessagesPerSec = 50;
sgcWebSocketFirewall1->FloodProtection->Action = faDeny;
```

두 보호 결합

속도 제한과 플러드 보호는 함께 작동하여 포괄적인 트래픽 제어를 제공합니다:

```
// 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;
```

이 구성을 사용하면 각 IP는 최대 5개의 동시 연결을 열 수 있으며, 이러한 모든 연결에 걸쳐 초당 최대 50개의 메시지를 결합하여 보낼 수 있습니다.

Tuning Guidelines

Scenario	MaxConnectionsPerIP	MaxMessagesPerSec
Chat application	3-5	10-30
실시간 데이터 피드	2-3	100-500
게임 서버	1-2	50-100
API gateway	10-20	200-1000

보수적인 제한으로 시작하고 모니터링을 기반으로 늘리십시오. faDeny를 활성화하기 전에 트래픽 패턴을 관찰하려면 처음에는 faLog 모드를 사용하십시오.

Notes

- 화이트리스트에 등록된 IP는 속도 제한과 flood 보호를 모두 우회합니다.

- 속도 제한 추적 데이터는 연결이 닫힐 때 자동으로 지워집니다. 모든 카운터를 수동으로 재설정하려면 ClearTracking 을 사용하십시오.
- 역방향 프록시 뒤에서는 모든 연결이 프록시 IP에서 오는 것처럼 보일 수 있습니다. 실제 클라이언트 IP(예: X-Forwarded-For 헤더)를 전달하도록 프록시를 구성하고 서버가 이를 올바르게 확인하는지 확인하십시오.

Firewall: Bot Detection

TsgcWebSocketFirewall은 연결하는 IP 주소를 봇, 크롤러, 데이터센터 호스트 또는 사람 방문자로 분류할 수 있습니다. Bot Detection은 알려진 봇 데이터베이스, forward-confirmed reverse DNS 검증, 데이터센터/ASN 범위 일치 및 DNSBL 차단 목록 조회를 결합하여 각 IP에 분류를 할당합니다. 이것은 **분류 전용** 모듈입니다. 자체적으로 연결을 차단하지 않습니다. 애플리케이션이 무엇을 할지 결정할 수 있도록 판정을 보고합니다.

작동 방식

1. IP가 서버에 연결합니다.
2. 방화벽이 결과 캐시에서 IP를 조회합니다. 캐시된 분류가 존재하면(그리고 만료되지 않았으면) 즉시 사용됩니다.
3. 캐시 미스 시, IP는 알려진 봇 파일(검증된 크롤러)과 데이터센터 ASN 파일(클라우드/호스팅 범위)에 대조됩니다.
4. ReverseDNS 또는 DNSBL 검증이 활성화되면 조회가 accept 스테드를 절대 차단하지 않도록 백그라운드 스테드에 예약됩니다. 조회가 완료될 때까지 IP는 **bcUnknown**으로 분류될 수 있습니다.
5. 분류가 해석되면 캐시에 저장되고(CacheDurationSec 초 동안) **OnBotDetected** 이벤트가 발생합니다.
6. 분류는 방화벽 Stats(**fvBot** 위반 카운터)를 업데이트하고 규칙의 **BotType** 속성을 통해 Custom Rules에서 사용할 수 있습니다.

Properties

Property	설명
BotDetection.Enabled	bot 감지를 활성화하거나 비활성화합니다. 기본값: False.
BotDetection.KnownBotsFile	검증된 크롤러의 CSV 파일 경로입니다. 각 라인은 CIDR,botname 입니다 (예: IP 범위와 그것이 속한 크롤러).
BotDetection.CacheDurationSec	확인된 분류가 다시 조회되기 전에 결과 캐시에 보관되는 시간(초)입니다. 기본값: 3600.
BotDetection.DNSTimeoutMS	각 DNS 조회(역방향 DNS 및 DNSBL)에 대한 밀리초 단위의 시간 초과입니다. 기본값: 2000.
BotDetection.ReverseDNS.Enabled	주장된 크롤러의 forward-confirmed reverse DNS(FCrDNS) 검증을 활성화합니다. 기본값: False.
BotDetection.Datacenter.Enabled	datacenter / 클라우드 호스팅 범위 탐지를 활성화합니다. 기본값: False.
BotDetection.Datacenter.ASNFile	데이터센터 ASN 범위의 CSV 파일 경로. 각 줄은 CIDR,asname 입니다 (예: 클라우드 공급자 범위와 그 ASN 이름).
BotDetection.DNSBL.Enabled	DNSBL(DNS 차단 목록) 조회를 활성화합니다. 기본값: False.
BotDetection.DNSBL.Zones	쿼리할 DNSBL zone의 TStringList입니다(예: zen.spamhaus.org).

Classification Values

TsgcBotClassification 열거형은 IP에 할당된 판정을 설명합니다:

Value	설명

bcUnknown	IP가 아직 분류되지 않았습니다. 비동기 조회가 진행 중인 동안 캐시 미스에서 반환됩니다.
bcVerifiedCrawler	known-bots 파일 또는 forward-confirmed reverse DNS에 의해 확인된 알려진 합법적인 크롤러(예: 검색 엔진 봇).
bcDatacenter	IP가 ASN 파일에 나열된 클라우드 또는 호스팅 공급자 범위에 속합니다. 주거용 사용자가 아니라 종종 자동화된 트래픽입니다.
bcSuspectedBot	IP가 봇 특성을 보이지만 검증된 크롤러와 일치시킬 수 없습니다.
bcBlocklisted	IP가 구성된 DNSBL 영역 중 하나에 나열되어 있습니다.
bcHuman	IP가 일반적인 사람 방문자처럼 보입니다(봇, 데이터센터 또는 차단 목록 일치 없음).

CSV File Formats

known-bots 파일(**KnownBotsFile**)은 검증된 크롤러 범위를 나열합니다. 각 줄은 CIDR 범위 다음에 bot 이름이 옵니다:

```
66.249.64.0/19, Googlebot
157.55.0.0/16, Bingbot
17.58.0.0/16, Applebot
```

데이터센터 ASN 파일(**Datacenter.ASNFile**)은 클라우드/호스팅 범위를 나열합니다. 각 줄은 CIDR 범위 뒤에 ASN 이름이 옵니다:

```
3.0.0.0/15, AMAZON-AES
13.64.0.0/11, MICROSOFT-AZURE
34.64.0.0/10, GOOGLE-CLOUD
```

Enabling Bot Detection

모듈을 활성화하고, known-bot 및 ASN 파일을 로드하고, 필요한 검증 하위 모듈을 켜십시오. DNS 조회는 백그라운드 스레드에서 실행되므로 ReverseDNS 또는 DNSBL을 활성화해도 accept 스레드가 느려지지 않습니다.

```
// 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 blacklist lookups
sgcWebsocketFirewall1->BotDetection->DNSBL->Enabled = true;
sgcWebsocketFirewall1->BotDetection->DNSBL->Zones->Add("zen.spamhaus.org");
```

Classify-Only 동작 및 OnBotDetected

Bot Detection은 자체적으로 연결을 거부하지 않습니다. IP가 분류되면 **OnBotDetected** 이벤트를 발생시키고, 방화벽 Stats(**fvBot** 카운터)를 업데이트하며, 결과를 캐시에 저장합니다. 트래픽을 로깅하거나, 분석을 구축하거나, 직접 작성한 로직에 입력하려면 이 이벤트를 사용하십시오:

```
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;
    }
}
```

Custom Rule로 Classification에 조치하기

Bot Detection은 분류만 하므로, 차단 또는 금지는 Custom Rules 엔진을 통해 수행됩니다. 각 **TsgcFirewallRuleItem**은 **BotType** 속성을 노출합니다. 설정되면, 규칙은 분류가 해당 값과 같은 연결과 일치하며, 그런 다음 이를 거부, 금지 또는 기록할 수 있습니다.

```
sgcWebSocketFirewall1->CustomRules->Enabled = true;

// Deny connections coming from blocklisted IPs
TsgcFirewallRuleItem *vRule =
    (TsgcFirewallRuleItem *)sgcWebSocketFirewall1->CustomRules->Rules->Add();
vRule->BotType = bcBlocklisted;
vRule->ActionType = raDeny;
```

OnResolveBot을 사용한 사용자 지정 판정

자체 봇 인텔리전스(서드파티 평판 서비스, 내부 허용/거부 목록 등)가 있는 경우 **OnResolveBot** 이벤트를 할당하십시오. 할당되면 핸들러가 내장 리졸버 대신 판정을 제공합니다. 결과를 반환하려면 *aClassification* 및 *aBotName* var 매개변수를 설정하십시오:

```
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;
}
```

참고: OnResolveBot이 할당되면, 해당 IP에 대해 내장 resolver(known-bots 파일, reverse DNS, datacenter 및 DNSBL 검사)가 우회됩니다. 핸들러가 반환하는 판정은 캐시되어 내장 결과와 똑같이 OnBotDetected 및 Custom Rules로 전달됩니다.

TsgcWSRateLimiter

*TsgcWSRateLimiter*는 서버 엔드포인트를 과도한 트래픽, 남용 및 스크래핑으로부터 보호하는 완전한 기능을 갖춘 속도 제한 구성 요소를 구현합니다.

소개

TsgcWSRateLimiter는 과도한 트래픽, 남용 및 스크래핑으로부터 서버 엔드포인트를 보호하는 완전한 기능의 속도 제한 구성 요소를 구현합니다. 세 가지 알고리즘(Token Bucket, Sliding Window, Fixed Window), 범위가 지정된 규칙(IP별, API 키별, 사용자별, 엔드포인트별), 장기 할당량(시간 / 일 / 월), 버스트 보호, 구성 가능한 HTTP 429 응답 형태 및 실시간 통계를 지원합니다. 내부 상태를 파일이나 스트림에 영속화하여 카운터가 재시작 후에도 유지될 수 있습니다.

rate limiter는 TsgcWebSocketHTTPServer 및 TsgcWSServer_HTTPAPI에서 사용 가능한 **RateLimiter** 속성을 통해 서버 구성 요소와 자동으로 통합됩니다. 할당되면 수동 이벤트 연결 없이 모든 연결과 모든 메시지를 평가합니다.

빠른 시작

1. TsgcWSRateLimiter 구성 요소를 폼에 놓습니다.
2. 하나 이상의 전략 및 scope을 활성화하십시오(아래의 섹션별 문서를 참조하십시오).
3. 속도 제한기를 서버 구성 요소에 할당하십시오:

```
sgcWebSocketHTTPServer1.RateLimiter := sgcWSRateLimiter1;
```

4. 서버를 시작하십시오. rate limiter는 모든 연결과 메시지를 자동으로 평가하고, 초과 트래픽을 HTTP 429로 거부하며, 통계 카운터를 업데이트합니다.

최상위 속성

기능: 구성 요소의 마스터 스위치와 상태 영속화 메서드가 사용하는 선택적 스토리지 파일입니다. Enabled가 False이면 모든 public 메서드가 "허용됨"으로 단축되고 카운터가 업데이트되지 않습니다.

사용 시기: 제어된 부하 테스트 중이나 유지 관리 기간 동안 서버에서 할당을 해제하지 않고 런타임에 속도 제한기를 비활성화 하십시오. 애플리케이션이 다시 시작되더라도 카운터가 유지되도록 StorageFile을 설정하십시오. 예를 들어 롤링 Token Bucket은 처음부터 다시 시작하여 거대한 버스트를 수락하는 대신 중단된 지점부터 계속 다시 채워집니다.

Property	Type	Default	설명
Enabled	Boolean	True	마스터 스위치입니다. False인 경우 IsAllowed / Consume / IsConnectionAllowed / IsMessageAllowed는 항상 True를 반환하고 카운터가 업데이트되지 않습니다.
StorageFile	string	"	재시작 간 카운터 지속성을 위해 SaveStateToFile / LoadStateFromFile에서 사용하는 선택적 파일 경로입니다.

```
sgcWSRateLimiter1->Enabled = true;
sgcWSRateLimiter1->StorageFile = "ratelimit.dat";
sgcWSRateLimiter1->LoadStateFromFile(sgcWSRateLimiter1->StorageFile);
```

TokenBucket Strategy

기능: 키당 최대 *Capacity*개의 토큰 버킷을 유지합니다. 버킷은 *RefillIntervalMs*밀리초마다 *RefillRate*개의 토큰으로 다시 채워집니다. 각 요청은 토큰 하나(또는 Consume에 전달된 Cost 인수)를 소비합니다. 버킷이 비어 있으면 요청이 거부되고 재충전 일정에서 *RetryAfterSec* hint가 계산됩니다.

사용 시기: API는 평균 초당 10개 요청의 속도를 알리지만, 잘 작동하는 클라이언트가 스로틀링되지 않고 최대 50개의 요청을 연속으로 버스트하도록 허용하려고 합니다. *Capacity=50*, *RefillRate=10/1000ms*인 Token Bucket은 클라이언트가 즉시 50개의 요청을 보낸 다음, 버킷이 다시 채워질 때까지 초당 10개 요청으로 낮춥니다.

Property	Type	Default	설명
Enabled	Boolean	False	Token Bucket 알고리즘을 전역 기본 전략으로 활성화합니다.
Capacity	Integer	100	버킷이 보유할 수 있는 최대 토큰 수. 최대 버스트 크기를 제어합니다.
RefillRate	Integer	10	RefillIntervalMs마다 버킷에 추가되는 토큰 수.
RefillIntervalMs	Integer	1000	밀리초 단위의 리필 주기. RefillRate와 함께 지속적인 요청 속도를 결정합니다.

```
// 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

기능: 모든 요청에 대해 타임스탬프를 저장하고 *WindowSec*초의 롤링 윈도우 내에 속하는 타임스탬프를 계산합니다. 카운트가 *MaxRequests*를 초과하면 거부합니다. 윈도우가 연속적으로 슬라이드하므로, 버스트가 빠져나갈 수 있는 강력한 재설정 순간이 없습니다.

사용 시기: "60초 롤링 윈도우에서 100개 이하의 요청"을 보장하는 엄격한 SLA를 게시합니다. Fixed Window는 클라이언트가 12:00:59에 100개의 요청을 발생시키고 12:01:00에 다른 100개를 발생시킬 수 있게 합니다(1초 내에 200개). Sliding Window는 이를 방지하고 SLA를 정확히 적용합니다.

Property	Type	Default	설명
Enabled	Boolean	False	Sliding Window 알고리즘을 전역 기본 전략으로 활성화합니다.
WindowSec	Integer	60	롤링 윈도우의 크기(초)입니다.
MaxRequests	Integer	1000	WindowSec 너비의 롤링 윈도우 내에서 허용되는 최대 요청 수입니다.

```
// 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
```

FixedWindow Strategy

기능: WindowSec초의 고정된 시계 윈도우 내에서 요청을 계산하고 윈도우 경계에서 카운터를 0으로 재설정합니다. 가장 간단한 알고리즘이며 메모리 측면에서 가장 저렴합니다. 그러나 경계 교차 주변의 짧은 기간 동안 최대 2×MaxRequests를 허용할 수 있습니다.

사용 시점: "시간당 1000개 요청, 정시에 카운터 재설정"과 같은 청구 스타일 할당량입니다. 고객은 명확하게 표시된 재설정 시간을 기대하며, 청구 단위가 롤링 분이 아닌 시간이므로 사소한 경계 버스트 동작은 허용됩니다.

Property	Type	Default	설명
Enabled	Boolean	False	Fixed Window 알고리즘을 전역 기본 전략으로 활성화합니다.
WindowSec	Integer	60	고정 윈도우의 크기(초). 윈도우가 경과하면 카운터가 0으로 재설정됩니다.
MaxRequests	Integer	1000	단일 고정 윈도우 내에서 허용되는 최대 요청 수.

```
// Billing-style: 1000 requests per hour, resets on the hour
sgcWSRateLimiter1->FixedWindow->Enabled = true;
sgcWSRateLimiter1->FixedWindow->WindowSec = 3600;
sgcWSRateLimiter1->FixedWindow->MaxRequests = 1000;
```

PerIP

기능: 소스 IP 주소로 범위가 지정된 속도 제한을 정의합니다. 요청 키가 PerAPIKey / PerUser / PerEndpoint와 일치하지 않으면 PerIP가 활성 규칙이 되어 전역 Token/Sliding/Fixed 기본값을 재정의합니다. 각 IP는 자체의 독립적인 카운터를 가집니다.

사용 시기: 클라이언트가 많은 public API. PerIP가 없으면 시끄러운 단일 클라이언트가 전역 예산을 소비하고 다른 모든 클라이언트를 고갈시킬 수 있습니다. 모든 IP에 자체 버킷(예: 분당 100개 요청)을 부여하여 한 클라이언트가 다른 클라이언트에 영향을 주지 못하게 하십시오.

Property	Type	Default	설명
Enabled	Boolean	False	IP별 속도 제한을 활성화합니다.
MaxRequests	Integer	100	윈도우당 최대 요청 수(또는 Strategy가 rlsTokenBucket일 때 버킷 용량).
WindowSec	Integer	60	시간 윈도우 크기(초)입니다. Token Bucket의 경우 리필 비율(초당 MaxRequests / WindowSec 토큰)을 도출하는 데도 사용됩니다.
Strategy	TsgcRateLimitStrategy	rlsTokenBucket	이 범위에 사용되는 알고리즘: rlsTokenBucket, rlsSliding 또는 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

기능: PerIP와 동일한 의미이지만 API 키 식별자로 범위가 지정됩니다. 구성 요소는 apiKey: 접두사로 시작하는 키를 API 키 범위로 인식합니다. 모든 고객이 소스 IP에 관계없이 독립적인 속도 예산을 얻도록 TsgcWSAPIKeyManager와 통합되도록 설계되었습니다.

사용 시기: 계층화된 가격 책정이 있는 SaaS. 모든 고객은 자체 API 키와, 연결하는 위치와 무관한 자체 할당량을 가집니다. PerAPIKey를 기본 계층으로 사용하고, 경로 접두사를 키로 하는 PerEndpoint 규칙을 통해 또는 OnThrottled 이벤트 핸들러 내에서 결정을 재정의하여 고객 계층(Free / Pro / Enterprise)을 차별화하십시오.

Property	Type	Default	설명
Enabled	Boolean	False	API 키별 속도 제한을 활성화합니다.
MaxRequests	Integer	100	윈도우당 최대 요청 수(또는 Token Bucket의 경우 버킷 용량)입니다.
WindowSec	Integer	60	시간 윈도우 크기(초)입니다.
Strategy	TsgcRateLimitStrategy	rlsTokenBucket	이 범위에 사용되는 알고리즘입니다.

```
// 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

기능: PerIP / PerAPIKey와 동일한 구조이지만 애플리케이션 정의 사용자 식별자로 키가 지정됩니다. 구성 요소는 접두사 `user:`로 시작하는 키를 사용자 범위로 인식합니다. 단일 사용자가 여러 장치나 IP에서 연결할 수 있고 해당 사용자에 대해 단일 공유 예산을 원할 때 유용합니다.

사용 시기: 인증된 웹 / 모바일 앱. 사용자 `alice@example.com`은 휴대폰, 노트북 또는 직장 데스크톱 중 어디에서 호출하든 관계 없이 하루에 1000개의 요청이 허용됩니다. IP별로 집계하면 이중 계산되고, 사용자별로 집계하면 실제로 청구하려는 예산을 얻습니다.

Property	Type	Default	설명
Enabled	Boolean	False	사용자별 속도 제한을 활성화합니다.
MaxRequests	Integer	100	윈도우당 최대 요청 수(또는 Token Bucket의 경우 버킷 용량)입니다.
WindowSec	Integer	60	시간 윈도우 크기(초)입니다.
Strategy	TsgcRateLimitStrategy	rlsTokenBucket	이 범위에 사용되는 알고리즘입니다.

```
// 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

기능: 패턴 기반 규칙의 모음. 각 항목은 자체 *Pattern*(와일드카드 * 및 ? 지원), *Strategy*, *MaxRequests* 및 *WindowSec*를 가집니다. 패턴이 키와 일치하는 첫 번째 활성화된 항목이 우선합니다 — 추가 항목은 평가되지 않습니다. PerAPIKey, PerUser 및 PerIP보다 우선합니다.

사용 시기: 클라이언트당 10 req/min으로 제한해야 하는 비용이 많이 드는 엔드포인트 `/api/v1/expense-report`가 있는 반면, 저렴한 상태 엔드포인트 `/api/v1/status`는 1000 req/min을 허용합니다. 엔드포인트마다 예산이 다릅니다. 경로 패턴당 하나의 PerEndpoint 항목을 구성하십시오.

Property	Type	Default	설명
Enabled	Boolean	False	PerEndpoint 규칙 집합을 활성화합니다. False이면 어떤 항목도 평가되지 않습니다.

규칙	TCollection	empty	TsgcRateLimitRuleItem의 컬렉션입니다. 순서가 중요합니다 — 첫 번째 일치가 우선합니다.

각 **TsgcRateLimitRuleItem**은 다음을 노출합니다:

Property	Type	Default	설명
Name	string	"	객체 인스펙터에 표시되는 친숙한 이름입니다.
Enabled	Boolean	True	이 항목을 삭제하지 않고 비활성화합니다.
패턴	string	"	키에 대해 일치하는 와일드카드 패턴(대소문자 구분 안 함)입니다. * 및 ? 를 지원합니다.
MaxRequests	Integer	100	윈도우당 최대 요청 수 / 버킷 용량입니다.
WindowSec	Integer	60	시간 윈도우 크기(초)입니다.
Strategy	TsgcRateLimitStrategy	rlsTokenBucket	일치하는 요청에 사용할 알고리즘입니다.

```
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;
```

할당량

기능: 메인 전략의 롤링 / 리필 윈도우를 넘어 지속되는 장기 제한입니다. 각 항목은 키(또는 전체 시스템)가 달력 기간(*qpHour*, *qpDay* 또는 *qpMonth*) 내에 할 수 있는 요청 수를 제한합니다. *Limit*에 도달하면 요청이 거부되고 *OnQuotaExceeded* 이벤트가 발생합니다.

사용 시기: 월 10,000개 요청 하드 캡으로 제한된 Free-tier 플랜입니다. rolling token bucket은 여전히 단기 트래픽을 평활화하지만, 월별 할당량이 소진되면 다음 달까지 추가 호출이 모두 실패합니다.

Property	Type	Default	설명
Enabled	Boolean	False	할당량 규칙 집합을 활성화합니다. False이면 항목이 평가되지 않습니다.

할당량	TCollection	empty	TsgcRateLimitQuotaItem의 컬렉션입니다. 활성화된 모든 항목이 순서대로 평가됩니다.

각 **TsgcRateLimitQuotaItem**은 다음을 노출합니다.

Property	Type	Default	설명
Name	string	"	OnQuotaExceeded와 Result.Reason ("quota:Name")에서 보고되는 친숙한 이름입니다.
Enabled	Boolean	True	이 quota를 삭제하지 않고 비활성화합니다.
Scope	TsgcRateLimitQuotaScope	qscPerKey	qscPerKey는 키별로 독립적으로 카운트합니다. qscGlobal은 모든 키를 합쳐서 카운트합니다.
Period	TsgcRateLimitQuotaPeriod	qpHour	qpHour, qpDay 또는 qpMonth입니다. 카운터는 각 기간이 시작될 때 자동으로 재설정됩니다.
Limit	Int64	10000	해당 기간 내에 허용되는 최대 요청 수입니다.

```
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

동작: 메인 전략과 병렬로 실행되는 단기 스파이크 감지기입니다. *BurstWindowMs* 밀리초 내에 동일한 키에서 *BurstThreshold*개 이상의 요청이 도착하면, 해당 키는 *CooldownSec*초 동안 쿨다운에 놓이며, 그 동안에는 메인 전략이 허용했을 요청이라도 모든 요청이 거부됩니다.

사용 시기: 분당 한도 미만으로 유지하면서 100ms 버스트로 100개 이상의 요청을 서버에 쏟아붓는 스크래퍼 및 스캐너 클라이언트를 감지합니다. 버스트 보호는 인간 사용자가 절대 만들어 내지 않는 이 기관총 패턴을 포착합니다.

Property	Type	Default	설명
Enabled	Boolean	False	버스트 감지기를 활성화합니다.
BurstThreshold	Integer	50	쿨다운을 트리거하는 BurstWindowMs 내의 요청 수입니다.
BurstWindowMs	Integer	500	버스트 관찰 윈도우(밀리초)입니다.
CooldownSec	Integer	30	버스트가 감지된 후 쿨다운 기간의 지속 시간입니다. 쿨다운 동안에는 모든 요청이 거부됩니다.

```
// 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;
```

응답

기능: 요청이 제한될 때 클라이언트에 반환되는 HTTP 응답의 형태를 구성합니다. 상태 코드, 본문 메시지, Retry-After 및 X-RateLimit-* 헤더가 응답에 추가되는지 여부를 제어합니다.

사용 시기: 잘 동작하는 클라이언트가 올바르게 백오프하기를 원합니다. 클라이언트가 얼마나 기다려야 하는지를 나타내는 Retry-After 헤더와 함께 표준 429 상태를 반환하고, 클라이언트가 거부에 반응하는 대신 사전에 자체 속도를 조절할 수 있도록 X-RateLimit-Limit / X-RateLimit-Remaining / X-RateLimit-Reset을 포함하십시오.

Property	Type	Default	설명
StatusCode	Integer	429	스로틀된 요청에 대해 반환되는 HTTP 상태 코드입니다. 429(표준) 또는 유지 관리 스타일 거부의 경우 503을 사용하십시오.
Message	string	"Too Many Requests"	throttle된 요청에 대한 HTTP 응답 본문.
RetryAfterHeader	Boolean	True	True인 경우 계산된 재시도 지연(초)이 포함된 Retry-After 헤더를 추가합니다.
IncludeRateLimitHeaders	Boolean	True	True인 경우 모든 응답에 X-RateLimit-Limit / X-RateLimit-Remaining / X-RateLimit-Reset 헤더를 추가합니다.

```
sgcWSRateLimiter1->Response->StatusCode = 429;
sgcWSRateLimiter1->Response->Message = "Too Many Requests - please slow down";
sgcWSRateLimiter1->Response->RetryAfterHeader = true;
sgcWSRateLimiter1->Response->IncludeRateLimitHeaders = true;
```

Stats

기능: rate limiter에 대한 실시간 통계를 노출하는 읽기 전용 카운터 객체입니다. 모든 카운터는 구성 요소의 내부 critical section 내부에서 atomic하게 업데이트됩니다.

사용 시기: 관리자 UI의 실시간 대시보드입니다. TTimer에서 Stats를 폴링하고 현재 TotalRequests / TotalThrottled / ActiveKeys를 레이블, 게이지 또는 차트에 표시하여 운영자가 실시간으로 트래픽 압력을 볼 수 있게 하십시오.

Property	Type	설명
TotalRequests	Int64	마지막 Reset 이후 평가된 총 요청 수.
TotalThrottled	Int64	전략 또는 버스트 보호에 의해 거부된 총 요청 수.
TotalQuotaExceeded	Int64	quota에 도달하여 거부된 총 요청 수입니다.
ActiveKeys	Integer	현재 추적 중인 키 수(버킷 + 윈도우 카운터 + 활성 연결).
Uptime	Int64	stats 객체가 마지막으로 재설정된 이후의 초.

```
// 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";
}
```

Events

OnThrottled: 요청이 전략 또는 버스트 보호에 의해 거부될 때 발생합니다. 키(IP, API 키 또는 사용자)와 사유 문자열("token_bucket", "sliding_window", "fixed_window", "burst", "connection_rate" 또는 "message_rate")을 제공합니다. *Allow* 매개변수(var Boolean)를 사용하면 결정을 재정의하고 요청을 통과시킬 수 있습니다.

OnQuotaExceeded: 장기 할당량에 도달하면 발생합니다. 초과된 할당량의 키와 Name을 제공합니다.

OnStateChange: 키에 대한 Token Bucket 크기가 변경될 때 발생합니다. 이전 및 새 버킷 크기와 함께 키를 제공합니다. 텔레메트리, 디버깅 및 실시간 시각화에 유용합니다.

Public Methods

Method	설명
IsAllowed(Key, Cost)	지정된 키에 대해 주어진 Cost(기본값 1)의 요청이 허용되면 True를 반환합니다.
Consume(Key, Cost)	Cost token을 소비하려고 시도하고 TsgcRateLimitResult(Allowed, Remaining, RetryAfterSec, Reason)를 반환합니다.
Reset(Key)	지정된 키에 대한 카운터를 지웁니다.
ResetAll	추적되는 모든 키에 대한 모든 내부 카운터를 지우고 Stats 객체를 재설정합니다.
GetRemaining(Key)	지정된 키에 대해 아직 사용 가능한 요청 수를 반환합니다.
GetRetryAfter(Key)	호출자가 재시도하기 전에 기다려야 하는 초 수를 반환합니다.

SaveStateToFile(FileName)	모든 bucket과 카운터를 지정된 파일에 영속화합니다.
LoadStateFromFile(FileName)	이전에 파일에 저장된 버킷과 카운터를 복원합니다.
SaveStateToStream(Stream)	내부 상태를 임의의 스트림에 유지합니다.
LoadStateFromStream(Stream)	임의의 스트림에서 내부 상태를 복원합니다.
IsConnectionAllowed(IP)	서버 통합 후크. IP로부터의 새 연결이 속도 제한을 통과하면 True를 반환합니다. 서버에 의해 자동으로 호출됩니다.
IsMessageAllowed(IP, Message)	서버 통합 후크입니다. IP로부터의 메시지가 속도 제한을 통과하면 True를 반환합니다. 서버에 의해 자동으로 호출됩니다.
RegisterConnection(IP)	추적을 위해 새 연결을 등록합니다. 자동으로 호출됩니다.
UnregisterConnection(IP)	추적에서 연결을 제거합니다. 연결 끊김 시 자동으로 호출됩니다.

자동 통합

서버의 RateLimiter 속성에 할당되면 구성 요소는 자동으로 다음을 수행합니다:

- **새 연결 시:** IsConnectionAllowed를 호출하여 PerIP, 버스트 보호 및 전역 전략을 평가합니다. 제한이 초과되면 OnConnect 이벤트 전에 HTTP 429로 연결을 거부합니다. RegisterConnection을 호출하여 클라이언트를 추적합니다.
- **메시지 수신 시:** IsMessageAllowed를 호출하여 PerIP, PerEndpoint, PerAPIKey, PerUser, 할당량 및 버스트 보호를 확인합니다. 제한이 초과되면 연결을 끊거나 메시지를 거부합니다.
- **연결 해제 시:** 추적 리소스를 해제하기 위해 UnregisterConnection을 호출합니다.

전체 예제

```
// 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);
}

```

스레드 안전성

속도 제한기는 완전히 스레드 안전합니다. 모든 공용 메서드는 내부 임계 영역과 스레드 안전 카운터를 사용하여 동시 액세스를 보호합니다. 구성 요소는 여러 서버 인스턴스에서 공유될 수 있으며 외부 동기화 없이 모든 스레드(서버 이벤트 핸들러, 타이머 스레드, 백그라운드 워커)에서 액세스할 수 있습니다.

참조

- 속성 — 구성 요소에서 사용 가능한 13개의 속성.
- Methods — 12개의 public 메서드.
- 이벤트 — 처리할 수 있는 3개의 이벤트.

TsgcWSCircuitBreaker

*TsgcWSCircuitBreaker*는 HTTP API에 대한 클라이언트 측 호출에 대해 회로 차단기 복원력 패턴을 구현합니다.

소개

*TsgcWSCircuitBreaker*는 HTTP API에 대한 클라이언트 측 호출을 위한 Circuit Breaker 복원력 패턴을 구현합니다. 롤링 윈도우에서 실패를 추적하고, 구성 가능한 임계값에 도달하면 회로를 열고, 쿨다운 기간 동안 호출을 즉시 거부함으로써 원격 엔드포인트가 사용 불가능해질 때 연쇄 장애를 방지합니다. 실패 횟수 및 실패율 임계값, 느린 호출 감지, half-open 시험 호출, 예외 분류(기록/무시), 엔드포인트별 재정의, 폴백 응답 및 실시간 메트릭을 지원합니다.

circuit breaker는 **CircuitBreaker** 속성을 통해 **TsgcHTTPAPI_client**의 모든 자손(OpenAI, Anthropic, Gemini, Grok, DeepSeek, Mistral, Ollama, Stripe, Binance, Bybit, Coinbase, Kraken 및 기타 API 래퍼)과 자동으로 통합됩니다. 일단 할당되면, 모든 Get / Post / Put / Patch / Delete 호출이 자동으로 래핑됩니다. 고유한 호스트 이름당 하나의 독립 회로가 유지됩니다.

빠른 시작

1. 폼에 **TsgcWSCircuitBreaker** 구성 요소를 놓으십시오.

2. Thresholds(언제 열지) 및 Recovery(어떻게 복구할지)를 구성하십시오:

```
sgcWSCircuitBreaker1.Thresholds.FailureCount := 5;
sgcWSCircuitBreaker1.Recovery.CooldownSec := 30;
```

3. circuit breaker를 HTTP API 클라이언트(**TsgcHTTPAPI_client** descendant)에 할당하십시오:

```
sgcAI_OpenAI1.CircuitBreaker := sgcWSCircuitBreaker1;
```

4. 클라이언트에서 Get / Post / Delete / Patch / Put를 호출하십시오. breaker는 성공과 실패를 자동으로 기록하고 회로가 Open인 동안 새 호출을 거부합니다.

States

기능: circuit breaker는 3상태 머신입니다. 각 호스트(키)는 아웃바운드 호출이 허용, 거부 또는 시험으로 전송될지 제어하는 자체의 독립적인 상태를 가집니다.

State	Behavior	Transition
csClosed	모든 호출이 통과됩니다. 실패는 롤링 윈도우에서 계산됩니다.	FailureCount, FailureRatePercent 또는 SlowCallRatePercent에 도달하면 csOpen으로 이동합니다.
csOpen	모든 호출이 즉시 거부됩니다. 각 시도마다 OnCallRejected가 발생합니다.	Recovery.CooldownSec초가 열린 이후 경과하면 csHalfOpen으로 전환됩니다.

csHalfOpen	복구를 탐지하기 위해 최대 Recovery.HalfOpenTrialCalls 시험 호출이 허용됩니다.	시도가 성공하면 csClosed로 이동합니다. 어떤 시도든 실패하면 다시 csOpen으로 이동합니다.

최상위 속성

기능: 구성 요소의 마스터 스위치와 호출별 키가 제공되지 않을 때 사용되는 두 개의 키 이름입니다. 클라이언트 측 통합은 대상 호스트 이름을 키로 자동 사용합니다 — DefaultKey와 ServerKey는 수동 Execute/RecordSuccess/RecordFailure 호출과 선택적 서버 통합 헬퍼에만 적용됩니다.

사용 시기: HTTP 클라이언트에서 할당을 해제하지 않고도 제어된 부하 테스트 중 또는 유지 보수 기간 동안 런타임에 브레이커를 비활성화하십시오. 사용자 지정 코드 내에서 Execute를 수동으로 구동할 때 의미 있는 DefaultKey(예: 'openai-api')를 설정하십시오.

Property	Type	Default	설명
Enabled	Boolean	True	마스터 스위치입니다. False인 경우 모든 공개 메서드가 "허용됨"으로 단락되고 카운터가 업데이트되지 않습니다.
DefaultKey	string	'default'	Execute, RecordSuccess 및 RecordFailure의 매개변수 없는 오버로드에서 사용하는 키입니다.
ServerKey	string	'server'	선택적 서버 통합 혹(IsConnectionAllowed, IsMessageAllowed, RecordMessageError)이 사용하는 키입니다.

```
sgcWSCircuitBreaker1->Enabled = true;
sgcWSCircuitBreaker1->DefaultKey = "openai-api";
sgcWSCircuitBreaker1->ServerKey = "server";
```

Thresholds

기능: 회로를 Closed에서 Open으로 이동시키는 조건을 정의합니다. FailureCount 원시 실패에 도달하거나, 윈도우 내 호출의 FailureRatePercent가 실패하거나(MinCalls 이후), 호출의 SlowCallRatePercent가 SlowCallDurationMs를 초과하면 차단기가 열립니다.

사용 시기: 서버가 OpenAI의 API를 호출합니다. 60초 이내에 5번의 실패가 발생하거나(또는 최소 10번의 호출이 관찰된 후 호출의 50%가 실패하면), 회로를 열고 OpenAI를 계속 압박하지 마십시오. 재시도를 쌓는 대신 원격이 복구되도록 하십시오.

Property	Type	Default	설명
Enabled	Boolean	True	임계값 기반 열림을 활성화합니다. False인 경우 회로가 자동으로 열리지 않습니다.
FailureCount	Integer	5	브레이커를 트립시키는 롤링 윈도우 내 절대 실패 횟수입니다.

FailureRatePercent	Integer	50	MinCalls에 도달하면 breaker를 발동시키는 실패 백분율 (0-100).
SlowCallDurationMs	Integer	3000	성공한 호출도 "느린" 것으로 간주되는 임계값(ms)입니다.
SlowCallRatePercent	Integer	0	차단기를 작동시키는 느린 호출의 백분율(0-100)입니다. 0은 느린 호출 감지를 비활성화합니다.
MinCalls	Integer	10	속도 임계값이 평가되기 전에 관찰되는 최소 호출 수(작은 샘플에서 발동되는 것을 방지).

```
// 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

동작: 임계값 평가에 사용되는 롤링 윈도우를 정의합니다. 윈도우는 BucketCount 하위 버킷으로 나뉘므로 시간이 지남에 따라 오래된 카운터가 부드럽게 노후화됩니다. 임계값은 항상 활성 버킷의 합계에 대해 평가됩니다.

사용 시기: 일시적인 급증이 회로를 작동시키지 않지만 지속적인 실패는 작동시키도록 6개의 버킷이 있는 30초 롤링 윈도우를 사용하십시오. 더 긴 윈도우는 노이즈를 완화합니다. 더 짧은 윈도우는 중단에 더 빠르게 반응합니다. 더 많은 버킷은 더 부드러운 노화를 제공하지만 호스트당 약간 더 많은 메모리를 소비합니다.

Property	Type	Default	설명
RollingWindowSec	Integer	60	롤링 윈도우의 총 너비(초)입니다. 오래된 샘플은 폐기됩니다.
BucketCount	Integer	10	창이 나뉘는 하위 버킷 수. 각 버킷은 success / failure / slow 카운터를 보유합니다.

```
// 30-second window split into 6 buckets (5 seconds each)
sgcWSCircuitBreaker1->TimeWindow->RollingWindowSec = 30;
sgcWSCircuitBreaker1->TimeWindow->BucketCount = 6;
```

Recovery

기능: circuit이 Open 상태에서 복구되는 방식을 제어합니다. CooldownSec가 경과한 후 breaker는 HalfOpen으로 전환되어 HalfOpenTrialCalls 프로브 호출을 통과시킵니다. trial이 성공하면 breaker가 닫히고, 실패하면 다시 열리고 cooldown 타이머가 다시 시작됩니다.

사용 시점: 열린 후 30초 대기한 다음 다시 닫기 전에 복구를 테스트하기 위해 하나의 시험 호출을 통과시킵니다. 더 긴 냉각 기간은 어려움을 겪는 업스트림을 또 다른 트래픽 파도로부터 보호합니다. 짧은 냉각 기간은 잠깐 깜박이는 서비스에 대해 빠르게 다시 엽니다.

Property	Type	Default	설명
CooldownSec	Integer	30	HalfOpen으로 전환되기 전에 Open 상태에서 대기할 초입니다.
HalfOpenTrialCalls	Integer	1	닫거나 다시 열기로 결정하기 전에 HalfOpen에서 허용되는 trial 호출 수입니다.
AutoReset	Boolean	True	True인 경우 상태 전환 시 카운터가 자동으로 지워집니다.
MaxRetries	Integer	3	재시도 도우미가 사용하는 정보 제공용 상한선. 차단기 자체에 의한 자동 재시도는 수행되지 않습니다.

```
// 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

기능: 회로가 Open인 동안 제공되는 대체 페이로드를 구성합니다. Fallback.Enabled가 True이고 Execute / ExecuteWithResult가 호출을 거부하면, CachedResponse(또는 CustomMessage)에서 초기화된 응답과 함께 OnFallback 이벤트가 발생합니다. 처리기는 이 문자열이 호출자에게 반환되기 전에 교체할 수 있습니다.

사용 시기: OpenAI가 다운되었을 때 캐시된 마지막 정상 응답을 반환하여, 사용자가 오류 대신 품질이 저하되었지만 작동하는 동작(예: 이전에 생성된 요약)을 보도록 하십시오. 정적인 "유지 보수 모드" JSON 본문에는 CustomMessage를 사용하십시오.

Property	Type	Default	설명
Enabled	Boolean	False	fallback 경로를 활성화합니다. False인 경우, 거부된 호출은 OnFallback을 발생시키지 않고 단순히 반환됩니다.
CachedResponse	string	"	circuit이 Open일 때 반환되는 정적 페이로드입니다. CustomMessage보다 우선합니다.
CustomMessage	string	"	CachedResponse가 비어 있을 때 사용되는 fallback 페이로드.
UseLastSuccess	Boolean	False	True인 경우, 키에 대해 관찰된 마지막 성공적인 응답이 폴백으로 제공됩니다.

```
// 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;
```

분류

동작: 발생한 예외를 "실패로 기록됨" 또는 "무시됨"으로 분류합니다. 기록된 예외만 실패 카운터를 증가시킵니다. MatchMode는 RecordAsFailure / IgnoreExceptions의 문자열이 예외 텍스트(ClassName: Message)와 어떻게 비교되는지를 선택합니다.

사 용 시 기: 500/502/503/504 가 있는 `HttpException` 은 실패로 처리 하 되, 400/401/404 가 있는 `HttpException`은 무시하십시오(business error는 circuit을 트립해서는 안 됨). RecordAsFailure를 비워 두면 무시되지 않은 모든 예외가 기록됩니다.

Property	Type	Default	설명
RecordAsFailure	TStringList	empty	예외 텍스트에 대해 매칭되는 패턴입니다. 매칭되는 예외는 실패로 계산됩니다. 빈 목록 = 무시되지 않은 모든 것을 기록합니다.
IgnoreExceptions	TStringList	empty	예외 텍스트에 대해 일치되는 패턴. 일치하는 예외는 폐기됩니다 (성공도 실패도 아님).
MatchMode	TsgcCircuitBreakerExceptionMatch	cemContains	비교 모드: cemExact, cemContains, cemStartsWith 또는 cemWildcard(* / ?).

```
// 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

기능: 패턴 기반 재정의의 컬렉션입니다. 각 `TsgcCircuitBreakerEndpointItem`은 자체 Pattern(와일드카드 * 및 ? 지원), OverrideThresholds 블록 및 OverrideRecovery 블록을 가집니다. 활성 키(호스트 이름)가 활성화된 항목과 일치하면 호출 기간 동안 해당 항목의 설정이 최상위 Thresholds / Recovery를 대체합니다.

사용 시기: API별 회로 분리: `api.openai.com`은 관대한 임계값을 가지지만(부하 시 가끔 느려짐), `api.stripe.com`은 결제에 중요하므로 fail closed를 위해 더 빨리 열려야 합니다. 조정된 임계값으로 호스트당 하나의 엔드포인트 항목을 추가하십시오.

Property	Type	Default	설명
Enabled	Boolean	False	엔드포인트 재정의 컬렉션을 활성화합니다. False이면 항목이 평가되지 않습니다.
Items	TCollection	empty	TsgcCircuitBreakerEndpointItem의 컬렉션입니다. 활성화된 첫 번째 일치 항목이 우선합니다.

각 **TsgcCircuitBreakerEndpointItem**은 다음을 노출합니다:

Property	Type	Default	설명
패턴	string	"	키(호스트 이름)에 대해 일치되는 와일드카드 패턴입니다. * 및 ? 를 지원합니다.
Enabled	Boolean	True	이 항목을 삭제하지 않고 비활성화합니다.
OverrideThresholds	TsgcCircuitBreakerThresholds	defaults	패턴이 일치할 때 적용되는 임계값입니다. 자체 Enabled가 True일 때만 활성화됩니다.
OverrideRecovery	TsgcCircuitBreakerRecovery	defaults	패턴이 일치할 때 적용되는 복구 설정.

```
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

기능: 호출이 breaker를 통과할 때 원자적으로 업데이트되는 읽기 전용 카운터. 모든 키 / 호스트명에 걸쳐 집계됩니다. CurrentOpenBreakers는 모든 상태 전환 후에 새로 고쳐집니다. AverageLatencyMs는 Execute가 관찰한 지속 시간의 누적 평균입니다.

사용 시기: 호스트별 회로 상태, 실패율 및 거부 수를 보여주는 라이브 대시보드. 운영자를 위한 게이지를 그리기 위해 TTimer에서 Metrics를 폴링하거나, 경보를 위해 Prometheus / StatsD로 방출하십시오.

Property	Type	설명
TotalCalls	Int64	처리된 총 호출 수(성공 + 기록된 실패).
TotalFailures	Int64	모든 키에 걸쳐 기록된 총 실패 수입니다.
TotalSuccesses	Int64	모든 키에서 기록된 총 성공 호출 수입니다.
TotalRejected	Int64	회로가 Open이어서 거부된 총 호출 수입니다.
CurrentOpenBreakers	Integer	현재 Open 상태인 키의 수입니다.
AverageLatencyMs	Double	Execute / ExecuteWithResult 호출을 중심으로 측정된 실행 중 평균 지연 시간(ms)입니다.

```
// 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";
}
```

Events

OnStateChange: 회로가 상태 간에 전환될 때 발생합니다. 키, 이전 상태, 새 상태(csClosed, csOpen 또는 csHalfOpen)를 제공합니다. 로깅, 경고, 라이브 UI 업데이트에 유용합니다.

OnCallRejected: 회로가 Open 상태이거나 HalfOpen 시험 예산이 소진되어 호출이 거부될 때 발생합니다. 키와 이유 문자열(예: Circuit open)을 제공합니다.

OnFallback: 폴백 응답이 반환되기 직전에 발생합니다. aResponse 매개변수(var string)는 Fallback.CachedResponse / CustomMessage에서 초기화되며 핸들러가 대체할 수 있습니다.

OnFailureRecorded: 실패가 기록될 때마다 발생합니다. 키와 예외 텍스트(Classname: Message)를 제공합니다. 구조화된 로깅이나 업스트림 인시던트 추적과의 상관 관계 분석에 사용하십시오.

OnSlowCall: 성공한 호출이 Thresholds.SlowCallDurationMs를 초과할 때 발생합니다. 키와 측정된 지속 시간을 밀리초 단위로 제공합니다.

Public Methods

Method	설명
Execute(Key, Action)	지정된 키 아래에서 TProc를 실행합니다. 먼저 IsCallAllowed를 확인하고 성공/실패/지연을 자동으로 기록하며 작업이 실행되면 True를 반환합니다.
Execute(Action)	DefaultKey를 사용하는 오버로드입니다.
ExecuteWithResult(Key, Action, out Result)	(D2009+) TFunc<TObject>를 실행하고 out 매개변수를 통해 결과를 반환합니다. Execute와 동일한 회계를 사용합니다.
RecordSuccess(Key)	성공적인 호출을 기록합니다. 시도가 성공하면 HalfOpen -> Closed로 전환합니다.
RecordSuccess	DefaultKey를 사용하는 오버로드입니다.
RecordFailure(Key, Exception)	선택적 예외 텍스트와 함께 실패를 기록합니다. 전환 평가를 트리거합니다.
RecordFailure(Exception)	DefaultKey를 사용하는 오버로드입니다.
ForceOpen(Key)	수동으로 회로를 Open 상태로 강제합니다.
ForceClose(Key)	수동으로 회로를 Closed 상태로 강제합니다.
Reset(Key)	지정된 키에 대한 카운터와 상태를 지웁니다.
ResetAll	추적된 모든 circuit을 지우고 Metrics를 재설정합니다.
GetState(Key)	해당 키에 대한 현재 TsgcCircuitState를 반환합니다.
IsCallAllowed(Key)	새 호출이 현재 허용되면 True를 반환합니다. 모든 요청 전에 HTTP API 클라이언트가 자동으로 호출합니다.
GetFailureRate(Key)	롤링 윈도우에서 관찰된 현재 실패 백분율(0.0 - 100.0)을 반환합니다.
GetTotalCalls(Key)	키에 대한 롤링 윈도우에서 관찰된 호출 수를 반환합니다.
GetLastStateChange(Key)	마지막 상태 전환의 TDateTime을 반환합니다.
GetOpenBreakers	현재 Open 상태에 있는 키의 TArray<string>를 반환합니다.
SaveStateToFile(FileName)	모든 회로의 상태와 카운터를 파일에 유지합니다.
LoadStateFromFile(FileName)	이전에 저장된 파일에서 회로 상태를 복원합니다.
IsConnectionAllowed(IP)	서버 측 자체 보호 후크. 기본 클라이언트 측 사용에 보조적입니다. ServerKey 회로가 Open이 아닌 경우 True를 반환합니다.
IsMessageAllowed(IP, Message)	서버 측 자체 보호 후크입니다. ServerKey 회로가 Open이 아니면 True를 반환합니다.
RegisterConnection(IP)	향후 IP별 메트릭을 위해 예약된 서버 측 후크입니다(현재 작동하지 않음).

UnregisterConnection(IP)	향후 IP별 메트릭을 위해 예약된 서버 측 훅입니다(현재 작동하지 않음).
RecordMessageError(IP, Exception)	서버 측 자체 보호 후크입니다. ServerKey에 실패를 기록합니다.
RecordMessageSuccess(IP)	서버 측 자체 보호 후크. ServerKey에 성공을 기록합니다.

자동 통합

CircuitBreaker 속성을 통해 TsgcHTTPAPI_client 자손에 할당되면, 클라이언트는 자동으로 다음을 수행합니다:

- **모든 Get / Post / Put / Patch / Delete 전에:** 대상 URL 호스트 이름(예: api.openai.com)에서 키를 호출하고 IsCallAllowed(host)를 호출합니다. 회로가 Open이면 호출이 예외 *"Circuit breaker open for <host> - call rejected"*와 함께 거부되고 OnCallRejected가 발생합니다.
- **성공적인 HTTP 응답 시:** RecordSuccess(host)를 호출하여 카운터를 업데이트하며, 원격이 복구될 때 HalfOpen trial이 circuit을 닫을 수 있도록 합니다.
- **예외 발생 시:** RecordFailure(host, exception.Message)를 호출하고, Classification 규칙을 적용하여 계산할지 무시할지 결정하며, 전환을 평가한 다음, 호출자에게 예외를 다시 발생시킵니다.
- **호스트당 하나의 circuit:** 서로 다른 호스트(예: api.openai.com, api.anthropic.com, api.stripe.com)에 대한 호출은 완전히 독립적인 circuit을 추적합니다.

전체 예제

```
// 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 TsgcHTTPAPI_client descendant)
```

```

sgcAI_OpenAI1->CircuitBreaker = sgcWSCircuitBreaker1;

// Use the client normally; the breaker intercepts Get / Post / etc.
try
{
    String Response = sgcAI_OpenAI1->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);
}

```

스레드 안전성

circuit breaker는 완전히 스레드 안전합니다. 모든 public 메서드는 내부 임계 영역과 스레드 안전 목록을 사용하여 키별 상태, 롤링 윈도우 및 메트릭 카운터에 대한 동시 액세스를 보호합니다. 단일 구성 요소를 여러 HTTP API 클라이언트가 공유할 수 있으며 외부 동기화 없이 모든 스레드(이벤트 핸들러, 타이머, 백그라운드 작업자)에서 액세스할 수 있습니다.

참조

- 속성 — 구성 요소에서 사용 가능한 속성 10개입니다.
- 메서드 — 17개의 public 메서드입니다.
- 이벤트 — 처리할 수 있는 5개의 이벤트.

TsgcWSAPIKeyManager

*TsgcWSAPIKeyManager*는 *sgcWebSockets* 서버가 발급한 API 키에 대한 전체 수명 주기 관리를 제공합니다.

소개

*TsgcWSAPIKeyManager*는 *sgcWebSockets* 서버가 발급한 API 키에 대한 전체 수명 주기 관리를 제공합니다. 구성 가능한 사람이 읽을 수 있는 접두사로 암호학적으로 무작위인 키를 생성하고, SHA-256, SHA-512 또는 Bcrypt로 저장 시 해시하고, 헤더 또는 쿼리 문자열로 들어오는 요청을 검증하고, 구성 가능한 grace period로 키를 회전하고, 키를 취소 및 갱신하고, scope 기반 권한 부여를 적용하고, 영구 audit log를 유지하며, 선택적 IP 허용 목록을 지원합니다. 키는 메모리 또는 재시작에도 유지되는 암호화된 파일에 저장됩니다.

관리자는 **APIKeyManager** 속성을 통해 *TsgcWebSocketHTTPServer* 및 *TsgcWSServer_HTTPAPI*에 연결됩니다. 서버의 자동 후크는 피어 IP만 보기 때문에, 실제 요청 인증은 서버의 OnConnect 이벤트에서 **IsRequestAuthorized**를 호출하여 수행됩니다. 이를 통해 전체 HTTP 헤더 블록과 URL/쿼리 문자열에 액세스할 수 있으므로, 구성된 **X-API-Key** 헤더(또는 쿼리 매개변수)를 구문 분석, 검증하고, 요청 시 필요한 범위와 일치시킬 수 있습니다.

빠른 시작

1. 폼에 **TsgcWSAPIKeyManager** 구성 요소를 놓으십시오.
2. Generation(prefix, length), Hashing(algorithm, salt) 및 Validation(header name, IP allowlist, FailClosed)을 구성하십시오. 아래의 섹션별 문서를 참조하십시오.
3. 매니저를 서버 구성 요소에 할당하십시오:

```
sgcWebSocketHTTPServer1.APIKeyManager := sgcWSAPIKeyManager1;
```

4. 연결의 헤더, URL 및 IP를 사용하여 **IsRequestAuthorized**를 호출하도록 서버의 OnConnect를 연결하십시오. False를 반환하면 클라이언트를 연결 해제하십시오.
5. 고객에게 첫 번째 키를 발급하고 평문 값을 정확히 한 번만 전달하십시오. 그 이후에는 해시된 형태만 저장소에 남습니다.

최상위 속성

동작: 전체 구성 요소의 마스터 스위치입니다. Enabled가 False이면 모든 공용 메서드가 "허용됨"으로 단락되고 서버 자동 후크는 작동하지 않으므로, 키 관리자가 사실상 우회됩니다.

사용 시기: 제어된 유지 관리 기간 또는 스테이징 배포 중에 구성 요소를 서버에서 할당 해제하지 않고 키 적용을 일시적으로 비활성화하십시오. 변경이 완료되면 다시 활성화하십시오 — 키가 손실되지 않고, 범위가 변경되지 않으며, 다시 켜는 순간 감사 로그가 계속 실행됩니다.

Property	Type	Default	설명
Enabled	Boolean	True	마스터 스위치입니다. False인 경우 ValidateKey / IsRequestAuthorized는 항상 True를 반환하고 서버 자동 후크는 작동하지 않습니다.


```
sgcWSAPIKeyManager1->Enabled = true;
```

Generation

동작: *IssueKey*에 의해 일반 텍스트 키 문자열이 구성되는 방식을 제어합니다. 키는 *KeyPrefix* 뒤에 *Charset*에서 추출한 *KeyLength*개의 임의 문자가 따라옵니다. *IncludeChecksum*이 *True*인 경우 짧은 체크섬이 추가되어 해시 조회 전에 잘못된 형식의 키를 저렴하게 거부할 수 있습니다.

사용 시점: SaaS용 브랜드 키 — 프로덕션에서는 *sgc_live_aB3...*를, 스테이징 클러스터에서는 *sgc_test_...*를 사용하십시오. 접두사를 통해 지원 직원은 유출된 키가 어떤 환경에 속하는지 쉽게 파악할 수 있고 로그 스크러버는 이를 수정(redact)할 수 있습니다.

Property	Type	Default	설명
Enabled	Boolean	True	False인 경우 <i>IssueKey</i> 는 새 키를 생성하는 대신 예외를 발생시킵니다.
KeyPrefix	string	"	무작위 본문 앞에 추가되는 사람이 읽을 수 있는 접두사입니다(예: <i>sgc_live_</i>).
KeyLength	Integer	32	키 본문의 무작위 문자 수입니다.
Charset	string	A-Za-z0-9	무작위 본문을 생성하는 데 사용되는 알파벳입니다. 비어 있으면 기본적으로 URL-safe 영숫자입니다.
IncludeChecksum	Boolean	False	True인 경우 짧은 체크섬 접미사를 추가하여 명백히 잘못된 형식의 키를 해시 조회 없이 거부할 수 있습니다.

```
// Branded production keys: sgc_live_aB3...
sgcWSAPIKeyManager1->Generation->Enabled = true;
sgcWSAPIKeyManager1->Generation->KeyPrefix = "sgc_live_";
sgcWSAPIKeyManager1->Generation->KeyLength = 32;
sgcWSAPIKeyManager1->Generation->IncludeChecksum = true;
```

Storage

기능: 해시된 키와 audit log가 저장되는 위치를 제어합니다. *kstMemory*는 모든 것을 프로세스 내부에 유지합니다(종료 시 지워짐). *kstFile*은 *FileName*에 영속화하며, 선택적으로 *EncryptionKey*로 암호화하고, *AutoSaveSeconds*초마다 다시 저장합니다. *kstCustom*은 자체 *SaveToFile* / *LoadFromFile* hook을 구현할 수 있도록 합니다.

사용 시기: 빠른 반복을 위해 개발 중에는 키를 메모리에 보관하십시오. 프로덕션에서는 키가 서버 재시작 후에도 유지되고 디스크에 평균으로 노출되지 않도록 *EncryptAtRest := True*와 함께 *kstFile*로 전환하십시오.

Property	Type	Default	설명

StorageType	TsgcAPIKeyStorageType	kstMemory	kstMemory (휘발성), kstFile (FileName에 유지) 또는 kstCustom (사용자 제공 SaveToFile / LoadFromFile).
FileName	string	"	StorageType이 kstFile일 때 SaveToFile / LoadFromFile에서 사용되는 파일 경로.
EncryptAtRest	Boolean	False	True인 경우 EncryptionKey를 사용하여 디스크의 키 스토어를 암호화합니다.
EncryptionKey	string	"	유지된 키 파일을 암호화하는 데 사용되는 비밀입니다. EncryptAtRest가 True인 경우 제공해야 합니다.
AutoSaveSeconds	Integer	0	> 0인 경우 관리자는 AutoSaveSeconds초마다 키 저장소를 디스크에 다시 저장합니다.

```
// 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

기능: 평문 키는 절대 영속화되지 않습니다. IssueKey는 선택적 Salt 및 Iterations 수와 함께 Algorithm을 사용하여 생성된 키를 해시합니다. ValidateKey는 들어오는 값을 다시 해시하고 저장된 digest에 대해 constant-time 비교를 수행합니다.

사용 시기: 평문을 절대 저장하지 마십시오. SHA-256은 대부분의 배포에 적합한 기본값입니다. 더 높은 보안 프로파일(FIPS 140-2 Level 3)에는 SHA-512를 사용하십시오. 의도적인 키 스트레칭이 필요할 때는 Bcrypt를 사용하십시오. 검증은 느리지만 유출된 키 저장소의 오프라인 무차별 대입을 극적으로 어렵게 만듭니다.

Property	Type	Default	설명
Algorithm	TsgcAPIKeyHashAlgorithm	khaSHA256	at-rest 저장에 사용되는 해시 함수: khaSHA256, khaSHA512 또는 khaBcrypt.
Salt	string	"	모든 해시에 혼합되는 정적 salt. 변경하면 기존의 모든 키가 무효화됩니다.
Iterations	Integer	1	hashing round 수입니다. 검증 지연을 대가로 key-stretching을 위해 증가시키십시오.

```
// 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

기능: 허용된 scope 문자열의 카탈로그입니다. *Enabled*가 True이면 발급된 키에 첨부된 모든 scope가 이 목록에 존재해야 하며, 알 수 없는 scope는 거부됩니다. 개별 카탈로그 항목은 삭제하지 않고 비활성화할 수 있습니다. *ValidateKey / IsRequestAuthorized*는 요청당 특정 scope를 요구할 수 있습니다.

사용 시기: 세분화된 permission. 한 고객의 키는 *read:orders*만 전달하며 *write* 엔드포인트에서 거부됩니다. 파트너 통합은 *read:orders + write:shipments*를 받지만 절대 *admin:**는 받지 않습니다. 각 요청은 scope hint를 제시하고 manager가 이를 일관되게 적용합니다.

Property	Type	Default	설명
Enabled	Boolean	True	scope 적용을 활성화합니다. False인 경우 scope 매개변수가 무시됩니다.
Scopes	TCollection	empty	허용된 scope를 정의하는 TsgcAPIKeyScopeItem 항목의 컬렉션입니다.

각 *TsgcAPIKeyScopeItem*은 다음을 노출합니다:

Property	Type	Default	설명
Name	string	"	Scope 식별자입니다(예: <i>read:orders</i>). 대소문자를 구분합니다.
설명	string	"	object inspector 및 admin UI에 표시되는 사람이 읽을 수 있는 설명입니다.
Enabled	Boolean	True	이 범위를 삭제하지 않고 비활성화합니다. 비활성화된 범위는 HasScope를 만족하지 않습니다.

```
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

기능: 키 회전을 투명하게 만듭니다. *RotateKey*는 동일한 소유자와 범위에 대해 새 키를 발급하고 이전 키를 *kksRotated*로 표시합니다. *GracePeriodSec* 윈도우 동안 두 키 모두 검증되므로 고객은 다운타임 없이 교체할 수 있습니다. *AutoRotateDays*

가 > 0이면 해당 임계값보다 오래된 키는 백그라운드 스왑에 의해 회전됩니다. *NotifyBeforeExpirySec*는 OnKeyExpired를 일찍 발생시켜 고객에게 이메일을 보낼 수 있게 합니다.

사용 시기: 고객이 자신의 키가 유출되었다고 보고하지만 통합은 여전히 활성 상태입니다. 키를 회전하십시오: 고객은 새 키를 받고, 이전 키는 인프라 전반에 걸쳐 롤링할 수 있도록 24시간 동안 계속 작동한 다음, 이전 키는 자동으로 무효화됩니다.

Property	Type	Default	설명
Enabled	Boolean	False	회전 기능을 활성화합니다(유예 기간 및 자동 회전 스왑).
GracePeriodSec	Integer	86400	RotateKey 후 이전 키가 유효하게 유지되는 초 수. 기본값은 24시간입니다.
AutoRotateDays	Integer	0	> 0인 경우 백그라운드 스왑이 이 일수보다 오래된 키를 회전시킵니다.
NotifyBeforeExpirySec	Integer	604800	키가 실제로 만료되기 이만큼의 초 전에 OnKeyExpired를 발생시킵니다. 기본값은 7일입니다.

```
// 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;
```

만료

기능: 발급된 키가 어떻게 만료되는지 제어합니다. *DefaultTTLSec*는 호출자가 ExpiresInSec로 0을 전달할 때 IssueKey에 적용됩니다. *EnforceExpiry*는 만료된 키가 거부되는지 여부를 토글합니다. *CheckIntervalSec*는 백그라운드 스왑이 만료된 항목을 얼마나 자주 스캔하고, OnKeyExpired를 발생시키며, Stats를 업데이트하는지 제어합니다.

사용 시기: 파트너 통합 키는 90일 후에 만료되어야 합니다. 파트너는 계약을 다시 서명하고 새 키를 가져와야 합니다. DefaultTTLSec을 90×86400으로 설정하면 호출자가 명시적으로 재정의하지 않는 한 모든 IssueKey 호출이 해당 정책을 준수합니다.

Property	Type	Default	설명
DefaultTTLSec	Integer	0	새로 발급된 키의 기본 수명(초)입니다. 0은 만료되지 않음을 의미합니다.
EnforceExpiry	Boolean	True	True인 경우 ValidateKey는 ExpiresAt이 과거인 키를 거부합니다.
CheckIntervalSec	Integer	60	백그라운드 스왑이 만료된 키를 스캔하고 OnKeyExpired를 발생시키는 빈도입니다.

```
// 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

기능: 각 키와 함께 키별 rate-limit 기본값(윈도우당 최대 요청 수)을 저장합니다. 이러한 값은 TsgcWSRateLimiter 인스턴스가 PerAPIKey scope를 통해 읽어 tier에 맞는 throttling을 적용할 수 있는 정보 힌트입니다. API key manager 자체는 트래픽을 throttle하지 않습니다.

사용 시기: Tiered SaaS. Free-tier 고객은 시간당 100개 요청, Pro-tier는 시간당 10,000개 요청, Enterprise는 시간당 1,000,000개 요청으로 프로비저닝됩니다. API 키 관리자에서 DefaultMaxRequests / DefaultWindowSec를 구성하고 이를 TsgcWSRateLimiter.PerAPIKey와 결합하여 rate limiter가 키별 예산을 자동으로 읽도록 하십시오.

Property	Type	Default	설명
Enabled	Boolean	False	발급된 각 키와 함께 키별 속도 제한 메타데이터를 활성화합니다.
DefaultMaxRequests	Integer	100	새 키에 할당되는 윈도우당 기본 최대 요청 수입니다.
DefaultWindowSec	Integer	60	DefaultMaxRequests와 연결된 기본 윈도우 크기(초).

```
// 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

기능: 모든 키 수명 주기 작업(발급, 유효성 검사, 취소, 교체, 만료, 범위 부여/취소, 유효성 검사 실패)을 인메모리 링 버퍼에 기록하고, LogFile이 설정된 경우 동일한 항목을 파일에 추가합니다. IncludeIP는 요청자의 IP를 기록하고, IncludePayload는 추가 세부 정보를 캡처합니다. RetentionDays는 오래된 항목을 정리하고, MaxMemoryEntries는 링 버퍼를 제한합니다.

사용 시기: 규정 준수 요구 사항: 모든 키 작업은 기록되고 12개월 동안 보관되어야 합니다. SIEM 드롭 존을 가리키는 LogFile과 365로 설정된 RetentionDays로 감사를 활성화하십시오. 모든 항목에 대해 OnAuditEvent가 발생하므로 메시지 버스로도 전송할 수 있습니다.

Property	Type	Default	설명
Enabled	Boolean	True	감사 로그를 활성화합니다. False인 경우 항목이 기록되지 않으며 OnAuditEvent가 발생하지 않습니다.
LogFile	string	"	감사 로그가 추가되는 선택적 파일 경로입니다.
IncludeIP	Boolean	True	True인 경우 모든 항목이 요청자의 IP를 기록합니다.

IncludePayload	Boolean	False	True인 경우 각 감사 항목의 Details 필드에 추가 컨텍스트가 포함됩니다.
RetentionDays	Integer	90	이 일수보다 오래된 항목은 백그라운드 스왑에 의해 정리됩니다.
MaxMemoryEntries	Integer	10000	in-memory ring buffer에 보관되는 최대 항목 수입니다.

```
// Compliance: 12-month retention, log to file, include IP + payload
sgcWSAPIKeyManager1->Audit->Enabled = true;
sgcWSAPIKeyManager1->Audit->LogFile = "apikeys-audit.log";
sgcWSAPIKeyManager1->Audit->IncludeIP = true;
sgcWSAPIKeyManager1->Audit->IncludePayload = true;
sgcWSAPIKeyManager1->Audit->RetentionDays = 365;
sgcWSAPIKeyManager1->Audit->MaxMemoryEntries = 50000;
```

Validation

기능: 클라이언트가 키를 제시하는 방법과 어떤 조건에서 요청이 수락되는지를 관리자에게 알려줍니다. *HeaderName*(기본값 X-API-Key)과 *QueryParamName*(기본값 api_key)은 ExtractKeyFromHeaders / ExtractKeyFromQuery가 찾는 두 위치입니다. *RequireHTTPS*는 평문 트래픽을 거부합니다. *IPAllowlist*는 키를 특정 소스 IP로 제한합니다. *FailClosed*(기본값 True)는 관리자가 명확하게 권한을 부여할 수 없는 모든 요청을 거부합니다.

사용 시기: 기업 전용 API: 모든 요청에 HTTPS를 요구하고 키를 사무실 egress CIDR 범위로 제한하여 유출된 키조차도 기업 네트워크 외부에서 악용될 수 없도록 합니다.

Property	Type	Default	설명
HeaderName	string	X-API-Key	들어오는 키를 읽는 데 사용되는 HTTP 헤더 이름입니다.
QueryParamName	string	api_key	헤더가 없을 때 폴백으로 사용되는 쿼리 문자열 매개변수.
RequireHTTPS	Boolean	False	True이면 키 유효성에 관계없이 HTTPS가 아닌 모든 요청을 거부합니다.
IPAllowlist	TStringList	empty	비어 있지 않은 경우, IP가 목록에 있는 요청만 권한이 부여됩니다.
FailClosed	Boolean	True	True인 경우 관리자가 명확하게 권한 부여할 수 없는 모든 요청이 거부됩니다.

```
// 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

기능: 키 스토어에 대한 라이브 통계를 노출하는 읽기 전용 카운터 객체. 모든 카운터는 구성 요소의 내부 임계 영역 내에서 원자적으로 업데이트됩니다.

사용 시기: 실시간 admin 대시보드. TTimer에서 Stats를 폴링하고 현재 ActiveKeys / RevokedKeys / TotalValidations / TotalValidationFailures를 표시하여 운영자가 인증 부하를 실시간으로 볼 수 있게 하십시오.

Property	Type	설명
TotalKeys	Int64	이 manager 인스턴스가 발급한 총 키 수입니다.
ActiveKeys	Int64	현재 kksActive 상태에 있는 키의 수입니다.
RevokedKeys	Int64	현재 kksRevoked 상태의 키 수입니다.
ExpiredKeys	Int64	현재 kksExpired 상태의 키 수입니다.
TotalValidations	Int64	ValidateKey / IsRequestAuthorized 호출의 총 수입니다.
TotalValidationFailures	Int64	키를 거부한 검증 호출의 총 횟수입니다.
LastIssuedAt	TDateTime	가장 최근 IssueKey 호출의 타임스탬프입니다.

```
// 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);
}
```

Events

OnKeyIssued: 새 키가 발급되면 발생합니다. 소유자, 원시 평문 키(이를 관찰할 수 있는 마지막 기회), scope 배열을 제공합니다. 이 핸들러에서 평문을 고객에게 전달하십시오.

OnKeyValidated: ValidateKey가 완료될 때마다 발생합니다. 키, 부울 aValid 플래그 및 기계 판독 가능한 이유 문자열을 제공합니다.

OnKeyRevoked: 키가 취소되었을 때 발생합니다. 키와 취소 이유를 제공합니다.

OnKeyRotated: 키가 회전되면 발생합니다. 고객에게 알릴 수 있도록 이전 키와 새 키를 제공합니다.

OnKeyExpired: 키의 ExpiresAt가 경과했을 때(또는 NotifyBeforeExpirySec초 이전에) 백그라운드 만료 스웸에 의해 발생합니다.

OnValidation: ValidateKey 중에 발생하는 최종 결정 후크입니다. Allow := False로 설정하고 Reason을 제공하여 그렇지 않으면 유효한 키를 거부하십시오(사용자 정의 비즈니스 규칙).

OnAuditEvent: 모든 감사 로그 항목마다 발생합니다. TsgcAPIKeyAuditEntry 레코드(Timestamp, KeyPrefix, Action, IP, Details)를 제공합니다. 외부 SIEM으로 전달하는 데 이상적입니다.

Public Methods

Method	설명
IssueKey(Owner, Scopes, ExpiresInSec)	새 키를 생성, 해시 및 저장합니다. 평문을 반환합니다(볼 수 있는 유일한 시간). Expiration.DefaultTTLSec를 상속하려면 ExpiresInSec = 0을 전달하십시오.
ValidateKey(Key, RequiredScope, IP)	원시 키를 검증합니다. 선택적으로 필요한 범위를 적용하고 요청자의 IP를 기록합니다. 성공 시 True를 반환합니다.
RevokeKey(Key, Reason)	키를 취소됨으로 표시합니다. 이후 ValidateKey 호출은 False를 반환합니다.
RotateKey(OldKey, out NewKey)	동일한 소유자와 범위에 대해 새 키를 발급합니다. 이전 키는 kksRotated로 표시되고 GracePeriodSec 동안 허용됩니다.
RenewKey(Key, NewExpiresInSec)	지금부터 지정된 초 수만큼 키의 만료를 연장합니다.
GetKeyInfo(Key)	owner, scopes, status, timestamps 및 usage count가 있는 TsgcAPIKeyInfo 레코드를 반환합니다.
GetKeysByOwner(Owner)	주어진 소유자와 연결된 해시된 키 식별자를 반환합니다.
ListScopes(Key)	키에 첨부된 스코프를 반환합니다.
IsExpired(Key)	키가 만료 기한을 지났으면 True를 반환합니다.
IsRevoked(Key)	키가 폐기되면 True를 반환합니다.
KeyExists(Key)	키가 알려진 경우(활성, 취소됨 또는 만료됨) True를 반환합니다.
GrantScope(Key, Scope)	기존 키에 scope을 추가합니다.
RevokeScope(Key, Scope)	기존 키에서 범위를 제거합니다.
HasScope(Key, Scope)	키가 주어진 범위를 가지고 있으면 True를 반환합니다.
GetAuditLog(Key, MaxEntries)	감사 항목을 반환합니다. Key가 비어 있으면 모든 키에 대한 항목을 반환합니다.
ClearAuditLog(Key)	특정 키 또는 (비어 있으면) 모든 키에 대한 감사 로그를 지웁니다.
SaveToFile(FileName)	주어진 파일에 키와 감사 로그를 유지합니다(비어 있으면 Storage.FileName으로 폴백됨).
LoadFromFile(FileName)	파일에서 키와 감사 로그를 복원합니다.
ExportKeys(Stream)	키 및 감사 로그의 스트림 기반 내보내기.
ImportKeys(Stream)	키 및 audit log의 stream 기반 가져오기입니다.

PurgeExpired	ExpiresAt이 과거인 모든 키를 제거합니다.
Count	현재 저장된 키의 수를 반환합니다.
IsRequestAuthorized(Headers, QueryString, IP, RequiredScope)	일회성 권한 부여 검사: 헤더 또는 쿼리 문자열에서 키를 추출하고, 이를 검증하며, 선택적으로 범위를 적용합니다.
ExtractKeyFromHeaders(Headers)	원시 HTTP 헤더 블록에서 Validation.HeaderName 값을 파싱합니다.
ExtractKeyFromQuery(QueryString)	원시 쿼리 문자열에서 Validation.QueryParamName 값을 파싱합니다.
IsConnectionAllowed(IP)	서버 혹은 FailClosed가 True이고 IPAllowlist가 비어 있지 않으며 IP가 목록에 없을 때 False를 반환합니다. 자동으로 호출됩니다.
IsMessageAllowed(IP, Message)	모든 인바운드 메시지에 대해 자동으로 호출되는 서버 후크입니다. 기본적으로 IP 수준에서 아무 작업도 수행하지 않습니다.
RegisterConnection(IP)	새 연결을 추적합니다. 서버에 의해 자동으로 호출됩니다.
UnregisterConnection(IP)	연결에 대한 추적을 해제합니다. 연결 해제 시 자동으로 호출됩니다.

자동 통합

서버의 APIKeyManager 속성에 할당되면, 구성 요소는 자동으로:

- **새 연결 시:** 서버는 FailClosed가 True일 때 선택적 IPAllowlist를 적용하는 IsConnectionAllowed(IP)를 호출합니다 (그렇지 않으면 no-op). 라이브 연결을 추적하기 위해 RegisterConnection(IP)가 호출됩니다.
- **메시지 수신 시:** 서버는 IsMessageAllowed(IP, Message)를 호출하는데, 허용 목록이 IP를 거부하지 않는 한 이 역시 IP 수준의 no-op입니다. 이 후크는 HTTP 헤더를 검사하지 않으므로, 여기서 API 키 인증은 수행되지 않습니다.
- **요청 권한 부여(사용자가 연결해야 함):** 서버의 OnConnect 이벤트에서 *IsRequestAuthorized(Connection.HeadersRequest.Text, Connection.URL, Connection.IP, RequiredScope)*를 호출하십시오. 이것이 원시 HTTP 헤더를 사용할 수 있는 지점이므로, 여기서 X-API-Key 헤더가 추출, 검증되고 선택적으로 범위가 확인됩니다. 함수가 False를 반환하면 연결을 끊으십시오.
- **클라이언트 측:** 서버가 HTTP 업그레이드 중에 읽을 수 있도록 클라이언트의 OnHandshake / OnBeforeConnect 이벤트에서 X-API-Key 헤더(또는 구성된 HeaderName)를 설정하십시오.
- **연결 해제 시:** 내부 추적을 해제하기 위해 UnregisterConnection(IP)이 호출됩니다.

전체 예제

```
// 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");

```

스레드 안전성

API 키 관리자는 완전히 스레드 안전합니다. 모든 public 메서드는 내부 critical section과 스레드 안전 목록을 사용하여 key store, scope index, owner index, audit log에 대한 동시 액세스를 보호합니다. 단일 구성 요소는 여러 서버 인스턴스가 공유할 수 있으며 외부 동기화 없이 모든 스레드(서버 이벤트 핸들러, timer 스레드, 백그라운드 워커)에서 액세스할 수 있습니다.

참조

- 속성 — 구성 요소에서 사용 가능한 10개의 속성.
- 메서드 — 26개의 public 메서드.
- 이벤트 — 처리할 수 있는 7개의 이벤트.

TsgcWebSocketLoadBalancerServer

TsgcWebSocketLoadBalancerServer 구성 요소.

소개

TsgcWebSocketLoadBalancerServer 구성 요소를 사용하면 **WebSocket** 및 **HTTP** 프로토콜의 로드 밸런싱을 할 수 있습니다. WebSocket 프로토콜의 경우, 서버 그룹에 메시지를 분산하고 무작위 시퀀스 또는 최소 연결 알고리즘을 사용하여 클라이언트 연결을 분산합니다.

Load Balancer Server는 **TsgcWebSocketHTTPServer**의 모든 메서드와 속성을 상속합니다.

Load Balancer Configuration

Load Balancer 서버는 **TsgcWebSocketHTTPServer**의 자손이므로, 구성 방법을 알려면 **TsgcWebSocketHTTPServer**에 대한 문서를 읽으십시오.

또한 Load Balancer에는 LoadBalancer 속성이 있으며, 이 속성에는 다음과 같은 속성이 있습니다:

- **LoadBalancing**: 연결을 분배하는 방법을 여기에서 구성합니다
 - **lbRandom**: (기본값) 새 클라이언트가 연결을 요청할 때마다 무작위 서버를 반환합니다.
 - **lbConnections**: 새 클라이언트가 연결을 요청할 때마다 연결된 클라이언트가 가장 적은 서버를 반환합니다.
- **Protocols**: 어떤 프로토콜이 활성화되는지 구성합니다
 - **WebSocket**: true인 경우 websocket 연결이 Load Balancer Server에 의해 처리됩니다.
 - **HTTP**: true인 경우 http 연결이 Load Balancer Server에 의해 처리됩니다.

Backup Server Configuration

Backup Server(로드 밸런서 뒤의 서버)는 **TsgcWebSocketServer**, **TsgcWebSocketHTTPServer** 또는 **Dataspap Server** 일 수 있습니다.

이러한 서버에는 **LoadBalancer** 라는 속성이 있어 LoadBalancer Server와 Backup Servers 간의 연결을 구성할 수 있습니다.

- **Enabled**: 백업 서버로 사용하려면 true로 설정하십시오.
- **Host**: LoadBalancer가 위치한 호스트입니다.
- **Port**: LoadBalancer의 수신 대기 port.
- **Guid**: 이 서버를 식별하는 고유 id입니다.
- **Bindings**: 연결이 전달될 공개 주소입니다. 예: 백업 WebSocket 서버가 포트 8000에서 수신 대기하고 IP 주소가 1.1.1.1인 경우 다음을 사용하십시오: ws://1.1.1.1:8000;
- **AutoRegisterBindings**: 활성화하면 LoadBalancer Server가 백업 서버의 Bindings 속성을 사용하여 public 바인딩을 구성합니다.
- **AutoRestart**: 초 단위로, 0보다 크면 백업 서버의 load balancer 클라이언트가 x초마다 연결이 활성 상태인지 확인하는 내부 watchdog을 활성화하며, 닫혀 있으면 다시 연결을 시도합니다.

참조

- 속성 — 구성 요소에서 사용할 수 있는 18개의 속성.
- 메서드 — 9개의 public 메서드.
- 이벤트 — 처리할 수 있는 22개의 이벤트입니다.

TsgcWebSocketProxyServer

*TsgcWebSocketProxyServer*는 클라이언트 WebSocket 연결을 수신 대기하고 데이터 연결을 일반 TCP/IP 서버로 전달하는 WebSocket 서버 구성 요소를 구현합니다.

소개

*TsgcWebSocketProxyServer*는 클라이언트 WebSocket 연결을 수신 대기하고 데이터 연결을 일반 TCP/IP 서버로 전달하는 WebSocket 서버 구성 요소를 구현합니다. 이는 브라우저가 가상으로 모든 서버에 연결할 수 있게 하므로 브라우저 연결에 특히 유용합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 속성 19개입니다.
- 메서드 — 7개의 공개 메서드입니다.
- 이벤트 — 처리할 수 있는 22개의 이벤트.

TsgclWWebSocketClient

*TsgclWWebSocketClient*는 *WebSocket Server*에 연결할 수 있는 *IntraWeb WebSocket* 구성 요소를 구현합니다.

소개

*TsgclWWebSocketClient*는 *WebSocket Server*에 연결할 수 있는 *IntraWeb WebSocket* 구성 요소를 구현합니다. 이 구성 요소를 구성하려면 아래 단계를 따르십시오:

1. *TsgclWWebSocketClient* 구성 요소를 폼에 놓습니다

2. 사용 가능한 *WebSocket Server*에 연결하려면 *Host*와 *Port*(기본값 80)를 설정하십시오. *URL* 속성을 설정할 수 있으며 *Host*, *Port*, *Parameters*... 가 *URL*에서 업데이트됩니다. 예: *wss://127.0.0.1:8080/ws/* 는 다음과 같이 됩니다:

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

3. *TLS*(보안 연결)를 사용할지 여부를 선택할 수 있으며, 기본적으로는 활성화되어 있지 않습니다.

4. Set Transports allowed.

WebSockets: it will use standard *WebSocket* implementation

Emulation: 브라우저가 *WebSockets*를 지원하지 않으면 루프 *AJAX* 콜백 연결을 사용합니다

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: 읽기 전용 변수이며 연결이 활성 상태이면 *True*를 반환하고, 그렇지 않으면 *False*를 반환합니다.

JSOpen: 연결이 열렸을 때 클라이언트 측에서 실행할 JavaScript 코드를 여기에 포함할 수 있습니다.

JSClose: 연결이 닫혔을 때 클라이언트 측에서 실행할 JavaScript 코드를 여기에 포함할 수 있습니다.

JSMesssage: 클라이언트가 서버로부터 메시지를 수신했을 때 클라이언트 측에서 실행할 JavaScript 코드를 여기에 포함할 수 있습니다. Javascript 변수 "text"를 사용하여 메시지 문자열을 가져올 수 있습니다.

JSError: 오류가 발생했을 때 클라이언트 측에서 실행할 JavaScript 코드를 여기에 포함할 수 있습니다. Javascript 변수 "text"를 사용하여 오류 메시지를 가져올 수 있습니다.

참조

- 속성 — 구성 요소에서 사용 가능한 11개의 속성입니다.
- 메서드 — 4개의 public 메서드.
- 이벤트 — 처리할 수 있는 5개의 이벤트.

TsgcWSConnection

TsgcWSConnection은 클라이언트 WebSocket 연결을 위한 래퍼입니다. 서버 또는 클라이언트 이벤트에서 이 객체에 액세스할 수 있습니다.

Methods

WriteData: 클라이언트에 메시지를 보냅니다.

Close: 다른 피어에 닫기 메시지를 보냅니다. "CloseCode"를 선택적으로 지정할 수 있습니다. 기본적으로 전송되는 값은 NORMAL 닫기 코드입니다. 음수 닫기 코드를 보내면 닫기 사유가 전송되지 않습니다.

Disconnect: 서버 측에서 클라이언트 연결을 닫습니다. "CloseCode"를 선택적으로 지정할 수 있습니다.

Ping: 클라이언트에 ping을 전송합니다.

AddTCPEndOfFrame: 연결이 일반 TCP인 경우, 메시지의 끝을 정의하는 바이트를 설정할 수 있습니다. 메시지는 완전히 수신될 때까지 버퍼링됩니다.

Subscribed: 연결이 사용자 정의 채널에 구독되어 있는지 반환합니다.

Subscribe: 이 연결을 채널에 구독시킵니다. 나중에 서버 구성 요소에서 이 채널에 구독된 모든 연결로 메시지를 Broadcast할 수 있습니다.

UnSubscribe: 이 연결을 채널에서 구독 해제합니다.

Properties

Protocol: 이 연결에서 사용된 서브프로토콜을 반환합니다.

IP: Peer IP Address를 반환합니다.

Port: Peer Port를 반환합니다.

LocalIP: 호스트 IP 주소를 반환합니다.

LocalPort: Host Port를 반환합니다.

URL: 클라이언트가 요청한 URL을 반환합니다.

Guid: connection ID를 반환합니다.

HeadersRequest: Request에서 수신된 Headers 목록을 반환합니다.

HeadersResponse: Response로 전송된 Headers 목록을 반환합니다.

RecBytes: 수신된 바이트 수.

SendBytes: 전송된 바이트 수입니다.

Transport: 연결의 전송 유형을 반환합니다:

trpRFC6455: 일반 WebSocket 연결입니다.

trpHixie76: draft WebSocket 사양을 사용하는 WebSocket 연결입니다.

trpFlash: Flash를 FallBack으로 사용하는 WebSocket 연결입니다.

trpSSE: Server-Sent Events 연결.

trpTCP: 일반 TCP 연결입니다.

TCPEndOfFrameScanBuffer: 메시지의 끝을 찾는 데 사용할 방법을 정의할 수 있습니다(trpTCP를 전송으로 사용하는 경우).

eofScanNone: 새 패킷이 도착할 때마다 OnBinary 이벤트가 호출됩니다.

eofScanLatestBytes: 최신 바이트가 AddTCPEndOfFrame 메서드로 추가된 바이트와 같으면 OnBinary 메시지가 호출됩니다. 그렇지 않으면 이 패킷이 버퍼링됩니다.

eofScanAllBytes: AddTCPEndOfFrame 메서드로 추가된 바이트와 동일한 바이트를 전체 패킷에서 검색합니다. 발견되면 OnBinary 이벤트가 호출되고, 그렇지 않으면 이 패킷은 버퍼링됩니다.

Data: 사용자 세션 데이터 객체입니다. 여기에 객체를 전달하고 필요할 때마다 액세스할 수 있습니다. 예: 데이터베이스 연결, 사용자 세션 속성 등을 전달할 수 있습니다...

프로토콜

WebSocket을 사용하면 서브프로토콜을 구현하여 맞춤형 통신을 만들 수 있습니다. 예를 들어, WebSocket 프로토콜 위에 서브프로토콜을 구현하여 JSON 메시지를 사용하는 맞춤형 애플리케이션과 통신할 수 있으며, XML 메시지를 사용하는 또 다른 서브프로토콜을 구현할 수 있습니다.

서버 측에서 연결이 열리면 클라이언트가 보낸 서브프로토콜이 서버에서 지원되는지 검증합니다. 그렇지 않으면 연결을 닫습니다. 서버는 여러 서브프로토콜을 구현할 수 있지만 단일 연결에서는 하나만 사용할 수 있습니다.

서브프로토콜은 사용자 지정 애플리케이션을 만들고 모든 클라이언트가 동일한 통신 인터페이스를 지원하도록 보장하는 데 매우 유용합니다.

프로토콜 이름은 임의적이지만, "dataset.esegce.com"과 같은 고유한 이름을 사용하는 것이 권장됩니다.

sgcWebSockets 패키지를 사용하면 자체 프로토콜을 빌드할 수 있으며 제공된 내장 서브프로토콜을 사용할 수 있습니다:

1. Protocol MQTT: MQTT는 클라이언트 서버 게시/구독 메시징 전송 프로토콜입니다. 경량이고, 개방적이며, 단순하고, 구현하기 쉽도록 설계되었습니다.

2. Protocol AppRTC: Google과 Mozilla가 개발한 webrtc 데모 애플리케이션으로, 두 브라우저가 WebRTC API를 사용하여 서로 "대화"할 수 있게 합니다.

3. Protocol WebRTC: 웹에 실시간 통신(RTC) 기능을 가능하게 하는 것을 목표로 하는 오픈 소스 프로젝트입니다.

4. Protocol Files: 바이너리 메시지를 사용하여 구현되며, 파일 전송 지원을 제공합니다: 패킷 크기, 권한 부여, QoS, 메시지 확인 등.

5. Protocol SGC: JSON-RPC 2.0 메시지를 사용하여 구현되며, 다음 패턴을 제공합니다: RPC, PubSub, 트랜잭션 메시지, 메시지 확인 등.

6. Protocol Dataset: Default Protocol에서 상속하며, 서버에서 클라이언트로 데이터셋 변경 사항(새 레코드, 레코드 저장 또는 레코드 삭제)을 보낼 수 있습니다.

7. Protocol Presence: 채널에 누가 구독했는지 알 수 있게 합니다. 예: 채팅방, 문서의 협업자, 동일한 웹 페이지를 보는 사람들, 게임의 경쟁자...

8. Protocol WAMP 1.0: RPC 및 PubSub의 두 가지 비동기 메시징 패턴을 제공하는 개방형 WebSocket 서브프로토콜.

9. Protocol WAMP 2.0: 두 가지 비동기 메시징 패턴(RPC 및 PubSub)을 제공하는 개방형 WebSocket 서브프로토콜입니다.

10. Protocol STOMP: STOMP는 Simple (또는 Streaming) Text Orientated Messaging Protocol입니다. STOMP는 STOMP 클라이언트가 모든 STOMP message broker와 통신할 수 있도록 상호 운용 가능한 와이어 형식을 제공하여 여러 언어, 플랫폼 및 브로커 간에 쉽고 광범위한 메시징 상호 운용성을 제공합니다.

10.1 STOMP for RabbitMQ: RabbitMQ 브로커용 클라이언트.

10.2 STOMP for ActiveMQ: ActiveMQ Broker용 클라이언트입니다.

11. Protocol AMQP: Advanced Message Queuing Protocol(AMQP 0.9.1)은 사용된 message broker 공급업체나 플랫폼에 관계없이 시스템 간 메시징 상호 운용성을 허용하는 개방형 표준 프로토콜로 만들어졌습니다.

12. Protocol AMQP1: Advanced Message Queuing Protocol(AMQP 1.0.0)은 메시지 브로커 공급업체나 사용된 플랫폼에 관계없이 시스템 간 메시징 상호 운용성을 허용하는 개방형 표준 프로토콜로 만들어졌습니다.

13. Protocol E2EE: WebSocket 연결을 통해 전송된 메시지는 애플리케이션 수준에서 종단간 암호화되므로, WebSocket 서버가 데이터를 릴레이하더라도 통신하는 클라이언트만 이를 읽을 수 있습니다.

단일 연결을 사용하여 둘 이상의 프로토콜을 사용해야 하는 경우(예: 원격 프로시저 호출을 처리하기 위해 기본 프로토콜을 사용하고 데이터베이스 연결을 처리하기 위해 Dataset 프로토콜을 사용해야 할 수 있음) 각 프로토콜 구성 요소에 "Broker"를 할당하면 모든 메시지가 이 중간 프로토콜을 사용하여 교환됩니다(이에 대한 간단한 예제는 "Tickets Demo"를 확인하십시오).

프로토콜은 런타임에 등록할 수 있습니다. **RegisterProtocol** 메서드를 호출하고 프로토콜 구성 요소를 매개변수로 전달하기만 하면 됩니다.

Javascript Reference

여기서 sgcWebSockets에서 사용되는 공통 JavaScript 라이브러리에 대한 [추가 정보](#)를 얻을 수 있습니다.

Protocols Javascript

기본적으로 Javascript sgcWebSockets는 **sgcWebSocket.js** 파일을 사용합니다.

여기에서 사용 가능한 메서드를 찾을 수 있습니다. `{%host%}` 및 `{%port%}` 변수를 필요에 따라 바꿔야 합니다. 예: sgcWebSocket 서버가 `www.example.com` 웹사이트에서 80 포트를 수신하도록 구성한 경우 sgcWebSocket.js 파일에 대한 액세스를 다음과 같이 구성해야 합니다:

```
<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>
```

sgcWebSocket에는 3개의 매개변수가 있으며, 첫 번째만 필수입니다:

```
sgcWebSocket(url, protocol, transport)
```

- **URL:** WebSocket 서버 위치입니다. 일반 WebSocket 연결에는 "ws:"를, 보안 WebSocket 연결에는 "wss:"를 사용할 수 있습니다.

```
sgcWebSocket('ws://127.0.0.1')
```

```
sgcWebSocket('wss://127.0.0.1')
```

- **Protocol:** 서버가 하나 이상의 프로토콜을 수락하는 경우, 사용할 프로토콜을 정의할 수 있습니다.

```
sgcWebSocket('ws://127.0.0.1', 'esegece.com')
```

- **Transport:** 기본적으로, 먼저 WebSocket 연결을 사용하여 연결을 시도하고 브라우저에서 구현되지 않은 경우, Server Sent Events를 전송 수단으로 시도합니다.

WebSocket이 구현되어 있으면 WebSocket을 사용하고, 그렇지 않으면 Server Sent Events를 사용하십시오:

```
sgcWebSocket('ws://127.0.0.1')
```

WebSocket만 전송 수단으로 사용:

```
sgcWebSocket('ws://127.0.0.1', '', ['websocket'])
```

전송으로 Server Sent만 사용:

```
sgcWebSocket('ws://127.0.0.1', '', ['sse'])
```

인증으로 연결 열기

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  var socket = new sgcWebSocket({ "host": "ws://{%host%}:{%port%}", "user": "admin", "password": "1234" });
</script>
```

메시지 보내기

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  var socket = new sgcWebSocket('ws://{%host%}:{%port%}');
  socket.send('Hello sgcWebSockets!');
</script>
```

수신된 메시지로 Alert 표시

```
<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>
```

OnConnect, OnDisconnect, OnError 이벤트에서 Alert 표시

```
<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>

```

Get Connection Status

```

<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
    socket.state();
</script>

```

Protocol MQTT

MQTT는 Client-Server 게시/구독 메시징 전송 프로토콜입니다. 경량이고, 개방적이며, 단순하고, 구현하기 쉽도록 설계되었습니다. 이러한 특성으로 인해 작은 코드 풋프린트가 필요하거나 네트워크 대역폭이 제한된 Machine to Machine(M2M) 통신 및 사물 인터넷(IoT) 컨텍스트와 같은 제약된 환경을 포함하여 많은 상황에서 사용하기에 이상적입니다.

이 프로토콜은 TCP/IP 또는 순서가 보장되고, 손실이 없으며, 양방향 연결을 제공하는 다른 네트워크 프로토콜을 통해 실행됩니다. 기능은 다음과 같습니다:

- 일대다 메시지 배포와 애플리케이션의 분리를 제공하는 게시/구독 메시지 패턴 사용.
- 페이로드의 내용에 구애받지 않는 메시징 전송.
- 메시지 전달을 위한 세 가지 서비스 품질:
 - "At most once", 여기서 메시지는 운영 환경의 최선의 노력에 따라 전달됩니다. 메시지 손실이 발생할 수 있습니다. 이 수준은 예를 들어 개별 측정값이 손실되어도 다음 측정값이 곧 게시되므로 문제가 되지 않는 주변 센서 데이터에 사용할 수 있습니다.
 - "At least once", 메시지가 도착하는 것은 보장되지만 중복이 발생할 수 있습니다.
 - "Exactly once", 메시지가 정확히 한 번 도착하도록 보장됩니다. 이 수준은 예를 들어 중복되거나 손실된 메시지가 잘못된 청구로 이어질 수 있는 청구 시스템에서 사용될 수 있습니다.
- 작은 전송 오버헤드와 네트워크 트래픽을 줄이기 위해 최소화된 프로토콜 교환.
- 비정상적인 연결 해제가 발생할 때 관심 있는 당사자에게 알리는 메커니즘.

Features

- 3.1.1 및 5.0 MQTT 버전을 지원합니다.
- **Publish/subscribe** 메시지 패턴은 일대다 메시지 분배와 애플리케이션의 분리를 제공합니다.
- **Acknowledgment** 전송된 메시지의 확인.
- 메시지 전달을 위한 **QoS** (Quality of Service)를 구현합니다(모든 수준: At most once, At least once 및 Exactly once)
- **Last Will Testament**.
- **보안 연결**.
- **HeartBeat** 및 **Watchdog**.
- 서버에 대한 인증.

구성 요소

[TsgcWSPClient_MQTT](#): MQTT Client Component.

Most common uses

- **연결**
 - [클라이언트 MQTT 연결](#)
 - [Mosquitto MQTT 서버에 연결](#)
 - [클라이언트 MQTT 세션](#)
 - [Client MQTT Version](#)
- **Publish & Subscribe**
 - [MQTT Publish Subscribe](#)
 - [MQTT Topics](#)
 - [MQTT Subscribe](#)
 - [MQTT Publish Message](#)
 - [MQTT Receive Messages](#)
 - [MQTT Publish and Wait Response](#)

- **Other**
 - [MQTT Clear Retained Messages](#)

TsgcWSPClient_MQTT

MQTT 구성 요소는 버전 3을 지원하는 경량의 완전한 기능을 갖춘 MQTT 클라이언트 구현을 제공합니다.

소개

MQTT 구성 요소는 버전 3.1.1 및 5.0을 지원하는 경량의 완전한 기능을 갖춘 MQTT 클라이언트 구현을 제공합니다. 구성 요소는 표준 TCP와 WebSockets 모두에서 평문 및 보안 연결을 지원합니다.

MQTT 서버에 대한 연결은 간단합니다. 이 구성 요소를 폼에 드롭하고 Client 속성을 사용하여 [TsgcWebSocketClient](#) 구성 요소를 선택해야 합니다. TsgcWebSocketClient에 host와 port를 설정하고 Active := True로 설정하여 연결하십시오.

MQTT v5.0은 (v3.1.1처럼) 하위 호환되지 않습니다. 분명히 너무 많은 새로운 것들이 도입되었으므로 기존 구현을 재검토해야 합니다.

사양에 따르면 MQTT v5.0은 핵심의 대부분을 유지하면서 MQTT에 상당수의 새로운 기능을 추가합니다.

- Clean Session 플래그 기능은 세션 상태 데이터에 대한 더 세밀한 제어를 위해 2개의 속성으로 나뉩니다: CleanStart 매개변수와 새로운 SessionExpInterval.
- Server disconnect: 연결이 닫힌 이유를 나타내기 위해 서버가 DISCONNECT를 보낼 수 있도록 허용합니다.
- 모든 응답 패킷(CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK, DISCONNECT)에는 이제 작업이 성공했거나 실패한 이유를 설명하는 reason code와 reason string이 포함됩니다.
- 향상된 인증: 상호 인증을 포함한 challenge/response 스타일 인증을 활성화하는 메커니즘을 제공합니다. 이를 통해 클라이언트와 서버 모두 지원하는 경우 SASL 스타일 인증을 사용할 수 있으며, 클라이언트가 연결 내에서 재인증할 수 있는 기능을 포함합니다.
- Request / Response 패턴은 ResponseTopic의 추가로 공식화됩니다.
- Shared Subscriptions: 구독의 부하 분산된 컨슈머를 허용하는 공유 구독 지원을 추가합니다.
- Topic Alias는 대역폭을 절약하기 위해 더 짧은 숫자 식별자로 topic 필터를 참조하도록 클라이언트와 서버 모두 전송할 수 있습니다.
- 서버는 ConnectionProperties에서 지원하는 기능을 전달할 수 있습니다.
- 서버 참조: 서버가 CONNACK 또는 DISCONNECT 시 사용할 대체 서버를 지정할 수 있게 합니다. 이는 리디렉션이나 프로비저닝을 수행하는 데 사용할 수 있습니다.
- 추가: message expiration, Receive Maximums 및 Maximum Packet Sizes, Will Delay 간격이 모두 지원됩니다.

참조

- 속성 — 구성 요소에서 사용 가능한 10개의 속성.
- 메서드 — 9개의 public 메서드.
- 이벤트 — 처리할 수 있는 13개의 이벤트.

TsgcWSPClient_MQTT | Client MQTT Connect

MQTT Server에 연결하려면 먼저 [TsgcWebSocketClient](#)와 [TsgcWSPClient_MQTT](#)를 만들어야 합니다. 그런 다음 MQTT Component를 WebSocket Client에 연결해야 합니다.

기본 사용법

websocket 프로토콜을 사용하여 Mosquitto MQTT 서버에 연결합니다. 연결 후 topic: "topic1"을 구독합니다.

```
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

MQTT는 클라이언트 연결을 식별하기 위해 **Client Identifier**가 필요합니다. 구성 요소는 자동으로 **임의의 값**을 설정하지만, 필요한 경우 직접 Client Identifier를 설정할 수 있습니다. 이렇게 하려면 **OnBeforeConnect** 이벤트를 처리하고 aClientIdentifier 매개변수에 값을 설정하기만 하면 됩니다.

```
void OnMQTTBeforeConnect(TsgcWSConnection *Connection, ref bool aCleanSession,
    ref string aClientIdentifier)
{
    aClientIdentifier = "your client id";
}
```

Authentication

일부 서버는 MQTT 연결을 **authorize**하기 위해 사용자와 비밀번호를 요구합니다. 서버에 연결하기 전에 **Authentication** 속성을 사용하여 사용자 이름과 비밀번호 값을 설정하십시오.

```
oMQTT = new TsgcWSPClient_MQTT();
oMQTT->Authentication->Enabled = true;
oMQTT->Authentication->UserName = "your user";
oMQTT->Authentication->Password = "your password";
```

TsgcWSPClient_MQTT | Connect MQTT Mosquitto

Mosquitto MQTT 서버에 연결하려면 다음 샘플 구성을 사용하십시오.

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 Plain TCP 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 이벤트에는 **aCleanSession**이라는 매개변수가 있습니다. 이 매개변수의 값이 **True**이면 클라이언트가 새 세션을 시작하려고 함을 의미하므로, 서버에 저장된 세션이 있으면 이를 폐기해야 합니다. 따라서 **OnMQTTConnect** 이벤트가 발생하면 **aSession** 매개변수는 **false**가 됩니다. 이 매개변수의 값이 **False**이고 이 클라이언트 식별자와 연결된 세션이 있으면 서버는 기존 세션이 있는 상태에서 클라이언트와의 통신을 재개해야 합니다.

따라서 클라이언트에 예기치 않은 연결 해제가 발생했고, 연결이 끊긴 위치에서 세션을 복구하려는 경우, **OnMQTTBeforeConnect**에서 **aCleanSession = True**와 **aClientIdentifier = Client ID of Session**을 설정하십시오.

Session

성공적인 연결이 이루어지면 **OnMQTTConnect** 이벤트에서 Session 매개변수의 값을 확인하십시오.

Session = true는 세션이 재개되었음을 의미합니다.

Session = false는 새 세션임을 의미합니다.

```
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)
    {
        WriteLn("Session resumed");
    }
    else
    {
        WriteLn("New Session");
    }
}
```

TsgcWSPClient_MQTT | 클라이언트 MQTT 버전

현재 MQTT Client는 다음 사양을 지원합니다:

- **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>

MQTTVersion 속성을 사용하여 MQTT Client 구성 요소가 사용할 버전을 선택할 수 있습니다.

MQTT 3.1.1: TsgcWSPClient_MQTT.Version = mqtt311

MQTT 5.0: sgcWSPClient_MQTT.Version = mqtt5

TsgcWSPClient_MQTT | MQTT Publish Subscribe

게시/구독 패턴(pub/sub라고도 함)은 기존 클라이언트-서버 아키텍처에 대한 대안을 제공합니다. 클라이언트-서버 모델에서 클라이언트는 엔드포인트와 직접 통신합니다. pub/sub 모델은 메시지를 보내는 클라이언트(게시자)를 메시지를 받는 클라이언트(구독자)로부터 분리합니다. 게시자와 구독자는 서로 직접 연락하지 않습니다. 실제로 그들은 서로의 존재조차 알지 못합니다. 그들 사이의 연결은 세 번째 구성 요소(브로커)에 의해 처리됩니다. 브로커의 역할은 들어오는 모든 메시지를 필터링하여 구독자에게 올바르게 배포하는 것입니다.

TsgcWSPClient_MQTT를 사용하면 메시지를 게시하고 Topics를 구독할 수 있습니다.

Subscribe Topic

성공적인 연결 후 "topic1" 토픽을 구독합니다.

```
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

"topic1"의 모든 구독자에게 메시지 게시

```
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

Topics

MQTT에서 topic이라는 단어는 브로커가 연결된 각 클라이언트에 대해 메시지를 필터링하는 데 사용하는 UTF-8 문자열을 나타냅니다. topic은 하나 이상의 topic 레벨로 구성됩니다. 각 topic 레벨은 슬래시(topic 레벨 구분자)로 구분됩니다

```
myHome / groundfloor / livingroom / temperature
```

메시지 큐와 비교할 때 MQTT 토픽은 매우 경량입니다. 클라이언트는 게시하거나 구독하기 전에 원하는 토픽을 생성할 필요가 없습니다. 브로커는 사전 초기화 없이 유효한 각 토픽을 수락합니다. 각 토픽은 최소 1자를 포함해야 하며 토픽 문자열은 빈 공백을 허용합니다. 토픽은 대소문자를 구분합니다.

WildCards

클라이언트가 토픽을 구독할 때, 게시된 메시지의 정확한 토픽을 구독하거나 와일드카드를 사용하여 여러 토픽을 동시에 구독할 수 있습니다. 와일드카드는 토픽 구독에만 사용할 수 있으며 메시지 게시에는 사용할 수 없습니다. 두 가지 종류의 와일드카드가 있습니다: `_single-level`과 `_multi-level`.

Single Level: +

이름에서 알 수 있듯이 단일 수준 와일드카드는 하나의 토픽 수준을 대체합니다. 더하기 기호는 토픽에서 단일 수준 와일드카드를 나타냅니다.

```
myHome / groundfloor / + / temperature
```

모든 topic은 와일드카드 대신 임의의 문자열을 포함하는 경우 단일 수준 와일드카드가 있는 topic과 일치합니다. 예를 들어 `_myhome/groundfloor/+/temperature`에 대한 구독은 다음 결과를 생성할 수 있습니다:

```
YES => myHome / groundfloor / livingroom / temperature
YES => myHome / groundfloor / kitchen / temperature
NO => myHome / groundfloor / livingroom / brightness
아니요 => myHome / firstfloor / livingroom / temperature
NO => myHome / groundfloor / kitchen / fridge / temperature
```

Multi Level:

다중 수준 와일드카드는 많은 토픽 수준을 다룹니다. 해시 기호는 토픽에서 다중 수준 와일드카드를 나타냅니다. 브로커가 일치하는 토픽을 결정하려면 다중 수준 와일드카드를 토픽의 마지막 문자로 배치하고 앞에 슬래시를 붙여야 합니다.

```
myHome / groundfloor / #
```

```
YES => myHome / groundfloor / livingroom / temperature
YES => myHome / groundfloor / kitchen / temperature
YES => myHome / groundfloor / kitchen / brightness
NO => myHome / firstfloor / kitchen / temperature
```

클라이언트가 멀티 레벨 와일드카드로 토픽을 구독하면, 토픽이 얼마나 길거나 깊은 관계없이 와일드카드 문자 앞의 패턴으로 시작하는 토픽의 모든 메시지를 수신합니다. 멀티 레벨 와일드카드만 토픽으로 지정하면 (`_#`), MQTT 브로커로 전송되는 모든 메시지를 수신합니다.

TsgcWSPClient_MQTT | MQTT Subscribe

TsgcWSPClient_MQTT의 Subscribe 메서드를 사용하여 Topic을 구독할 수 있습니다. 이 메서드에는 다음 매개변수가 있습니다:

Topic: 구독할 토픽의 이름입니다.

QoS: 3가지 QoS 레벨 중 하나입니다(모든 브로커가 3가지 레벨을 모두 지원하는 것은 아닙니다). 지정되지 않은 경우 mtqsAtMostOnce를 사용합니다. [QoS Levels](#)에 대해 자세히 알아보십시오.

SubscribeProperties: MQTT 5.0인 경우 구독에 대한 추가 속성입니다.

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

Publish 메서드를 사용하여 Topic의 모든 구독자에게 메시지를 게시할 수 있으며, 다음 매개변수가 있습니다:

Topic: 메시지가 게시될 토픽의 이름입니다.

Text: 메시지의 텍스트입니다.

QoS: 3가지 QoS 레벨 중 하나입니다(모든 브로커가 3가지 레벨을 모두 지원하는 것은 아닙니다). 지정되지 않은 경우 `mtqsAtMostOnce`를 사용합니다. [QoS Levels](#)에 대해 자세히 알아보십시오.

Retain: `true`인 경우, 이 메시지는 보존됩니다. 그리고 새 클라이언트가 이 토픽을 구독할 때마다 이 메시지가 해당 클라이언트로 전송됩니다.

PublishProperties: MQTT 5.0인 경우 메시지의 속성입니다.

간단한 메시지 게시

```
MQTT->Publish("topic1", "Hello Subscribers topic1");
```

Publish QoS = At Least Once

```
MQTT->Publish("topic1", "Hello Subscribers topic1", mtqsAtLeastOnce);
```

Publish Retained message

```
MQTT->Publish("topic1", "Hello Subscribers topic1", mtqsAtMostOnce, true);
```


TsgcWSPClient_MQTT | MQTT Receive Messages

서버가 보낸 메시지는 **OnMQTTPublish** 이벤트에서 수신됩니다. 이 이벤트에는 다음 매개변수가 있습니다:

Topic: 이 메시지와 연결된 topic의 이름입니다.

Text: 메시지의 텍스트입니다.

PublishProperties: MQTT 5.0인 경우, 게시된 메시지의 속성입니다.

Read published Messages

```
void OnMQTTPublish(TsgcWSConnection *Connection, string aTopic, string aText,
    TsgcWSMQTTPublishProperties *PublishProperties)
{
    WriteLine("Topic: " + aTopic + ". Message: " + aText);
}
```

TsgcWSPClient_MQTT | MQTT Receive Messages (Extended)

OnMQTTPublishEx 이벤트는 **TsgcWSMQTTPublishData** 객체를 통해 게시된 메시지 페이로드를 여러 형식으로 제공합니다. 이 이벤트에는 다음 매개변수가 있습니다:

Topic: 이 메시지와 연결된 topic의 이름입니다.

Data: 게시된 메시지의 페이로드를 포함합니다. 다음 속성을 가집니다:

- **Value:** 문자열 형태의 페이로드.
- **Bytes:** TBytes 형식의 원시 페이로드입니다.
- **Stream:** TMemoryStream 형식의 원시 페이로드입니다.

PublishProperties: MQTT 5.0인 경우, 게시된 메시지의 속성입니다.

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 | 게시 및 응답 대기

MQTT 클라이언트는 일부 유형의 QoS 수준을 사용할 수 있으며, 각 수준은 메시지가 예상대로 처리되었는지 확인하기 위해 다른 수준에서 작동합니다.

다음 QoS 수준이 있습니다:

- **mtqsAtMostOnce:** (기본값) 메시지는 기본 TCP/IP 네트워크의 최선의 노력에 따라 전달됩니다. 응답이 예상되지 않으며 프로토콜에 재시도 의미가 정의되어 있지 않습니다. 메시지는 서버에 한 번 도착하거나 전혀 도착하지 않습니다.
- **mtqsAtLeastOnce:** 서버의 메시지 수신은 ACKNOWLEDGMENT 메시지로 확인됩니다. 통신 링크 또는 송신 디바이스에서 식별된 실패가 있거나 acknowledgement 메시지가 지정된 시간 후에 수신되지 않으면 송신자가 메시지를 다시 보냅니다. 메시지는 서버에 적어도 한 번 도착합니다. QoS level 1의 메시지에는 메시지에 ID 매개변수가 있습니다.
- **mtqsExactlyOnce:** 메시지가 정확히 한 번 도착하도록 보장됩니다. 이 레벨은 예를 들어 중복 또는 손실된 메시지가 잘못된 청구로 이어질 수 있는 청구 시스템에 사용될 수 있습니다. 통신 링크나 송신 장치 중 하나에 식별된 장애가 있거나 지정된 시간 후에도 확인 메시지가 수신되지 않으면, 송신자는 메시지를 다시 전송합니다.

OnPubAck 또는 OnPubComp 이벤트를 처리하여 메시지가 서버에서 처리되었는지 알 수 있으며, 또는 **PublishAndWait** 메서드를 사용하여 메시지가 서버에서 처리되었는지 알 수 있습니다.

PublishAndWait의 사용법은 일반 Publish 메서드와 동일하지만, 이제 Timeout이라는 새 매개변수가 있어 일정 시간이 지난 후에도 서버로부터 응답이 없으면 메서드가 false를 반환합니다. 기본적으로 이 값은 10초입니다.

```
if mqtt->PublishAndWait("topic", "text")
{
    ShowMessage("Message processed")
}
else
{
    ShowMessage("Message error");
}
```

TsgcWSPClient_MQTT | MQTT Clear Retained Messages

기본적으로 모든 MQTT 토픽은 보관 메시지를 가질 수 있습니다. 보관 메시지를 정리하는 표준 MQTT 메커니즘은 토픽에 빈 페이로드가 있는 보관 메시지를 보내는 것입니다. 이렇게 하면 보관 메시지가 제거됩니다.

```
MQTT->Publish("topic1", "", mqttAtMostOnce, true);
```

Protocol AMQP 0.9.1

Advanced Message Queuing Protocol(AMQP)은 메시지 지향 미들웨어를 위한 개방형 표준 애플리케이션 계층 프로토콜입니다. AMQP의 정의적 특징은 메시지 지향성, 큐잉, 라우팅(point-to-point 및 publish-and-subscribe 포함), 신뢰성 및 보안입니다.

AMQP는 다양한 메시징 애플리케이션과 통신 패턴을 효율적으로 지원하도록 설계된 바이너리 애플리케이션 계층 프로토콜입니다. 흐름 제어되는 메시지 지향 통신을 제공하며 at-most-once(각 메시지가 한 번 전달되거나 전혀 전달되지 않음), at-least-once(각 메시지가 반드시 전달되지만 여러 번 전달될 수 있음), exactly-once(메시지가 항상 반드시 도착하고 한 번만 도착함)와 같은 메시지 전달 보장과 SASL 및/또는 TLS 기반의 인증 및/또는 암호화를 제공합니다. Transmission Control Protocol(TCP)과 같은 기본 신뢰성 있는 전송 계층 프로토콜을 가정합니다.

Features

AMQP는 클라이언트와 브로커 간에 고품질의 안전한 메시지 전달이 필요한 모든 상황에서 사용할 수 있습니다.

AMQP는 다음 기능을 제공합니다.

- 모니터링 및 업데이트 공유.
- 요청에 대한 서버의 빠른 응답을 보장하고 시간이 많이 걸리는 작업을 추가 처리를 위해 전송합니다.
- 메시지를 여러 수신자에게 배포합니다.
- 추가 데이터 검색을 위한 Connection offline 클라이언트.
- 애플리케이션의 안정성과 원활한 작동을 향상시킵니다.
- 메시지 전달의 신뢰성입니다.
- 고속 메시지 전달.
- 메시지 수락입니다.

구성 요소

TsgcWSPClient_AMQP: AMQP 0.9.1 프로토콜을 구현하는 클라이언트 구성 요소입니다.

Most common uses

- 연결
 - [Client AMQP Connect](#)
 - [Client AMQP Disconnect](#)
- Commands
 - [AMQP Channels](#)
 - [AMQP Exchanges](#)
 - [AMQP Queues](#)
 - [AMQP Publish Messages](#)
 - [AMQP Consume Messages](#) (비동기)
 - [AMQP Get Messages](#) (동기)
 - [AMQP QoS](#)
 - [AMQP Transactions](#)

TsgcWSPClient_AMQP

AMQP 0.

소개

TsgcWSPClient_AMQP 클라이언트는 OASIS 사양을 따르는 전체 **AMQP 0.9.1** 프로토콜을 구현합니다. 클라이언트는 Plain TCP 및 WebSocket 연결을 지원하며, TLS(보안) 연결도 지원합니다.

연결

AMQP 0.9.1 프로토콜은 channel 개념을 정의하며, 이를 통해 단일 소켓 연결을 여러 가상 채널과 공유할 수 있습니다. 클라이언트는 수신된 바이트를 읽고 모든 메시지를 올바른 채널(이미 자체 스레드에서 실행 중)로 디스패치하는 내부 스레드를 구현합니다. 따라서 5개의 채널로 AMQP 연결을 실행하는 경우 클라이언트는 6개의 스레드를 실행합니다(모든 채널의 데이터를 처리하는 5개의 스레드와 연결의 데이터를 처리하는 1개의 스레드).

AMQP 서버에 연결하기 전에 AMQP 프로토콜의 다음 속성을 구성하십시오.

- **AMQPOptions.Locale:** 사용할 메시지 로케일이며, 협상된 값이므로 서버가 지원하는 로케일과 비교하여 변경될 수 있습니다. 기본값은 "en_US"입니다.
- **AMQPOptions.MaxChannels:** 열 수 있는 최대 채널 수입니다. 협상된 값이므로 서버 구성과 비교할 때 변경될 수 있습니다. 기본값은 65535입니다.
- **AMQPOptions.MaxFrameSize:** AMQP 프레임의 최대 크기(바이트)이며, 협상된 값이므로 서버 구성과 비교할 때 변경될 수 있습니다. 기본값은 2147483647입니다.
- **AMQPOptions.VirtualHost:** virtual host의 이름입니다. 기본값은 "/"입니다.

AMQP HeartBeat도 서버에 연결하기 전에 구성할 수 있으며, 하트비트 사용을 활성화하거나 비활성화할 수 있습니다.

- **HeartBeat.Enabled:** 클라이언트가 HeartBeat를 지원하는 경우 true로 설정하십시오.
- **HeartBeat.Interval:** 원하는 간격(초)입니다.

AMQP 클라이언트가 구성되면, **TsgcWebSocketClient**에 연결하고 이제 AMQP Server에 연결하기 위한 서버 연결 속성을 구성할 수 있습니다.

WebSocket 연결 대신 Plain TCP 연결을 사용하는 경우 속성 값 **Specifications.RFC6455**를 false로 설정하십시오.

```
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;
```

Channels

AMQP 클라이언트가 연결되면 첫 번째 채널을 열 수 있습니다.

```
oAMQP->OpenChannel("channel_name");
```

Exchanges

Channel이 열리면 클라이언트는 새 exchange를 선언하거나, 존재 여부를 확인할 수 있습니다... 새 exchange를 선언하려면 **DeclareExchange** 메서드를 사용하십시오.

```
oAMQP->DeclareExchange("channel_name", "exchange_name");
```

Queues

Channel이 열리면, 클라이언트는 새 큐를 선언하고 존재 여부를 확인하는 등의 작업을 할 수 있습니다... 새 Queue를 선언하려면 **DeclareQueue** 메서드를 사용하십시오. 큐는 서버에서 기본적으로 제공되지 않으므로(익스체인지와 달리), 새 큐를 선언하는 것이 항상 필요합니다(다른 클라이언트가 이미 큐를 생성한 경우 제외).

```
oAMQP->DeclareQueue("channel_name", "queue_name");
```

Binding Queues

Exchange와 Queue가 구성되면 큐를 익스체인지에 바인딩해야 할 수 있습니다. 이렇게 하면 익스체인지가 어떤 메시지를 큐로 디스패치할지 알 수 있습니다.

AMQP 서버는 큐 이름을 라우팅 키로 사용하여 큐를 "direct" 익스체인지에 자동으로 바인딩합니다. 이를 통해 바인딩을 선언할 필요 없이 특정 큐에 메시지를 보낼 수 있습니다(PublishMessage 메서드를 호출하고 Exchange 인수를 빈 값으로, 큐 이름을 RoutingKey 인수에 전달하기만 하면 됨).

```
oAMQP->BindQueue("channel_name", "queue_name", "exchange_name", "routing_key");
```

Send Messages

새 AMQP 메시지를 게시하려면 **PublishMessage** 메서드를 호출하십시오. 이 메서드를 사용하면 **String** 또는 **TStream** 메시지를 게시할 수 있습니다.

```
oAMQP->PublishMessage("channel_name", "exchange_name", "routing_key", "Hello from sgcWebSockets!!!");
```

메시지 수신

AMQP를 사용하면 2가지 모드로 메시지를 수신할 수 있습니다:

- **클라이언트 요청:** **GetMessage** 메서드를 사용합니다. 큐에 메시지가 없으면 **OnAMQPBasicGetEmpty** 이벤트가 호출됩니다.
- **서버에 의한 푸시:** Consume 메서드를 사용합니다.

Request By Client

```
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->AsString);
}
```

서버에 의해 푸시됨

```
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
}
```

참조

- 속성 — 구성 요소에서 사용 가능한 6개의 속성.
- 메서드 — 공개 메서드 41개입니다.
- 이벤트 — 처리할 수 있는 25개의 이벤트입니다.

Connection | Client AMQP Connect

AMQP 서버에 연결하려면 먼저 [TsgcWebSocketClient](#)와 [TsgcWSPClient_AMQP](#)를 생성해야 합니다. 그런 다음 AMQP 구성 요소를 WebSocket 클라이언트에 연결해야 합니다.

기본 사용법

인증 없이 AMQP 서버에 연결합니다. AMQPOptions 속성 값과 가상 호스트를 정의한 다음 TsgcWebSocketClient에 서버의 Host와 Port를 설정하십시오.

TCP Plain 연결을 사용하는 경우, TsgcWebSocketClient 속성 Specifications.RFC6455를 false로 설정하십시오.

```
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;
```

Authentication

서버에 인증이 필요한 경우, **OnAMQPAuthentication** 이벤트를 사용하여 인증 메커니즘(필요한 경우)을 선택하고 User / Password를 설정하십시오.

```
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

클라이언트는 다음 메서드를 사용하여 현재 활성 연결을 끊을 수 있습니다:

Close Reason 보내기

AMQP 클라이언트는 연결이 종료될 것임을 서버에 알리고 연결을 종료하는 이유에 대한 정보를 제공할 수 있습니다. 서버에 연결 종료를 요청하려면 Close 메서드를 사용하십시오.

```
oAMQP.Close(541, "Internal Error");
```

소켓 연결 닫기

[TsgcWebSocketClient](#)의 Active 속성을 False로 설정하기만 하면 됩니다. [연결 종료](#)에 대해 자세히 읽을 수 있습니다.

Commands | AMQP Channels

AMQP는 다중 채널 프로토콜입니다. 채널은 무거운 TCP/IP 연결을 여러 개의 가벼운 연결로 다중화하는 방법을 제공합니다. 이로 인해 포트 사용이 예측 가능해지므로 프로토콜이 더욱 "방화벽 친화적"이 됩니다. 또한 트래픽 셰이핑 및 기타 네트워크 QoS 기능을 쉽게 사용할 수 있음을 의미합니다.

모든 채널은 자체 스레드에서 실행되므로, 새 메시지가 수신될 때마다 먼저 클라이언트가 채널을 식별하고 메시지를 채널 스레드가 처리하는 큐에 넣습니다.

채널 수명 주기는 다음과 같습니다:

1. 클라이언트가 새 채널을 엽니다(Open).
2. 서버가 새 채널이 준비되었음을 확인합니다(Open-Ok).
3. 클라이언트와 서버는 필요에 따라 채널을 사용합니다.
4. 한 피어(클라이언트 또는 서버)가 채널을 닫습니다(Close).
5. 다른 피어가 채널 닫기를 핸드셰이크합니다(Close-Ok).

Open Channel

새 채널을 생성하려면 **OpenChannel** 메서드를 호출하고 채널 이름을 인수로 전달하면 됩니다. 채널이 열렸다는 서버가 보낸 확인으로 **OnAMQPChannelOpen** 이벤트가 발생합니다.

```
AMQP->OpenChannel("channel_name");

private void OnAMQPChannelOpen(TObject *Sender, const string aChannel)
{
    DoLog("#AMQP_channel_open: " + aChannel);
}
```

OpenChannelEx 메서드를 호출하여 동기 호출을 수행할 수도 있습니다. 이 메서드는 채널이 열렸으면 true를 반환하고 서버로부터 확인이 도착하지 않으면 false를 반환합니다.

```
if AMQP->OpenChannelEx("channel_name")
{
    DoLog("#AMQP_channel_open channel_name");
}
else
{
    DoLog("#AMQP_channel_open_error");
}
```

채널 닫기

기존 채널을 닫으려면 **CloseChannel** 메서드를 호출하고 채널 이름을 인수로 전달하십시오. 클라이언트가 채널이 닫혔다는 확인을 받으면 **OnAMQPChannelClose** 이벤트가 호출됩니다.

동기 호출은 **CloseChannelEx** 메서드를 호출하여 수행할 수 있습니다. 이 메서드는 채널이 닫혔으면 true를 반환하고 서버로부터 확인이 도착하지 않았으면 false를 반환합니다.

Channel Flow

플로 제어는 피어로부터의 메시지 흐름을 중단하는 데 사용되는 비상 절차입니다. 클라이언트와 서버 간에 동일한 방식으로 작동하며 **EnableChannel** / **DisableChannel** 명령으로 구현됩니다. 플로 제어는 과도하게 생성하는 게시자를 멈출 수 있는 유일한 메커니즘입니다.

채널의 Flow를 비활성화하려면 **DisableChannel** 메서드를 호출하십시오. 채널 flow가 비활성화되었다는 확인을 클라이언트가 수신하면 **OnAMQPChannelFlow** 이벤트가 호출됩니다.

채널의 흐름을 활성화할 때도 동일하게 적용되며, **EnableChannel** 메서드를 호출하면, 클라이언트가 채널 흐름이 활성화되었다는 확인을 수신할 때 **OnAMQPChannelFlow** 이벤트가 호출됩니다.

동기 요청은 **EnableChannelEx**와 **DisableChannelEx** 함수를 통해 사용할 수 있습니다.

Commands | AMQP Exchanges

exchange 클래스를 사용하면 애플리케이션이 서버의 exchange를 관리할 수 있습니다. 이 클래스를 사용하면 애플리케이션이 (어떤 구성 인터페이스에 의존하기보다는) 자체 연결을 스크립팅할 수 있습니다. 참고: 대부분의 애플리케이션은 이 수준의 정교함이 필요하지 않으며, 레거시 미들웨어가 이 의미론을 지원할 수 있을 가능성은 낮습니다.

익스체인지 수명 주기는 다음과 같습니다:

1. 클라이언트는 서버에 exchange가 존재하는지 확인하도록 요청합니다(Declare). 클라이언트는 이를 "존재하지 않으면 exchange를 생성하라" 또는 "존재하지 않으면 경고하되 생성하지 말라"로 세분화할 수 있습니다.
2. 클라이언트가 익스체인지에 메시지를 게시합니다.
3. 클라이언트는 교환을 삭제하도록 선택할 수 있습니다>Delete).

Declare Exchange

이 메서드는 새 exchange를 생성하거나 Exchange가 이미 존재하는지 확인합니다. 메서드에는 다음 인수가 있습니다:

- **ChannelName:** 채널의 이름입니다(이 메서드를 호출하기 전에 열려 있어야 함).
- **ExchangeName:** 익스체인지의 이름입니다. 255자를 넘지 않아야 하며 "amq."로 시작해서는 안 됩니다(passive 매개변수가 true인 경우 제외).
- **ExchangeType:** 익스체인지 유형입니다. 모든 AMQP 서버는 "direct" 및 "fanout" 익스체인지를 지원합니다. 지원되는 익스체인지 유형을 알아보려면 서버 설명서를 확인하십시오.
- **Passive:** passive가 true이면 서버는 exchange가 이미 선언되었지만 확인합니다. passive가 false이고 exchange가 존재하지 않으면 서버는 새로 생성합니다.
- **Durable:** true인 경우, 서버가 시작될 때 exchange가 다시 생성됩니다. false인 경우, 서버가 중지될 때 exchange가 삭제됩니다.
- **AutoDelete:** true이면 모든 큐가 바인딩 해제되면 익스체인지가 삭제됩니다.
- **Internal:** 항상 false입니다.
- **NoWait:** true인 경우 서버는 클라이언트에 확인 응답을 보내지 않습니다.
- **Arguments:** 사용자 지정 인수를 포함하는 문자열입니다. 값은 json 문자열로 전달되어야 합니다. 예: {"x-dead-letter-exchange":"my-dlx"}.

새 Exchange를 Declare하려면 **DeclareExchange** 메서드를 호출하고 채널 이름, 교환 이름 및 교환 유형을 인수로 전달하십시오. 교환이 선언되었음을 서버가 보낸 확인으로 **OnAMQPExchangeDeclare** 이벤트가 발생합니다.

```
AMQP->DeclareExchange("channel_name", "exchange_name", "direct");

private void OnAMQPExchangeDeclare(TObject *Sender, const string aChannel, const string aExchange)
{
    DoLog("#AMQP_exchange_declare: [" + aChannel + "] " + aExchange);
}
```

DeclareExchangeEx 메서드를 호출하여 동기식 호출도 수행할 수 있습니다. 이 메서드는 Exchange가 선언되면 true를, 서버로부터 확인이 도착하지 않으면 false를 반환합니다.

```
if AMQP->DeclareExchangeEx("channel_name", "exchange_name", "direct")
{
    DoLog("#AMQP_exchange_declare: [" + aChannel + "] " + aExchange);
}
else
{
    DoLog("#AMQP_exchange_declare_error");
}
```

Delete Exchange

이 메서드는 기존 Exchange를 삭제하는 데 사용됩니다. 이 메서드에는 다음 인수가 있습니다:

- **ChannelName:** 채널의 이름입니다(이 메서드를 호출하기 전에 열려 있어야 함).

- **ExchangeName:** 익스체인지의 이름입니다. 255자를 넘지 않아야 하며 "amq."로 시작해서는 안 됩니다(passive 매개변수가 true인 경우 제외).
- **IfUnused:** 서버는 exchange에 바인딩된 큐가 없는 경우에만 exchange를 삭제합니다.
- **NoWait:** true인 경우 서버는 클라이언트에 확인 응답을 보내지 않습니다.

기존 Exchange를 삭제하려면 **DeleteExchange** 메서드를 호출하고 채널 이름과 exchange 이름을 인수로 전달하십시오. exchange가 삭제되었다는 서버의 확인으로 **OnAMQPExchangeDelete** 이벤트가 발생합니다.

DeleteExchangeEx 메서드를 호출하여 동기 호출을 수행할 수도 있습니다. 이 메서드는 Exchange가 삭제되면 true를, 서버로부터 확인이 도착하지 않으면 false를 반환합니다.

Commands | AMQP Queues

큐 클래스를 사용하면 애플리케이션이 서버의 메시지 큐를 관리할 수 있습니다. 이는 메시지를 소비하는 거의 모든 애플리케이션에서 기본 단계로, 적어도 예상되는 메시지 큐가 실제로 존재하는지 확인하는 데 사용됩니다.

영구 메시지 큐의 수명 주기는 상당히 단순합니다:

1. 클라이언트는 메시지 큐가 존재함을 선언합니다(Declare, "passive" 인수 사용).
2. 서버는 메시지 큐가 존재함을 확인합니다(Declare-Ok).
3. 클라이언트는 메시지 큐에서 메시지를 읽습니다.

임시 메시지 큐의 수명 주기는 더 흥미롭습니다:

1. 클라이언트가 메시지 큐를 생성합니다(Declare, 종종 메시지 큐 이름 없이 하므로 서버가 이름을 할당함). 서버가 확인합니다(Declare-Ok).
2. 클라이언트는 메시지 큐에서 consumer를 시작합니다. consumer의 정확한 기능은 Basic 클래스에 의해 정의됩니다.
3. 클라이언트가 명시적으로 또는 채널 및/또는 연결을 닫아 consumer를 취소합니다.
4. 마지막 컨슈머가 메시지 큐에서 사라지고, 정중한 시간 초과 후 서버가 메시지 큐를 삭제합니다.

AMQP는 토픽 구독을 위한 전달 메커니즘을 메시지 큐로 구현합니다. 이를 통해 구독이 협력하는 구독자 풀 사이에서 로드 밸런싱될 수 있는 흥미로운 구조가 가능해집니다. 애플리케이션.

구독의 수명 주기에는 추가 bind 단계가 포함됩니다:

1. 클라이언트가 메시지 큐를 생성하고(Declare), 서버가 확인합니다(Declare-Ok).
2. 클라이언트가 메시지 큐를 topic exchange에 바인딩하고(Bind) 서버가 확인합니다(Bind-Ok).
3. 클라이언트는 이전 예제와 같이 메시지 큐를 사용합니다.

Declare Queue

이 메서드는 새 큐를 생성하거나 Queue가 이미 존재하는지 확인합니다. 메서드는 다음 인수를 가집니다:

- **ChannelName:** 채널의 이름입니다(이 메서드를 호출하기 전에 열려 있어야 함).
- **QueueName:** 큐의 이름이며, 255자를 초과할 수 없고 "amq."로 시작하면 안 됩니다(passive 매개변수가 true인 경우 제외).
- **Passive:** passive가 true이면 서버는 큐가 이미 선언되었는지만 확인합니다. passive가 false이고 큐가 존재하지 않으면 서버가 새로 생성합니다.
- **Durable:** true인 경우 서버가 시작될 때 큐가 다시 생성됩니다. false인 경우 서버가 중지될 때 큐가 삭제됩니다.
- **Exclusive:** true인 경우 큐가 현재 연결에서만 접근됨을 의미합니다.
- **AutoDelete:** true이면 모든 consumer가 더 이상 큐를 사용하지 않을 때 큐가 삭제됩니다.
- **NoWait:** true인 경우 서버는 클라이언트에 확인 응답을 보내지 않습니다.
- **Arguments:** 사용자 지정 인수를 포함하는 문자열입니다. 값은 json 문자열로 전달되어야 합니다. 예: {"x-dead-letter-exchange": "my-dlx"}.

새 Queue를 선언하려면 **DeclareQueue** 메서드를 호출하고 채널 이름과 큐 이름을 인수로 전달하기만 하면 됩니다. **OnAMQPQueueDeclare** 이벤트는 exchange가 선언되었다는 서버가 보낸 확인으로 발생합니다.

```
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));
}
```

DeclareQueueEx 메서드를 호출하여 동기 호출도 수행할 수 있습니다. 이 메서드는 Queue가 선언되었으면 true를, 서버로부터 확인이 도착하지 않았으면 false를 반환합니다.

```
if AMQP->DeclareQueueEx("channel_name", "queue_name")
{
    DoLog("#AMQP_queue_declare: [" + aChannel + "] " + aQueue);
}
else
{
    DoLog("#AMQP_queue_declare_error");
}
```

큐 삭제

이 메서드는 기존 Queue를 삭제하는 데 사용됩니다. 이 메서드는 다음 인수를 가집니다:

- **ChannelName:** 채널의 이름입니다(이 메서드를 호출하기 전에 열려 있어야 함).
- **QueueName:** 큐의 이름이며, 255자를 초과할 수 없고 "amq."로 시작하면 안 됩니다(passive 매개변수가 true인 경우 제외).
- **IfUnused:** 서버는 큐에 연결된 컨슈머가 없는 경우에만 큐를 삭제합니다.
- **IfEmpty:** 메시지가 없을 때만 서버가 queue를 삭제합니다.
- **NoWait:** true인 경우 서버는 클라이언트에 확인 응답을 보내지 않습니다.

기존 Queue를 삭제하려면 **DeleteQueue** 메서드를 호출하고 채널 이름과 queue 이름을 인수로 전달하십시오. queue가 삭제되었다는 서버의 확인으로 **OnAMQPQueueDelete** 이벤트가 발생합니다.

DeleteQueueEx 메서드를 호출하여 동기 호출도 수행할 수 있으며, 이 메서드는 큐가 삭제되었으면 true를, 서버로부터 확인이 도착하지 않았으면 false를 반환합니다.

Bind Queue

이 메서드는 Queue를 Exchange에 바인딩하는 데 사용됩니다. Exchange는 바인딩을 사용하여 메시지를 라우팅하는 데 어떤 큐가 사용될지 알 수 있습니다.

모든 AMQP Server는 Queue Name을 binding routing key로 사용하여 모든 큐를 default exchange(이름이 없는 "direct" exchange)에 자동으로 바인딩합니다. 이를 통해 binding을 선언하지 않고도 특정 큐에 메시지를 보낼 수 있습니다. **PublishMessage** 메서드를 호출하고, Exchange Name으로 빈 값을 전달하고, **RoutingKey**를 **Queue Name** 값으로 설정하기만 하면 됩니다.

메서드에는 다음 인수가 있습니다:

- **ChannelName:** 채널의 이름입니다(이 메서드를 호출하기 전에 열려 있어야 함).
- **QueueName:** 큐의 이름이며, 255자를 초과할 수 없고 "amq."로 시작하면 안 됩니다(passive 매개변수가 true인 경우 제외).
- **ExchangeName:** 익스체인지의 이름입니다. 255자를 넘지 않아야 하며 "amq."로 시작해서는 안 됩니다(passive 매개변수가 true인 경우 제외).
- **RoutingKey:** 바인딩의 routing key입니다.
- **NoWait:** true인 경우 서버는 클라이언트에 확인 응답을 보내지 않습니다.
- **Arguments:** 사용자 지정 인수를 포함하는 문자열입니다. 값은 json 문자열로 전달되어야 합니다. 예: {"x-dead-letter-exchange":"my-dlx"}.

Queue를 Exchange에 바인딩하려면 **BindQueue** 메서드를 호출하고 채널 이름, 큐 이름, exchange 및 라우팅 키를 인수로 전달하십시오. 큐가 바인딩되었음을 서버가 보내는 확인으로 **OnAMQPQueueBind** 이벤트가 발생합니다.

```
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);
}
```

BindQueueEx 메서드를 호출하여 동기 호출도 수행할 수 있습니다. 이 메서드는 Queue가 바인딩되면 true를, 서버로부터 확인이 도착하지 않으면 false를 반환합니다.

```
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

이 메서드는 기존 큐 바인딩을 삭제합니다.

메서드에는 다음 인수가 있습니다:

- **ChannelName:** 채널의 이름입니다(이 메서드를 호출하기 전에 열려 있어야 함).
- **QueueName:** 큐의 이름이며, 255자를 초과할 수 없고 "amq."로 시작하면 안 됩니다(passive 매개변수가 true인 경우 제외).
- **ExchangeName:** 익스체인지의 이름입니다. 255자를 넘지 않아야 하며 "amq."로 시작해서는 안 됩니다(passive 매개변수가 true인 경우 제외).
- **RoutingKey:** 바인딩의 routing key입니다.

큐를 UnBind하려면 **UnBindQueue** 메서드를 호출하고 채널 이름, 큐 이름, 교환 및 라우팅 키를 인수로 전달하십시오. 큐가 바인딩 해제되었음을 서버가 보낸 확인으로 **OnAMQPQueueUnBind** 이벤트가 발생합니다.

동기 호출은 **UnBindQueueEx** 메서드를 호출하여 수행할 수도 있습니다. 이 메서드는 Queue가 언바인드되면 true를 반환하고, 서버로부터 확인이 도착하지 않으면 false를 반환합니다.

Purge Queue

이 메서드는 큐의 모든 메시지를 비웁니다. 전송되었지만 acknowledgment를 기다리는 모든 메시지는 영향을 받지 않습니다.

메서드에는 다음 인수가 있습니다:

- **ChannelName:** 채널의 이름입니다(이 메서드를 호출하기 전에 열려 있어야 함).
- **QueueName:** 큐의 이름이며, 255자를 초과할 수 없고 "amq."로 시작하면 안 됩니다(passive 매개변수가 true인 경우 제외).
- **NoWait:** true인 경우 서버는 클라이언트에 확인 응답을 보내지 않습니다.

Queue를 Purge하려면 **PurgeQueue** 메서드를 호출하고 채널 이름과 큐 이름을 인수로 전달하기만 하면 됩니다. **OnAMQPQueuePurge** 이벤트는 큐가 Purge되었다는 서버가 보낸 확인으로 발생합니다.

PurgeQueueEx 메서드를 호출하여 동기 호출도 수행할 수 있습니다. 이 메서드는 Queue가 비워지면 true를, 서버로부터 확인이 도착하지 않으면 false를 반환합니다.

Commands | AMQP Publish Messages

메시지 게시

PublishMessages 메서드는 AMQP 서버에 메시지를 보내는 데 사용됩니다.

AMQP 서버는 큐 이름을 라우팅 키로 사용하여 큐를 "direct" 익스체인지에 자동으로 바인딩합니다. 이를 통해 바인딩을 선언할 필요 없이 특정 큐에 메시지를 보낼 수 있습니다(PublishMessage 메서드를 호출하고 Exchange 인수를 빈 값으로, 큐 이름을 RoutingKey 인수에 전달하기만 하면 됨).

메서드에는 다음 인수가 있습니다:

- **ChannelName:** 채널의 이름입니다(이 메서드를 호출하기 전에 열려 있어야 함).
- **ExchangeName:** 교환의 이름으로, 255자를 초과해서는 안 되며 "amq."로 시작해서는 안 됩니다(passive 매개변수가 true인 경우 제외).
- **RoutingKey:** 바인딩의 routing key 이름입니다.
- **Mandatory:** true이고 메시지를 어떤 큐로도 라우팅할 수 없으면 서버가 메시지를 반환하고 OnAMQPBasicReturn 이벤트가 발생합니다.
- **Immediate:** true이고 메시지가 어떤 큐로도 라우팅될 수 없는 경우, 메시지는 서버에 의해 반환되고 OnAMQPBasicReturn 이벤트가 발생합니다.

```
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 + " " +
}
```

Publish Confirmations

메시지를 게시하는 동안 네트워크가 실패할 수 있으며, 메시지가 손실되지 않도록 보장하는 유일한 방법은 트랜잭션을 사용하는 것입니다. 따라서 각 메시지에 대해 **select transaction**, **send the message** 및 **commit**을 수행하십시오. 성공적인 트랜잭션의 확인은 **OnAMQPTransactionOk** 이벤트가 발생할 때 수신됩니다.

AMQP Consume Messages

소비자는 큐에서 소비합니다. 메시지를 소비하려면 큐가 있어야 합니다. 새 소비자가 추가될 때 큐에 이미 준비된 메시지가 있다고 가정하면 전달이 즉시 시작됩니다.

대상 큐는 소비자 등록 시점에 비어 있을 수 있습니다. 이 경우 새 메시지가 큐에 추가될 때 첫 전달이 발생합니다.

메시지 소비는 **비동기** 작업이며, 이는 새 메시지가 소비자 큐에 전달될 수 있을 때마다 서버에 의해 자동으로 클라이언트로 푸시됨을 의미합니다. **동기적으로 메시지 수신**하는 대체 방법을 읽을 수 있습니다.

Consume

Consume 메서드는 큐에 새 소비자를 생성하며, 새 메시지가 있을 때마다 이것이 소비자 클라이언트에게 자동으로 전달됩니다.

메서드에는 다음 인수가 있습니다:

- **ChannelName:** 채널의 이름입니다(이 메서드를 호출하기 전에 열려 있어야 함).
- **QueueName:** 큐의 이름입니다. 255자를 초과해서는 안 되며 "amq."로 시작해서는 안 됩니다(passive 매개변수가 true인 경우 제외).
- **ConsumerTag:** consumer의 이름이며 고유해야 합니다. 설정되지 않으면 서버가 새로 생성합니다.
- **NoLocal:** true이면 소비자가 동일한 채널에 게시된 메시지를 절대 소비하지 않음을 의미합니다.
- **NoAck:** true인 경우 서버가 전달된 모든 메시지에 대한 확인을 기대하지 않음을 의미합니다.
- **Exclusive:** true인 경우 다른 소비자가 이 큐에서 메시지를 소비하는 것을 방지합니다.
- **NoWait:** true인 경우 서버는 클라이언트에게 확인 응답을 보내지 않습니다.
- **Arguments:** 사용자 지정 인수를 포함하는 문자열입니다. 값은 json 문자열로 전달되어야 합니다. 예: {"x-dead-letter-exchange":"my-dlx"}.

메시지는 **OnAMQPBasicDeliver** 이벤트에서 전달됩니다.

```
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());
}
```

ConsumeEx 메서드를 호출하기만 하면 동기 호출을 수행할 수 있으며, 이 메서드는 Consumer가 생성되었으면 true를, 서버로부터 확인이 도착하지 않았으면 false를 반환합니다.

Cancel Consume

이 메서드는 기존 consumer queue를 Cancel하는 데 사용됩니다.

메서드에는 다음 인수가 있습니다:

- **ChannelName:** 채널의 이름입니다(이 메서드를 호출하기 전에 열려 있어야 함).
- **ConsumerTag:** 소비자의 이름입니다.
- **NoWait:** true인 경우 서버는 클라이언트에게 확인 응답을 보내지 않습니다.

```
AMQP->CancelConsume("channel_name", "consumer_tag");

private void OnAMQPBasicCancelConsume(TObject *Sender, const string aChannel, const string aConsumerTag)
{
}
```

```
DoLog("#AMQP_basic_cancel_consume: " + aChannel + " " + aConsumerTag);  
}
```

CancelConsumeEx 메서드를 호출하기만 하면 동기 호출을 수행할 수 있으며, 이 메서드는 Consumer가 취소되면 true를, 서버로부터 확인이 도착하지 않으면 false를 반환합니다.

Commands | AMQP Get Messages

메시지를 가져오는 것은 **Synchronous** 작업입니다. 즉, 큐에 메시지가 있는지 서버에 묻는 것은 클라이언트입니다. 메시지를 **비동기적으로 수신**하는 대체 방법을 읽을 수 있습니다.

Get Message

GetMessage 메서드는 AMQP 서버에 큐에 사용 가능한 메시지가 있는지 묻는 요청을 보냅니다. 메시지가 있으면 **OnAMQPBasicGetOk** 이벤트에서 디스패치되고, 큐가 비어 있으면 **OnAMQPBasicGetEmpty** 이벤트가 호출됩니다.

메서드에는 다음 인수가 있습니다:

- **ChannelName:** 채널의 이름입니다(이 메서드를 호출하기 전에 열려 있어야 함).
- **QueueName:** 큐의 이름이며, 255자를 초과할 수 없고 "amq."로 시작하면 안 됩니다(passive 매개변수가 true인 경우 제외).
- **NoWait:** true인 경우 서버는 클라이언트에게 확인 응답을 보내지 않습니다.

```
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);
}
```

동기 호출은 **GetMessageEx** 메서드를 호출하기만 하면 수행할 수 있습니다. 이 메서드는 큐에 사용 가능한 메시지가 있으면 true를 반환하고, 그렇지 않으면 결과가 false가 됩니다.

Commands | AMQP QoS

AMQP를 사용하면 서버가 메시지 확인을 받기 위해 대기하기 전에 클라이언트에 보내는 메시지 수를 제한하는 QoS 레벨을 설정할 수 있습니다.

Set QoS

SetQoS 메서드는 서버가 AMQP 클라이언트에 보내는 메시지 수를 제한하는 데 사용됩니다. 메서드에는 다음 인수가 있습니다:

- **ChannelName:** 채널의 이름입니다(이 메서드를 호출하기 전에 열려 있어야 함).
- **PrefetchSize:** 바이트 단위의 창 크기입니다. 이미 보낸 현재 미확인 메시지 전체 크기에 보낼 다음 메시지를 더한 것이 **PrefetchSize** 인수보다 크면 서버는 클라이언트에 메시지를 보내지 않습니다. 값이 0이면 한도가 없음을 의미합니다.
- **PrefetchCount:** 이미 전송되고 확인되지 않은 메시지의 최대 수입니다. 숫자가 더 크면 서버는 클라이언트로의 메시지 전송을 중지합니다.
- **Global:** true이면 QoS가 연결의 기존 및 새 컨슈머 모두에 적용됩니다. false이면 QoS가 채널의 기존 및 새 컨슈머 모두에 적용됩니다.

서버로부터의 응답은 **OnAMQPBasicQoS** 이벤트에서 수신됩니다.

```
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));
}
```

동기 호출은 **SetQoSEx** 메서드를 호출하기만 하면 수행할 수 있습니다. 이 메서드는 요청이 처리되면 true를 반환하고, 그렇지 않으면 결과가 false가 됩니다.

Commands | AMQP 트랜잭션

AMQP는 두 가지 종류의 트랜잭션을 지원합니다:

1. 모든 게시된 메시지와 확인이 독립형 트랜잭션으로 처리되는 자동 트랜잭션.
2. 서버 로컬 트랜잭션으로, 서버는 게시된 메시지와 확인을 버퍼링하고 클라이언트의 요청에 따라 커밋합니다.

Transaction 클래스("tx")는 애플리케이션이 두 번째 유형, 즉 서버 트랜잭션에 접근할 수 있도록 합니다. 이 클래스의 의미는 다음과 같습니다:

1. 애플리케이션은 이러한 트랜잭션을 원하는 각 채널에서 서버 트랜잭션을 요청합니다(Select).
2. 애플리케이션이 작업을 수행합니다(Publish, Ack).
3. 애플리케이션이 작업을 커밋하거나 롤백합니다(Commit, Roll-back).
4. 애플리케이션이 무한정 작동합니다.

트랜잭션은 전달이 아니라 게시된 콘텐츠와 확인을 다룹니다. 따라서 롤백은 어떤 메시지도 다시 큐에 넣거나 재전달하지 않으며, 클라이언트는 후속 트랜잭션에서 이러한 메시지를 확인할 권리가 있습니다.

Transaction 메서드를 사용하면 publish 및 ack 작업을 원자적 작업 단위로 일괄 처리할 수 있습니다. 트랜잭션 내에서 실행된 모든 publish 및 ack 요청이 성공적으로 완료되거나 그중 어느 것도 완료되지 않도록 하는 것이 의도입니다.

Start Transaction

StartTransaction 메서드는 서버에서 새 트랜잭션을 시작합니다. 클라이언트는 Commit 또는 Rollback 메서드를 사용하기 전에 채널에서 최소 한 번 이 메서드를 사용합니다. 서버가 트랜잭션 사용을 확인 응답하면 **OnAMQPTransactionOk** 이벤트가 발생합니다.

```
AMQP->StartTransaction("channel_name");
```

StartTransactionEx 메서드를 호출하여 동기 호출을 수행할 수 있습니다. 이 메서드는 요청이 처리되었으면 true를, 그렇지 않으면 결과는 false가 됩니다.

Commit Transaction

이 메서드는 현재 트랜잭션에서 수행된 모든 메시지 publish와 acknowledgment를 커밋합니다. 커밋 직후 새 트랜잭션이 시작됩니다. 서버가 트랜잭션 사용을 확인하면 **OnAMQPTransactionOk** 이벤트가 발생합니다.

```
AMQP->CommitTransaction("channel_name");
```

CommitTransactionEx 메서드를 호출하여 동기 호출을 수행할 수 있습니다. 이 메서드는 요청이 처리되었으면 true를, 그렇지 않으면 결과는 false가 됩니다.

Rollback Transaction

이 메서드는 현재 트랜잭션에서 수행된 모든 메시지 게시 및 확인 응답을 포기합니다. 롤백 직후 새 트랜잭션이 즉시 시작됩니다. 확인되지 않은 메시지는 롤백에 의해 자동으로 재전달되지 않습니다. 재전달이 필요한 경우 명시적인 recover 호출을 실행해야 합니다. 서버가 트랜잭션 사용을 확인하면 **OnAMQPTransactionOk** 이벤트가 발생합니다.

```
AMQP->RollbackTransaction("channel_name");
```

RollbackTransactionEx 메서드를 호출하지만 하면 동기 호출을 수행할 수 있습니다. 이 메서드는 요청이 처리되면 true를 반환하고, 그렇지 않으면 결과는 false가 됩니다.

Protocol Kafka

Apache Kafka는 레코드 스트림을 게시, 저장 및 소비하는 데 사용되는 오픈 소스 분산 이벤트 스트리밍 플랫폼입니다. 생산자는 topic에 메시지를 쓰고, 브로커는 이를 파티션된 추가 전용 로그에 저장하며, 소비자는 개별적으로 또는 조정된 소비자 그룹의 일부로 순서대로 다시 읽습니다.

Kafka는 TCP를 통한 바이너리 애플리케이션 계층 프로토콜을 사용합니다. 각 topic은 하나 이상의 파티션으로 분할되며, 파티션의 모든 레코드는 단조 증가하는 offset으로 식별되고, 브로커는 레코드를 보존하므로 소비자가 자신의 속도로 읽고 필요할 때 기록을 재생할 수 있습니다. 이로 인해 Kafka는 고처리량 메시징, event sourcing, 로그 집계 및 스트림 처리에 적합합니다.

Features

sgcWebSockets Kafka 클라이언트는 REST 프록시나 외부 라이브러리 없이 네이티브 Kafka 프로토콜을 통해 브로커와 직접 통신합니다.

- key, value 및 구성 가능한 확인 응답으로 임의의 topic 및 partition에 레코드를 게시(produce)합니다.
- 관리되는 소비자 그룹으로 또는 그룹 없이(모든 파티션을 직접 읽음) 레코드를 소비합니다.
- 컨슈머 그룹 조정: 코디네이터 발견, 조인/동기화, 파티션 할당 및 오프셋 커밋/폐치.
- Offset 관리: 가장 이른, 가장 늦은 및 commit된 offset을 쿼리합니다.
- 토픽 관리: 토픽 생성 및 삭제, 클러스터 메타데이터 요청, 소비자 그룹 나열.
- 현재 Kafka 브로커가 사용하는 v2 레코드 배치 형식을 읽고 씁니다.

Supported Versions

클라이언트는 **Apache Kafka 0.11**에서 도입된 v2 레코드 배치 형식을 사용하며 **Apache Kafka 3.x**에 대해 테스트되었습니다. 기본 브로커 포트는 9092입니다.

구성 요소

TsgcWSPClient_Kafka: Apache Kafka 프로토콜을 구현하는 클라이언트 구성 요소입니다.

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

Apache Kafka 클라이언트.

소개

TsgcWSPClient_Kafka 클라이언트는 **Apache Kafka** 프로토콜을 구현하며, 일반 TCP 연결에서 네이티브 Kafka 바이너리 프로토콜을 통해 브로커와 직접 통신합니다. REST 프록시나 외부 라이브러리 없이 레코드를 생성하고 소비하고, consumer group을 관리하고, topic을 관리하고, offset을 쿼리할 수 있습니다.

연결

Kafka 클라이언트는 **TsgcWebSocketClient** 위에서 실행되는 프로토콜 구성 요소입니다. Kafka는 WebSocket 프로토콜이 아니므로, 클라이언트가 일반 TCP 소켓으로 연결되도록 **Specifications.RFC6455**를 **false**로 설정하십시오. 그런 다음 클라이언트를 Kafka 구성 요소에 할당하고 브로커 **Host**와 **Port**(기본값 9092)를 설정하십시오.

프로듀서와 컨슈머의 동작은 **KafkaOptions** 속성을 통해 구성됩니다:

- **KafkaOptions.ClientId**: 애플리케이션을 브로커에 식별합니다.
- **KafkaOptions.Producer.Acks**: 필요한 확인, kafkaAcksNone, kafkaAcksLeader 또는 kafkaAcksAll.
- **KafkaOptions.Producer.TimeoutMs**: 브로커가 필요한 확인 응답을 기다리는 시간.
- **KafkaOptions.Consumer.GroupId**: 소비자 그룹 이름입니다. 그룹 없이 소비하려면 비워 두십시오.
- **KafkaOptions.Consumer.OffsetReset**: kafkaOffsetEarliest 또는 kafkaOffsetLatest로, 완전히 새로운 그룹이 읽기를 시작하는 위치입니다.

```
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;
```

메시지 생성

레코드를 게시하려면 **Produce** 메서드를 호출하십시오. 모든 게시의 결과는 레코드가 저장된 오프셋을 포함하여 **OnKafkaProduce** 이벤트를 통해 보고됩니다.

```
oKafka->KafkaOptions->Producer->Acks = kafkaAcksLeader;
oKafka->Produce("my-topic", "Hello Kafka", "key-1");
```

Consume Messages

GroupId와 **OffsetReset**를 설정하고, 하나 이상의 토픽을 구독한 다음 **Poll**을 호출하여 사용 가능한 레코드를 가져오십시오. Poll은 **TsgcKafkaMessages** 목록을 반환하고 각 레코드에 대해 **OnKafkaMessage** 이벤트도 발생시킵니다. 계속 수신하려면 (예: 타이머에서) Poll을 반복적으로 호출하십시오.

```
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();  
}
```

참조

- 속성 — 구성 요소에서 사용 가능한 3개의 속성.
- 메서드 — 18개의 public 메서드.
- 이벤트 — 처리할 수 있는 6개의 이벤트.

Connection | Client Kafka Connect

Apache Kafka 브로커에 연결하려면 먼저 [TsgcWebSocketClient](#)와 [TsgcWSPClient_Kafka](#)를 생성해야 합니다. 그런 다음 Kafka 구성 요소를 WebSocket 클라이언트에 연결하십시오.

Kafka는 일반 TCP 연결을 사용하므로 TsgcWebSocketClient 속성 Specifications.RFC6455를 false로 설정해야 합니다. 브로커 Host와 Port(기본 Kafka 포트는 9092)를 설정하고, 선택적으로 KafkaOptions.ClientId를 설정한 다음, Active를 true로 설정하여 연결을 시작하십시오.

브로커에 대한 연결이 설정되면 **OnKafkaConnect** 이벤트가 발생합니다.

기본 사용법

```
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

Apache Kafka 브로커에서 연결을 끊으려면 WebSocket Client 속성 Active를 false로 설정하십시오. 이렇게 하면 브로커에 대한 기본 TCP 연결이 닫힙니다.
연결이 닫히면 **OnKafkaDisconnect** 이벤트가 발생합니다.

기본 사용법

```
oClient->Active = false;  
  
void OnKafkaDisconnect(TObject *Sender)  
{  
    ShowMessage("Disconnected from Kafka broker");  
}
```

Commands | Kafka Produce Messages

Kafka topic에 레코드를 게시하려면 **Produce(topic, value, key, partition)** 메서드를 사용하십시오. key 및 partition 인수는 선택적입니다. partition이 설정되지 않으면 브로커가 하나를 선택합니다(예: key를 해싱하여). 바이너리 페이로드를 게시하려면 **ProduceBytes**를 호출하십시오.

KafkaOptions.Producer.Acks로 전달 보장을 구성하십시오: kafkaAcksNone(fire and forget), kafkaAcksLeader(파티션 리더 대기) 또는 kafkaAcksAll(모든 in-sync 복제본 대기). Producer.TimeoutMs와 Producer.Compression(kafkaCompressionNone 또는 kafkaCompressionGzip)도 설정할 수 있습니다.

레코드가 전송된 후, **OnKafkaProduce** 이벤트는 생성된 레코드의 topic, partition, offset 및 오류 코드를 보고합니다.

기본 사용법

```
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

레코드를 소비하려면 `KafkaOptions.Consumer.GroupId`와 `KafkaOptions.Consumer.OffsetReset(kafkaOffsetEarliest` 또는 `kafkaOffsetLatest`)을 설정한 다음, 하나 이상의 토픽으로 **Subscribe([topic])**를 호출하십시오.

구독 후 **Poll(timeoutMs)**를 호출하여 레코드를 가져오십시오. Poll은 `TsgcKafkaMessages` 목록을 반환하고 레코드당 한 번씩 **OnKafkaMessage** 이벤트를 발생시킵니다. 일반적으로 새 레코드가 도착하는 대로 계속 소비하기 위해 예를 들어 타이머에서 Poll을 반복적으로 호출합니다.

목록의 각 레코드는 `TsgcKafkaMessage`입니다. Count를 사용하여 목록을 반복하고 `GetKeyString`, `GetValueString`, `Topic`, `Partition` 및 `Offset`으로 각 레코드를 읽으십시오.

구독 및 폴링

```
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

Poll이 반환한 목록을 반복하는 대신 `OnKafkaMessage` 이벤트에서 각 레코드를 처리할 수 있습니다.

```
void OnKafkaMessage(TObject *Sender, TsgcKafkaMessage *Message)
{
    ShowMessage(Message->Topic + ": " + Message->GetKeyString() + " = " + Message->GetValueString());
}
```

Commands | Kafka Consumer Groups

소비자 그룹을 사용하면 여러 소비자가 토픽 읽기 작업을 공유할 수 있습니다. 브로커는 GroupId별로 커밋된 오프셋을 추적하고 토픽 파티션을 그룹 멤버에게 할당하므로, 각 파티션은 한 번에 한 멤버에 의해서만 읽힙니다. `KafkaOptions.Consumer.GroupId`로 그룹을 설정하십시오.

커밋된 오프셋이 없는 새 그룹의 경우 `KafkaOptions.Consumer.OffsetReset`이 시작 위치를 결정합니다: `kafkaOffsetEarliest`는 파티션의 처음부터 읽고, `kafkaOffsetLatest`는 새 레코드만 읽습니다. 오프셋이 커밋되면 마지막으로 커밋된 위치부터 소비가 재개됩니다.

마지막 Poll이 반환한 레코드의 offset을 커밋하려면 **CommitSync**를 호출하여 그룹이 다음 세션에서 그 이후부터 재개되도록 하십시오. GroupId를 비워 두면 클라이언트가 그룹에 가입하지 않고 브로커에 offset을 커밋하지 않고 모든 파티션을 직접 소비합니다.

기본 사용법

```
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

Kafka 클라이언트는 브로커에서 토픽을 관리할 수 있습니다. 주어진 수의 파티션과 복제 계수로 새 토픽을 생성하려면 **CreateTopic(name, partitions, replication)**을 사용하고, 토픽을 제거하려면 **DeleteTopic(name)**을 사용하십시오. 브로커, 토픽 및 그 파티션 레이아웃과 같은 클러스터 및 토픽 정보를 검색하려면 **GetMetadata([topics])**를 사용하십시오. 전체 클러스터에 대한 메타데이터를 요청하려면 빈 배열을 전달하십시오.

기본 사용법

```
// 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

Kafka 클라이언트는 주어진 토픽과 파티션에 대한 오프셋을 쿼리하고 설정할 수 있습니다. **GetEarliestOffset(topic, partition)**은 가장 오래된 사용 가능한 오프셋을 반환하고, **GetLatestOffset(topic, partition)**은 작성될 다음 오프셋을 반환하며, **GetCommittedOffset(topic, partition)**은 소비자 그룹이 커밋한 마지막 오프셋을 반환합니다. 세 가지 모두 `Int64`를 반환합니다.

소비자 그룹에 대해 특정 오프셋을 커밋하여 다음 세션에서 해당 위치부터 소비가 재개되도록 하려면 **CommitOffset(topic, partition, offset)**을 사용하십시오.

기본 사용법

```
__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

AMQP(Advanced Message Queuing Protocol) 1.0.0은 분산 시스템 간의 신뢰할 수 있는 비동기 통신을 위해 설계된 메시징 프로토콜입니다. 애플리케이션이나 구성 요소 간에 디커플링된 방식으로 메시지 교환을 용이하게 하여, 직접적인 종속성 없이 통신할 수 있도록 합니다. 다음은 AMQP 1.0.0의 일부 주요 측면에 대한 기술적 분석입니다:

- **메시지 지향 통신:** AMQP 1.0.0은 메시지 개념을 중심으로 합니다. 메시지는 데이터, 명령어 또는 커맨드를 전달할 수 있으며 통신의 기본 단위입니다.
- **메시지 브로커:** 이 프로토콜은 브로커 기반 메시징 모델에서 작동합니다. 서버나 중개 엔티티가 될 수 있는 브로커는 생산자와 소비자 간의 메시지 라우팅 및 전달을 관리합니다.
- **Queues and Exchanges:** Queue는 메시지가 임시로 저장되는 브로커 내의 저장 엔티티입니다. Exchange는 메시지 콘텐츠나 routing key와 같은 기준에 따라 생산자에서 큐로 메시지를 라우팅하는 규칙을 정의합니다.
- **Addresses 및 Links:** Address는 메시징 인프라 내에서 메시지 목적지를 식별합니다. Link는 특정 address와 연관된 발신자(producer)와 수신자(consumer) 간의 통신 채널입니다.
- **세션 및 연결:** 세션은 통신을 위한 논리적 채널을 나타내며, 단일 연결 내에서 여러 메시지 스트림을 허용합니다. 연결은 클라이언트 애플리케이션과 메시지 브로커 간의 전체 통신 링크를 관리합니다.
- **Security:** AMQP 1.0.0은 클라이언트와 브로커 간의 안전한 통신을 보장하기 위해 인증 및 권한 부여를 포함한 다양한 보안 메커니즘을 지원합니다.
- **전송 독립적:** 이 프로토콜은 전송 독립적으로 설계되어 TCP, TLS 또는 WebSocket과 같은 다양한 네트워크 전송을 통해 작동할 수 있으므로 배포에 유연성을 제공합니다.
- **Flow Control:** AMQP 1.0.0에는 소비자가 주어진 속도로 들어오는 메시지를 처리할 수 있는 능력을 나타낼 수 있도록 하는 흐름 제어 메커니즘이 포함되어 있습니다. 이는 많은 수의 메시지로 소비자를 압도하는 것을 방지하는 데 도움이 됩니다.
- **오류 처리:** 프로토콜은 메시지의 확인 및 거부를 포함하여 오류를 처리하는 메커니즘을 지정하여 메시지 전달의 견고성과 신뢰성을 보장합니다.
- **SASL Authentication:** Simple Authentication and Security Layer(SASL)는 클라이언트와 브로커 간 연결을 인증하고 보호하는 데 사용됩니다.

전반적으로 AMQP 1.0.0은 다양한 소프트웨어 구성 요소와 시스템이 느슨하게 결합된 방식으로 통신할 수 있는 표준화되고 상호 운용 가능한 방법을 제공하여 다양한 분산 및 엔터프라이즈급 애플리케이션에 적합합니다.

구성 요소

TsgcWSPClient_AMQP1: AMQP 1.0.0 프로토콜을 구현하는 클라이언트 구성 요소입니다.

Most common uses

- **연결**
 - [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](#)

TsgcWSPClient_AMQP1

AMQP 1.

소개

TsgcWSPClient_AMQP1 클라이언트는 OASIS 사양을 따라 **AMQP 1.0.0** 프로토콜을 구현합니다. 클라이언트는 일반 TCP 및 WebSocket 연결을 지원하며, TLS(보안) 연결도 지원됩니다.

구성

AMQP 1.0.0 클라이언트에는 연결을 구성할 수 있는 **AMQPOptions** 속성이 있습니다.

- **ChannelMax**: channel-max 값은 연결에서 사용할 수 있는 가장 높은 채널 번호입니다. 이 값에 1을 더한 값이 연결에서 동시에 활성화될 수 있는 최대 세션 수입니다.
- **ContainerId**: (선택적) 소스 컨테이너의 이름으로, 서버에서 연결을 고유하게 식별합니다.
- **CreditSize**: credit flow의 기본 크기.
- **IdleTimeout**: 임계값을 초과한 후 프레임이 수신되지 않으면 로컬 피어가 시간 초과를 트리거합니다. idle 시간 초과는 밀리초 단위로 측정되며, 마지막 프레임이 수신된 시점부터 시작됩니다.
- **MaxFrameSize**: 수락되는 최대 프레임 크기입니다.
- **MaxLinksPerSession**: 세션당 최대 링크 수.
- **WindowSize**: 기본 window 크기입니다.

AMQP Authentication은 Authentication 속성에 구성되어야 합니다.

- **AuthType**: 인증 유형
 - **amqp1authNone**: 구성되지 않음.
 - **amqp1authSASLAnonymous**: 익명 인증
 - **amqp1authSASLPlain**: user/password 인증입니다. 이 유형의 인증에는 다음 속성을 채워야 합니다.
 - Username
 - Password
 - **amqp1authSASLExternal**: 외부 인증

연결

연결은 클라이언트(일반적으로 메시징 애플리케이션 또는 서비스)가 서버(메시지 브로커)에 대한 TCP 연결을 시작하면서 시작됩니다. 클라이언트는 서버의 포트에 연결하며, 일반적으로 비-TLS 연결의 경우 5672, TLS 보안 연결의 경우 5671입니다. TCP 연결이 설정되면, 클라이언트와 서버는 사용할 AMQP 프로토콜 버전을 협상합니다. AMQP 1.0.0은 다양한 버전을 지원하며, 협상 중에 양측은 버전 1.0.0 사용에 합의합니다.

프로토콜 협상 후, 클라이언트는 서버의 구성에 따라 서버에 자신을 인증해야 할 수 있습니다. 인증 메커니즘에는 PLAIN, EXTERNAL과 같은 SASL(Simple Authentication and Security Layer) 메커니즘, 또는 서버가 지원하는 기타 메커니즘이 포함될 수 있습니다.

예제: 보안 포트 5671에서 수신 대기하고 SASL 자격 증명을 사용하는 AMQP 서버에 연결

```
// 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;
```

Sessions

인증되면 클라이언트는 AMQP 세션을 엽니다. 세션은 클라이언트와 서버 간 통신을 위한 논리적 컨텍스트입니다. 세션은 관련 메시징 작업을 함께 그룹화하는 데 사용됩니다. 새 세션을 생성하려면 **CreateSession** 메서드를 사용하십시오. 이 메서드를 사용하면 세션 이름을 설정하거나 비워 두면 구성 요소가 자동으로 하나를 할당합니다.

세션이 성공적으로 생성되면 세션의 세부 정보와 함께 **OnAMQPSessionOpen** 이벤트가 발생합니다.

```
oAMQP->CreateSession("MySession");
oAMQP->OnAMQPSessionOpen = AMQP1AMQPSessionOpen;
void __fastcall TMyClass::AMQP1AMQPSessionOpen(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Fr
{
    ShowMessage("#session-open: " + aSession->Id);
}
```

링크

세션 내에서 클라이언트는 queue, topic 또는 서버가 제공하는 기타 리소스와 같은 특정 엔터티와 통신하기 위해 link를 만듭니다. Link는 메시지를 보내고 받는 데 사용되는 양방향 통신 채널입니다.

구성 요소는 발신자 및 수신자 노드로 작동할 수 있습니다. **MaxLinksPerSession** 속성에 설정된 제한까지 각 세션에 대해 원하는 수의 링크를 생성할 수 있습니다.

송신자 링크

새 sender link를 생성하려면 **CreateSenderLink** 메서드를 사용하고 세션 이름과 선택적으로 sender link 이름을 전달하십시오. 링크가 성공적으로 생성되면 **OnAMQPLinkOpen** 이벤트가 발생합니다.

```
oAMQP->CreateSenderLink("MySession", "MySenderLink");
oAMQP->OnAMQPLinkOpen = AMQP1AMQPLinkOpen;
void __fastcall TMyForm::AMQP1AMQPLinkOpen(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *
{
    ShowMessage("#link-open: " + aLink->Name);
}
```

Receiver Links

새 receiver 링크를 생성하려면 **CreateReceiverLink** 메서드를 사용하여 세션 이름과 선택적으로 receiver 링크 이름을 전달하십시오. 링크가 성공적으로 생성되면 **OnAMQPLinkOpen** 이벤트가 발생합니다.

```
oAMQP->CreateSenderLink("MySession", "MySenderLink");
oAMQP->OnAMQPLinkOpen = AMQP1AMQPLinkOpen;
void __fastcall TMyForm::AMQP1AMQPLinkOpen(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *
{
    ShowMessage("#link-open: " + aLink->Name);
}
```

Sending Messages

세션이 설정되고 링크가 생성되면, 클라이언트는 대상으로 메시지를 보내는 것과 같은 메시지 작업을 수행하기 시작할 수 있습니다. sender 링크를 사용하여 메시지를 보내려면 `SendMessage` 메서드를 사용하십시오.

```
oAMQP->SendMessage("MySession", "MySenderLink", "My first AMQP Message");
```

메시지 수신

기본적으로 Receiver Link는 **자동 모드**로 생성되며, 이는 새 메시지가 도착할 때마다 클라이언트에게 전달됨을 의미합니다.

Receiver Link가 **수동 모드**로 생성된 경우, 동기 메서드 **GetMessage** 를 사용하여 새 메시지가 도착할 때까지 가져오고 기다리십시오.

Automatic 및 Manual 모드에서 새 메시지가 도착할 때마다 **OnAMQPMessage** 이벤트가 발생합니다.

```
void __fastcall OnAMQPMessageEvent(TObject *Sender, TsgcAMQP1Session *const aSession,
    TsgcAMQP1ReceiverLink *const aLink, TsgcAMQP1Message *const aMessage,
    TsgcAMQP1MessageDeliveryState &DeliveryState)
{
    ShowMessage(aMessage->ApplicationData->AMQPValue->Value);
}
```

참조

- 속성 — 구성 요소에서 사용 가능한 5개의 속성.
- 메서드 — 12개의 public 메서드.
- 이벤트 — 처리할 수 있는 14개의 이벤트.

Connection | Client AMQP1 Connect

AMQP 서버에 연결하려면, 먼저 [TsgcWebSocketClient](#)와 [TsgcWSPClient_AMQP1](#)을 생성해야 합니다. 그런 다음 AMQP1 구성 요소를 WebSocket Client에 연결해야 합니다.

연결이 성공하면 OnAMQPConnect 이벤트가 발생합니다.

기본 사용법

인증 없이 AMQP 1.0.0 서버에 연결합니다. AMQPOptions 속성 값, virtual host를 정의한 다음 TsgcWebSocketClient에서 서버의 Host와 Port를 설정하십시오.

TCP Plain 연결을 사용하는 경우, TsgcWebSocketClient 속성 Specifications.RFC6455를 false로 설정하십시오.

```
// 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;
```

Authentication

서버에 인증이 필요한 경우 AMQP:Authentication 속성을 사용하여 Username/Password 값을 설정하고 AuthType을 "amqp1authSASLPlain" 값으로 설정하십시오.

```
// 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 | Client AMQP1 Disconnect

클라이언트는 다음 메서드를 사용하여 현재 활성 연결을 끊을 수 있습니다:

Close Reason 보내기

AMQP 클라이언트는 연결이 종료될 것임을 서버에 알리고 연결을 종료하는 이유에 대한 정보를 제공할 수 있습니다. 서버에 연결 종료를 요청하려면 Close 메서드를 사용하십시오.

```
oAMQP.Close('invalid-frame', "The received frame has an invalid format.");
```

Await Close

기본적으로 **Close** 메서드는 **비동기**이므로 메서드를 호출한 후 코드는 계속됩니다. **Close** 메서드가 완료되고 서버가 보낸 확인이 수신될 때까지 기다리려면 Options 매개변수에서 **Await** 속성을 True로 설정하십시오.

```
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;
    }
}
```

소켓 연결 닫기

[TsgcWebSocketClient](#)의 Active 속성을 False로 설정하기만 하면 됩니다. [연결 종료](#)에 대해 자세히 읽을 수 있습니다.

Connection | Idle Timeout

연결은 **idle timeout** 임계값의 적용을 받습니다. 임계값을 초과한 후 서버로부터 프레임이 수신되지 않으면 클라이언트가 타임아웃을 트리거합니다. idle timeout은 밀리초로 측정되며 마지막 프레임이 수신된 시점부터 시작됩니다. 임계값을 초과하면 구성 요소가 서버에 Close Frame을 보냅니다. 서버가 10초 후에 응답하지 않으면 클라이언트가 TCP 소켓을 닫습니다.

Idle Timeout 값은 다음 속성에서 구성할 수 있습니다:

AMQPOptions.IdleTimeout

이 속성에 설정된 값은 AMQP 연결을 열 때 서버로 전송됩니다. 값이 0보다 크고 MaxInt 값의 절반보다 작으면, idle 시간 초과가 초과되지 않았는지 확인하기 위해 내부 타이머가 활성화됩니다.

예제: IdleTimeout 값을 60초로 설정

```
AMQPOptions.IdleTimeout = 60000
```

Connection | Connection State

AMQP 1.0.0은 다음 연결 상태를 정의합니다:

- **amqp1csUnknown:** 초기 상태입니다.
- **amqp1csStart:** 이 상태에서는 연결이 존재하지만, 아무것도 전송되거나 수신되지 않았습니다. 이는 구현이 socket connect 또는 socket accept를 수행한 직후에 있게 되는 상태입니다.
- **amqp1csHeaderReceived:** 이 상태에서는 피어로부터 연결 헤더가 수신되었지만 연결 헤더가 전송되지 않았습니다.
- **amqp1csHeaderSent:** 이 상태에서는 연결 헤더가 피어로 전송되었지만 아직 연결 헤더를 수신하지 않았습니다.
- **amqp1csHeaderExchanged:** 이 상태에서는 연결 헤더가 피어로 전송되었고 피어로부터 연결 헤더가 수신되었습니다.
- **amqp1csOpenPipe:** 이 상태에서는 연결 헤더와 open 프레임이 모두 전송되었지만 아무것도 수신되지 않았습니다.
- **amqp1csOpenClosePipe:** 이 상태에서는 연결 헤더, open 프레임, 파이프라인된 연결 트래픽 및 close 프레임이 전송되었지만 아무것도 수신되지 않았습니다.
- **amqp1csOpenReceived:** 이 상태에서는 연결 헤더가 교환되었습니다. 피어로부터 open 프레임을 받았지만 open 프레임은 아직 전송되지 않았습니다.
- **amqp1csOpenSent:** 이 상태에서는 연결 헤더가 교환되었습니다. open 프레임이 피어에게 전송되었지만 아직 open 프레임을 받지 못했습니다.
- **amqp1csClosePipe:** 이 상태에서는 연결 헤더가 교환되었습니다. open 프레임, 모든 파이프라인된 연결 트래픽 및 close 프레임이 전송되었지만 피어로부터 아직 open 프레임을 수신하지 못했습니다.
- **amqp1csOpened:** 이 상태에서는 연결 헤더와 open 프레임이 모두 송수신되었습니다.
- **amqp1csCloseReceived:** 이 상태에서는 피어가 AMQP close를 시작했음을 나타내는 close 프레임이 수신되었습니다. 연결에 더 이상 프레임이 도착할 것으로 예상되지 않습니다. 그러나 프레임은 여전히 전송될 수 있습니다. 원하는 경우 구현은 이 시점에서 연결의 읽기 측을 종료하기 위해 TCP half-close를 수행할 수 있습니다(MAY).
- **amqp1csCloseSent:** 이 상태에서는 close 프레임이 피어에 전송되었습니다. 연결에 더 이상 아무것도 쓰는 것은 불법이지만 여전히 들어오는 프레임이 있을 수 있습니다. 원하는 경우 구현은 이 시점에서 TCP half-close를 수행하여 연결의 쓰기 측을 종료할 수 있습니다.
- **amqp1csDiscarding:** DISCARDING 상태는 close가 오류에 의해 트리거되는 CLOSE SENT 상태의 변형입니다. 이 경우 연결에 들어오는 모든 프레임은 피어의 close 프레임이 수신될 때까지 자동으로 무시되어야 합니다.
- **amqp1csEnd:** 이 상태에서는 어느 엔드포인트도 연결에 더 이상 아무것도 쓰는 것이 불법입니다. 연결을 안전하게 닫고 폐기할 수 있습니다.

AMQP Client에는 클라이언트 구성 요소가 어떤 연결 상태에 있는지 확인할 수 있는 **ConnectionState** 속성이 있습니다.

Connection | AMQP1 Authentication

구성 요소에는 다음 인증 메서드가 있습니다:

- **amqp1authNone:** 서버에 연결할 때 사용할 인증 방법이 없습니다.
- **amqp1authSASLAnonymous:** 익명으로 연결합니다.
- **amqp1authSASLPlain:** 기본값, 사용자/비밀번호 인증을 사용합니다.
- **amqp1authSASLExternal:** 현재 지원되지 않습니다.

SASL Authentication

가장 일반적인 인증은 **amqp1authSASLPlain** 유형을 사용하는 것입니다. 이 인증 유형은 AMQP1 구성 요소에서 `AMQPOptions.Authentication` 속성에 액세스하여 활성화할 수 있습니다.

- **AuthType:** `amqp1authSASLPlain`을 선택하십시오
- **Username:** SASL Authentication에 사용할 사용자입니다.
- **Password:** SASL 인증에 사용할 비밀 값입니다.

SASL Authentication의 결과는 **OnAMQP1SASLAuthentication** 이벤트가 발생할 때 얻을 수 있습니다.

```
void __fastcall OnAMQP1SASLAuthentication(System::TObject* Sender, TsgcAMQP1SaslCode aCode,
    const System::UnicodeString aDescription, bool &Handled)
{
    ShowMessage("#sas1-authentication: " + aDescription);
}
```

Connection | Azure MessageBus

AMQP(Advanced Message Queuing Protocol)는 다양한 노드의 다양한 컨테이너 간 통신을 용이하게 하도록 설계된 강력한 메시징 시스템입니다. 메시지 전송을 위한 프로토콜과 메시지 자체의 구조적 프레임워크를 모두 표준화하여 일관되고 신뢰할 수 있는 통신을 보장합니다. AMQP의 기본 사항을 더 깊이 파고들려면 [Getting Started with AMQP](#) 가이드를 참조하십시오.

eSeGeCe 라이브러리 내의 AMQP 구성 요소는 Amazon MQ 및 Azure Service Bus를 포함한 주요 클라우드 메시징 브로커와의 원활한 통합을 가능하게 합니다. 이 가이드는 AMQP 구성 요소를 사용하여 Azure Service Bus와 연결하는 데 중점을 두며, 메시지를 효율적으로 보내고 받을 수 있는 멀티 테넌트 애플리케이션을 구축하는 방법을 보여줍니다.

이 구성 요소는 큐, 토픽 및 구독과 같은 주요 기능을 지원하는 포괄적인 구현을 제공하여 최신 IoT 및 엔터프라이즈 애플리케이션에 이상적인 선택입니다.

Azure Configuration

시작하려면 Azure Portal 내에 **Service Bus** 리소스를 생성하십시오. 리소스가 설정되면 통합 및 구성에 필수적이므로 리소스의 **도메인 이름**을 반드시 기록해 두십시오.

namespace가 성공적으로 생성된 후에는 Azure Portal의 **namespace 개요**에서 직접 관리하고 모니터링할 수 있습니다. 이 중앙 집중식 인터페이스는 주요 관리 도구 및 설정에 대한 액세스를 제공하여 Service Bus 리소스의 원활한 관리를 가능하게 합니다.

SAS Authentication을 사용할 때 **사용자 이름**은 **SAS 정책 이름**이고 **비밀번호**는 **기본 또는 보조 키**입니다.

```
// 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
TsgcWScClient_AMQP1* oAMQP1 = new TsgcWScClient_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

Azure Service Bus는 초기 SASL 핸드셰이크 후 발신자와 수신자를 권한 부여하기 위해 AMQP를 통해 Claims-Based Security (CBS)를 구현합니다. 클라이언트는 **\$cbs** 노드에 대한 관리 링크를 열고 Shared Access Signature (SAS) 토큰 또는 Microsoft Entra ID가 발급한 JSON Web Token (JWT)을 포함하는 **put-token** 요청을 보냅니다. 브로커가 토큰을 검증하면 권한 부여가 수명 동안 캐시되고 애플리케이션은 큐, 토픽 또는 구독에 대해 발신자 및 수신자 링크를 생성할 수 있습니다.

AMQP1 클라이언트는 두 가지 도우미 메서드를 통해 이 흐름을 자동화합니다:

- **CreateAzureCbsSasToken**은 CBS 송신자/수신자 링크 쌍을 설정하고, 대상 엔터티에 대한 SAS 토큰을 생성하며, 이를 **\$cbs**에 게시합니다. 공유 액세스 정책으로 인증할 때 사용하십시오.
- **CreateAzureCbsJWT**는 동일한 CBS 교환을 따르지만 JWT를 **\$cbs**로 보내기 전에 client-credentials 그랜트를 사용하여 Microsoft Entra ID(Azure AD)에서 액세스 토큰을 얻습니다.

두 메서드 모두 활성 AMQP 연결이 필요하며 다음 매개변수를 받습니다:

- **aName**: 내부적으로 생성된 CBS link pair의 식별자입니다.

- **aNamespace** 및 **aEntityName**: 토큰 audience를 빌드하는 데 사용되는 Service Bus 네임스페이스 (.servicebus.windows.net 접미사 제외)와 queue, topic 또는 subscription 경로.
- **aKeyName** / **aKeyValue**: SAS 토큰을 위한 공유 액세스 정책 이름과 키입니다. 구성 요소는 토큰에 서명하고 토큰 유형 servicebus.windows.net:sastoken을 사용하여 전송합니다.
- **aTenant**, **aApplicationId**, **aSecret**: client credentials 흐름으로 JWT를 요청하는 데 사용되는 Microsoft Entra(Azure AD) 디렉터리 ID, 애플리케이션(클라이언트) ID 및 클라이언트 시크릿입니다.
- **aListeningPort** (JWT): OAuth 2.0 리디렉션을 위한 로컬 HTTP 포트입니다(제공되지 않으면 기본값 8080).
- **aExpiration** 및 **aTimeout**: 발급된 토큰의 수명(초)과 CBS 협상의 최대 대기 시간(밀리초)입니다.
- **aRaiseIfError**: True로 설정되면 토큰 획득 또는 CBS 응답이 실패할 때 메서드가 예외를 발생시킵니다.

다음 예제는 메시지를 보내기 전에 CBS로 인증하는 방법을 보여줍니다.

```
// 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
TsgcWScClient_AMQP1* oAMQP1 = new TsgcWScClient_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);
```

다음 예제는 JWT를 사용하는 Microsoft Entra ID(Azure AD) 인증에만 초점을 맞춥니다. 클라이언트 자격 증명 흐름으로 토큰을 요청하고 메시지를 보내거나 받기 위한 링크를 생성하기 전에 이를 \$cbs에 게시하는 방법을 보여줍니다.

```
// 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
TsgcWScClient_AMQP1* oAMQP1 = new TsgcWScClient_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

AMQP(Advanced Message Queuing Protocol) 1.0.0 사양의 맥락에서, 세션은 클라이언트와 메시지 브로커 간의 통신을 위한 논리적 컨텍스트를 나타냅니다. AMQP 1.0.0 세션이 수반하는 내용은 다음과 같습니다:

- **논리적 컨텍스트:** 세션은 AMQP 클라이언트(생산자 또는 소비자)와 AMQP 브로커 간의 메시징 작업에 대한 논리적 컨텍스트를 설정합니다. 단일 연결 내에서 관련 메시징 작업을 함께 그룹화하는 방법을 제공합니다.
- **Communication Channel:** 세션은 메시지가 전송되고 수신되는 통신 채널 역할을 합니다. 메시지, acknowledgment 및 flow control 메커니즘의 교환을 캡슐화합니다.
- **Transactional Boundaries:** 세션은 메시지 작업에 대한 트랜잭션 경계를 정의합니다. 여러 메시지 송신 또는 수신을 단일 원자 단위로 그룹화할 수 있게 하여, 세션 내의 모든 작업이 성공적으로 처리되거나 전혀 처리되지 않도록 보장합니다.
- **Flow Control:** 세션은 클라이언트와 브로커 간에 메시지가 교환되는 속도를 규제하기 위해 flow control 메커니즘을 지원합니다. Flow control은 양측의 리소스가 과부하되는 것을 방지하여 효율적이고 안정적인 메시지 전달을 보장합니다.
- **Lifetime Management:** 세션은 생성될 때 시작되고 닫힐 때 끝나는 수명 주기를 가집니다. 클라이언트는 메시지 처리를 병렬화하거나 메시지 스트림을 격리하기 위해 단일 연결 내에서 여러 세션을 설정할 수 있습니다.
- **리소스 할당:** 세션은 브로커 내의 큐, 토픽 또는 구독과 같은 특정 리소스와 연결될 수 있습니다. 세션 내에서 송수신되는 메시지는 이러한 리소스에 바인딩되어 대상 지정 메시지 라우팅 및 전달을 가능하게 합니다.

요약하자면, AMQP 1.0.0 세션은 AMQP 클라이언트와 브로커 간의 메시지 교환을 위한 논리적 컨텍스트를 제공하여, 메시징 시스템 내에서 트랜잭션 무결성, 흐름 제어 및 리소스 관리를 용이하게 합니다. 메시징 작업이 수행되는 경계를 정의하고 효율적이고 안정적인 메시지 교환을 보장하는 데 도움이 됩니다.

세션 열기

CreateSession 메서드는 지정된 이름(또는 비어 있으면 임의의 이름)으로 새 세션을 생성합니다. 세션이 이미 존재하면 예외가 발생합니다. 클라이언트를 사용하면 동일한 AMQP 연결을 사용하여 여러 세션을 생성할 수 있습니다.

세션이 성공적으로 생성되면 **OnAMQPSessionOpen** 이벤트가 발생합니다.

```
oAMQP->CreateSession("MySession");
oAMQP->OnAMQPSessionOpen = OnAMQPSessionOpenEvent;
void __fastcall TMyClass::OnAMQPSessionOpenEvent(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Frame *aFrame)
{
    ShowMessage("#session-open: " + aSession->Id);
}
```

CreateSession 메서드는 세션 정보를 포함하는 **TsgcAMQP1Session** 클래스를 반환합니다.

Await Open Session

기본적으로 **CreateSession** 메서드는 비동기이므로, 메서드를 호출한 후 코드가 계속됩니다. **CreateSession** 메서드가 완료되고 서버가 보낸 확인이 수신될 때까지 기다리려면, Options 매개변수에서 **Await** 속성을 True로 설정하십시오.

```
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

기존 세션을 닫으려면 닫을 세션의 이름을 전달하여 **CloseSession** 메서드를 사용하십시오.

세션이 성공적으로 닫히면, **OnAMQPSessionClose** 이벤트가 발생합니다.

```
oAMQP->CloseSession("MySession");
oAMQP->OnAMQPSessionClose = OnAMQPSessionCloseEvent;
// OnAMQPSessionCloseEvent implementation
void __fastcall TMyClass::OnAMQPSessionCloseEvent(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Frame *aEnd, TsgcAMQP1Error *aErr)
{
    ShowMessage("#session-close: " + aSession->Id + " [" + IntToStr(aSession->Channel) + "] reason: " + aEnd->ErrorReason);
}
```

Await Close Session

기본적으로 **CloseSession** 메서드는 비동기이므로 메서드를 호출한 후 코드가 계속됩니다. **CloseSession** 메서드가 완료되고 서버가 보낸 확인이 수신될 때까지 기다리려면 Options 매개변수에서 **Await** 속성을 True로 설정하십시오.

```
void CloseSession(const System::UnicodeString aSession)
{
    TsgcAMQP1MethodOptions_SessionClose *oOptions = new TsgcAMQP1MethodOptions_SessionClose();
    try
    {
        oOptions->Await = true;
        AMQP1->CloseSession(aSession, oOptions);
    }
    __finally
    {
        delete oOptions;
    }
}
```

Commands | AMQP1 Links

AMQP(Advanced Message Queuing Protocol) 1.0.0 사양에서 link는 AMQP 클라이언트와 메시지 브로커 간의 단방향 통신 채널을 나타냅니다. AMQP 1.0.0 link가 무엇을 수반하는지 더 자세히 살펴보겠습니다:

- **통신 채널:** 링크는 AMQP 발신자와 수신자 간에 메시지가 흐르는 경로 역할을 합니다. 발신자에서 수신자로 또는 그 반대로 한 방향으로 메시지를 전송할 수 있습니다.
- **단방향 흐름:** 각 링크는 단방향이며, 이는 메시지가 링크를 따라 한 방향으로만 이동할 수 있음을 의미합니다. 양방향 통신이 필요한 경우, 각 방향당 하나씩 두 개의 링크를 설정해야 합니다.
- **Message Transfer:** 메시지는 AMQP 프로토콜 규칙에 따라 링크를 통해 전송됩니다. 이러한 메시지에는 페이로드, 메시지 속성 및 통신에 필요한 추가 메타데이터가 포함될 수 있습니다.
- **Resource Binding:** 링크는 AMQP 브로커 내의 특정 리소스(예: queues, topics 또는 exchanges)와 연결됩니다. 링크를 통해 송수신되는 메시지는 이러한 리소스로 향하거나 그로부터 시작됩니다.
- **Flow Control:** Link는 메시지가 송수신되는 속도를 조절하는 흐름 제어 메커니즘을 지원합니다. 흐름 제어는 발신자나 수신자 모두 교환되는 메시지의 양에 압도되지 않도록 보장합니다.
- **수명 관리:** 링크는 설정될 때 시작되고 닫힐 때 끝나는 수명 주기를 가집니다. 필요에 따라 동적으로 생성되고 더 이상 필요하지 않을 때 닫힐 수 있습니다.
- **주소 지정:** Link는 통신의 소스 및 대상 엔드포인트를 지정하는 고유 주소로 식별됩니다. 이러한 주소를 통해 클라이언트와 브로커는 적절한 엔드포인트를 식별하고 연결을 설정할 수 있습니다.
- **트랜잭션 경계:** Link는 메시지 작업의 트랜잭션 경계를 정의합니다. 여러 메시지 전송 또는 수신을 단일 원자 단위로 그룹화하여 메시지 전달의 일관성과 신뢰성을 보장합니다.

요약하자면, AMQP 1.0.0 링크는 AMQP 클라이언트와 브로커 간의 단방향 통신 수단을 제공하여, 메시징 시스템 내에서 흐름 제어, 리소스 바인딩, 주소 지정 및 트랜잭션 무결성을 지원하면서 메시지 전송을 용이하게 합니다. AMQP 프로토콜에서 메시지 교환의 기본 구성 요소를 형성합니다.

Link에는 2가지 유형이 있습니다.

- **Sender Links:** 이러한 link는 메시지를 보내는 데 사용됩니다.
- **Receiver Links:** 이러한 링크는 메시지를 수신하는 데 사용됩니다.

새 링크가 생성되거나 삭제될 때마다 다음 이벤트가 발생합니다:

- **OnAMQPLinkOpen:** 이 이벤트는 새 link가 생성될 때 트리거됩니다. aLink.Mode 속성을 사용하여 link가 receiver 또는 sender 모드인지 확인하십시오.
- **OnAMQPLinkClose:** 이 이벤트는 링크가 닫힐 때 트리거됩니다.

Commands | AMQP1 Sender Links

AMQP 1.0.0 프로토콜에서 **Sender Link**는 메시지 전송을 목적으로 AMQP 클라이언트와 AMQP 서버 간에 설정된 통신 채널입니다. 이는 클라이언트와 서버 간 통신을 위한 논리적 채널을 나타내는 AMQP 세션의 컨텍스트 내에서 작동합니다.

Sender Link 생성

새 **Sender Link**를 생성하려면 다음 매개변수를 포함하는 **CreateSenderLink** 메서드를 호출하십시오.

- **Session:** sender link가 첨부될 세션 이름입니다.
- **Name:** (선택 사항) sender 링크의 이름입니다. 설정하지 않으면 임의의 이름이 자동으로 할당됩니다.
- **Target:** (선택 사항) "target" 매개변수를 설정하여 원격 호스트에서 메시지를 수신할 대상을 지정할 수 있습니다. 그러나 특정 시나리오에서는 target 지정이 필요하지 않을 수 있습니다. 이러한 경우 빈 문자열을 제공하는 것으로 충분합니다.
- **SendSettleMode:** (기본값 mixed) AMQP는 Message Settlement 메커니즘을 통해 전달 보장을 논의하는 기능을 제공합니다. 링크를 설정할 때 발신자와 수신자 모두 결제 모드(각 역할별로 하나씩)를 논의하고 합의합니다. 발신자는 다음 모드 중 하나에서 작동합니다:
 - **amqp1ssmSettled:** 메시지가 전송되면 성공적으로 전달되고 확인된 것으로 간주됩니다.
 - **amqp1ssmUnsettled:** 메시지가 수신자에 의해 명시적으로 수락되거나 거부될 때까지 결제된 것으로 간주되지 않습니다. 이를 통해 메시지 처리 및 핸들링에 대한 더 많은 제어가 가능합니다.
 - **amqp1ssmMixed:** 단일 AMQP 세션 내에서 결제 및 미결제 모드의 조합을 사용할 수 있습니다. **SendMessage** 메서드의 **MessageOptions** 매개변수를 사용하여 메시지의 결제 여부를 구성하십시오.

Sender Link가 성공적으로 생성되면 **OnAMQPLinkOpen** 이벤트가 발생합니다.

```
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

기본적으로 **CreateSenderLink** 메서드는 비동기이므로, 메서드를 호출한 후 코드가 계속됩니다. **CreateSenderLink** 메서드가 완료되고 서버가 보낸 확인이 수신될 때까지 기다리려면, Options 매개변수에서 **Await** 속성을 True로 설정하십시오.

```
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;
    }
}
```

Sending Messages

새 Message를 보내려면 다음 매개변수를 포함하는 **SendMessage** 메서드를 호출하십시오:

- **Session:** 세션의 이름.

- **Link:** 발신자 링크의 이름.
- **Text:** 문자열 메시지의 텍스트.

```
oAMQP1->SendMessage("MySession", "MySenderLink", "My first AMQP Message");
```

Sending Messages Mixed Mode

Sender Link가 Mixed 모드(기본값)로 생성된 경우, 메시지를 전송할 때 사용자는 메시지가 **settled** 되기를 원하는지 **아닌지** 설정할 수 있습니다. 메시지가 settled되는지 여부를 정의하려면 **MessageOptions** 매개변수를 사용하십시오.

```
TsgcAMQP1MessageOptions* oMessageOptions = new TsgcAMQP1MessageOptions();
try
{
    oMessageOptions->Settled = true;
    oAMQP1->SendMessage("MySession", "MySenderLink", "MyMessage", "message-id", oMessageOptions);
}
finally
{
    delete oMessageOptions;
}
```

Close Sender Link

기존 Sender Link를 닫으려면, 다음 매개변수를 포함하는 **CloseLink** 메서드를 호출하십시오:

- **Session:** 링크를 포함하는 세션의 이름입니다.
- **Link:** sender 링크의 이름입니다.
- **Error:** (선택적) 여기에 링크가 닫힌 이유를 설정할 수 있습니다.

Sender Link가 성공적으로 닫히면 **OnAMQP1LinkClose** 이벤트가 발생합니다.

```
oAMQP->CloseLink("MySession", "MySenderLink");
void __fastcall TForm1::OnAMQP1LinkCloseEvent(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *const aLink, TsgcAMQP1LinkCloseEventArgs *const aArgs)
{
    ShowMessage("#link-close: " + aLink->Name);
}
```

Sender Link 닫기 대기

기본적으로 **CloseLink** 메서드는 **비동기**이므로, 메서드를 호출한 후 코드가 계속됩니다. **CloseLink** 메서드가 완료될 때까지 대기하고 서버가 보낸 확인을 수신하려면, Options 매개변수에서 **Await** 속성을 True로 설정하십시오.

```
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;
    }
}
```


명령 | AMQP1 Receiver Link

AMQP 1.0.0 프로토콜에서 **Receiver Link**는 메시지 수신을 위해 AMQP 클라이언트와 AMQP 서버 간에 설정된 통신 채널입니다. 이는 클라이언트와 서버 간의 통신을 위한 논리적 채널을 나타내는 AMQP 세션의 컨텍스트 내에서 작동합니다.

Receiver Link 생성

새 **Receiver Link**를 생성하려면 다음 매개변수를 포함하는 **CreateReceiverLink** 메서드를 호출하십시오:

- **Session:** sender link가 첨부될 세션 이름입니다.
- **Name:** (선택 사항) sender 링크의 이름입니다. 설정하지 않으면 임의의 이름이 자동으로 할당됩니다.
- **Source:** (선택적) source는 송신자 역할을 하도록 되어 있는 원격 호스트의 노드 위치를 나타내도록 구성할 수 있습니다. 일부 상황에서는 이 주소를 지정할 필요가 없을 수 있습니다. 그러한 경우 매개변수 값으로 단순히 빈 문자열을 제공하는 것으로 충분합니다.
- **ReadMode:** (기본값 amqp1srmAuto) 수신자 링크는 메시지를 수신하기 위해 두 가지 모드 중 하나로 작동할 수 있습니다:
 - **amqp1srmAuto:** 자동 모드, 이 모드에서 수신자는 메시지가 사용 가능해지는 즉시 신속하게 수신되도록 능동적으로 작동합니다. 새 메시지가 도착할 때마다 명시적인 지시 없이 자동으로 메시지를 수신 대기하고 수신합니다.
 - **amqp1srmManual:** 페치 기반 모드, 이 모드에서 수신자는 특별히 지시받았을 때만 새 메시지를 검색하거나 페치합니다. 자동 모드와 달리 수신자는 새 메시지를 능동적으로 수신 대기하지 않고 대신 다음 메시지를 페치하라는 수동 지시를 기다립니다.
- **RcvSettleMode:** (기본값 amqp1rsmFirst) Receiver Link는 다음 모드 중 하나에서 작동합니다:
 - **amqp1rsmFirst:** 메시지가 도착하면 즉시 처리되고 확인됩니다. 메시지가 전송될 때 이미 확인되지 않은 경우, 송신자는 메시지가 수신되었음을 통보받습니다.
 - **amqp1rsmSecond:** 도착하는 메시지는 송신자가 먼저 확인한 후에만 확인됩니다. 또한 메시지가 전송될 때 이미 확인되지 않은 경우, 송신자는 메시지가 수신되었을 때 알림을 받습니다.

Receiver Link가 성공적으로 생성되면 **OnAMQPLinkOpen** 이벤트가 발생합니다.

```
oAMQP1->CreateReceiverLink("MySession", "MyReceiverLink");
oAMQP1->OnAMQPLinkOpen = AMQP1AMQPLinkOpen;
void __fastcall TMyForm::AMQP1AMQPLinkOpen(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *const aLink)
{
    ShowMessage("#link-open: " + aLink->Name);
}
```

Receiver Link 생성 대기

기본적으로 **CreateReceiverLink** 메서드는 비동기이므로, 메서드를 호출한 후 코드가 계속됩니다. **CreateReceiverLink** 메서드가 완료될 때까지 대기하고 서버가 보낸 확인을 수신하려면, Options 매개변수에서 **Await** 속성을 True로 설정하십시오.

```
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

Receiver Link가 Manual ReadMode에서 작동할 때, 새 메시지를 가져오려면 **GetMessage** 메서드를 호출하십시오. 이 메서드는 동기식이며, 이는 시간 초과(기본값 10초)가 초과될 때까지 기다린다는 것을 의미합니다. 메서드가 호출되면, 구성 요소는 크레딧을 한 단위 증가시키고 새 메시지가 도착하거나 시간 초과가 초과될 때까지 기다립니다. 메시지가 도착하지 않으면 크레딧이 다시 0으로 설정됩니다.

GetMessage 메서드에는 다음 매개변수가 있습니다:

- **Session:** 링크를 포함하는 세션의 이름입니다.
- **Link:** 수신자 링크의 이름입니다.
- **Timeout:** (기본값 1000 = 10초) 함수가 새 메시지를 받기 위해 대기하는 최대 시간입니다.

Close Receiver Link

기존 Receiver Link를 닫으려면 다음 매개변수를 포함하는 **CloseLink** 메서드를 호출하십시오.

- **Session:** 링크를 포함하는 세션의 이름입니다.
- **Link:** 수신자 링크의 이름입니다.
- **Error:** (선택 사항) 링크가 닫힌 이유를 여기에 설정할 수 있습니다.

Receiver Link가 성공적으로 닫히면 **OnAMQPLinkClose** 이벤트가 발생합니다.

```
oAMQP1->CloseLink("MySession", "MyReceiverLink");
void __fastcall TForm1::OnAMQPLinkCloseEvent(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *cc
{
    ShowMessage("#link-close: " + aLink->Name);
}
```

Await Close Receiver Link

기본적으로 **CloseLink** 메서드는 비동기이므로, 메서드를 호출한 후 코드가 계속됩니다. **CloseLink** 메서드가 완료될 때까지 대기하고 서버가 보낸 확인을 수신하려면, Options 매개변수에서 **Await** 속성을 True로 설정하십시오.

```
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

Sender Link를 생성하는 방법을 알아보려면 먼저 [AMQP1 Sender Links](#)를 읽으십시오.

메시지 보내기

AMQP1 Server에 텍스트 메시지를 보내려면 Session 및 SenderLink 이름을 전달하여 **SendMessage** 메서드를 사용하십시오. 이 메서드에는 다음 매개변수가 있습니다:

- **Session:** 세션의 이름.
- **Link:** 발신자 링크의 이름.
- **Text:** 메시지의 텍스트.
- **MessageId:** (선택적) 메시지의 id입니다. unsettled 모드를 사용할 때 서버가 메시지를 처리했는지 확인하는 데 사용할 수 있습니다.
- **Options:** (선택적) 메시지를 보낼 때 일부 옵션을 사용자 정의할 수 있습니다.
 - **Settled:** 혼합 모드에서 sender 링크를 사용할 때, 메시지를 보낼 때 Settled 속성을 사용자 정의할 수 있습니다.
 - **Await:** 메시지가 미정산 상태이고 값이 true이면, 코드는 메시지가 서버에 의해 처리되거나 타임아웃을 초과할 때까지 대기합니다.
 - **Timeout:** await가 true인 경우 밀리초 단위의 값(기본값 10000).
 - **RaiseTimeoutException:** 타임아웃을 초과하면 예외가 발생합니다(기본값 true).

```
oAMQP1->SendMessage("MySession", "MySenderLink", "My first AMQP Message");
```

Await Send Message

기본적으로 **SendMessage** 메서드는 메시지를 unsettled로 보낼 때 비동기적입니다. **Await** 속성을 true로 설정하면 클라이언트는 메시지가 처리되었다는 서버의 확인을 받을 때까지 기다립니다.

```
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;
    }
}
```

Events

메시지를 전송할 때 메시지가 언제 전송되었는지 그리고 메시지가 서버에 의해 처리되었는지(unsettled로 전송하는 경우)를 알 수 있는 2개의 이벤트를 사용할 수 있습니다.

- **OnAMQPMessageSent:** 이 이벤트는 메시지가 서버로 전송된 후에 호출됩니다. SendMessage 메서드를 호출하면 메시지가 내부 큐에 저장되고 보조 스레드에 의해 처리되므로 메시지가 전송된 후 이 이벤트가 호출됩니다.

- **OnAMQPMessageSentAck**: 클라이언트가 메시지가 AMQP1 Server에 의해 처리되었다는 확인을 받을 때 이 이벤트가 호출됩니다.

```
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

새 메시지가 수신될 때마다 **OnAMQPMessage** 이벤트가 발생합니다.

TsgcAMQP1Message 인스턴스에는 수신된 메시지가 포함됩니다. **aMessage.ApplicationData.AMQPValue.Value** 속성을 사용하여 텍스트 메시지에 액세스할 수 있습니다.

Delivery Outcome을 지정하려면 **DeliveryState** 매개변수를 사용하십시오. 기본적으로 모든 메시지는 accepted 상태이지만 다음 중 하나를 설정할 수 있습니다:

- **amqp1mdtsAccepted:** 메시지가 성공적으로 처리되었습니다.
- **amqp1mdtsRejected:** 메시지를 성공적으로 처리하지 못했습니다. **DeliveryState.Rejected** 속성을 사용하여 오류를 설정하십시오.
- **amqp1mdtsReleased:** 메시지가 처리되지 않았고 처리되지 않을 것입니다.
- **amqp1mdtsModified:** amqp1mdtsReleased와 동일하지만 **DeliveryState.Modified** 속성을 사용하여 추가 데이터를 추가할 수 있습니다.

```
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

STOMP는 Simple (또는 Streaming) Text Orientated Messaging Protocol입니다. STOMP는 상호 운용 가능한 와이어 포맷을 제공하므로 STOMP 클라이언트가 모든 STOMP 메시지 브로커와 통신하여 여러 언어, 플랫폼 및 브로커 간에 쉽고 광범위한 메시징 상호 운용성을 제공할 수 있습니다.

당사의 STOMP 클라이언트 구성 요소는 다음 STOMP 버전을 지원합니다: 1.0, 1.1 및 1.2.

구성 요소

TsgcWSPClient_STOMP: 일반 STOMP Protocol 클라이언트로, 모든 STOMP Server에 연결할 수 있습니다.

TsgcWSPClient_STOMP_RabbitMQ: RabbitMQ Broker용 STOMP 클라이언트.

TsgcWSPClient_STOMP_ActiveMQ: ActiveMQ Broker용 STOMP 클라이언트.

TsgcWSPClient_STOMP

이것은 클라이언트 프로토콜 STOMP 구성 요소입니다.

소개

이것은 Client Protocol STOMP 구성 요소입니다. 이 구성 요소를 폼에 놓고 Client 속성을 사용하여 [TsgcWebSocketClient](#) 구성 요소를 선택해야 합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 속성 8개입니다.
- 메서드 — 11개의 public 메서드.
- 이벤트 — 처리할 수 있는 6개의 이벤트.

TsgcWSPClient_STOMP_RabbitMQ

이것은 RabbitMQ Broker용 Client Protocol STOMP 구성 요소입니다.

소개

이것은 RabbitMQ Broker용 Client Protocol STOMP 구성 요소입니다. 이 구성 요소를 폼에 놓고 Client 속성을 사용하여 [TsgcWebSocketClient](#) 구성 요소를 선택해야 합니다.

Destinations

STOMP 사양은 브로커가 지원해야 하는 destination의 종류를 규정하지 않으며, 대신 SEND 및 MESSAGE 프레임의 destination 헤더 값은 브로커별로 다릅니다. RabbitMQ STOMP 어댑터는 여러 가지 다른 destination 유형을 지원합니다:

- **Topic:** transient 및 durable 토픽에 SEND 및 SUBSCRIBE합니다.
- **Queue:** STOMP 게이트웨이가 관리하는 큐에 SEND 및 SUBSCRIBE.
- **QueueOutside:** STOMP 게이트웨이 외부에서 생성된 큐에 SEND 및 SUBSCRIBE합니다.
- **TemporaryQueue:** 임시 큐를 생성합니다(reply-to 헤더에만).
- **Exchange:** 임의의 라우팅 키에 SEND하고 임의의 바인딩 패턴에 SUBSCRIBE합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 8개의 속성.
- 메서드 — 25개의 public 메서드.
- 이벤트 — 처리할 수 있는 이벤트 6개입니다.

TsgcWSPClient_STOMP_ActiveMQ

이것은 ActiveMQ Broker를 위한 클라이언트 프로토콜 STOMP 구성 요소입니다.

소개

이것은 ActiveMQ Broker용 클라이언트 프로토콜 STOMP 구성 요소입니다. 폼에 이 구성 요소를 놓고 Client 속성을 사용하여 [TsgcWebSocketClient](#) 구성 요소를 선택해야 합니다.

Destinations

STOMP 사양은 브로커가 지원해야 하는 대상의 종류를 규정하지 않으며, 대신 SEND 및 MESSAGE 프레임의 destination 헤더 값은 브로커마다 다릅니다. Active STOMP 어댑터는 다음과 같은 여러 가지 다른 대상 유형을 지원합니다:

- **Topic:** transient 및 durable 토픽에 SEND 및 SUBSCRIBE합니다.
- **Queue:** STOMP 게이트웨이가 관리하는 큐에 SEND 및 SUBSCRIBE.

Publish Options

STOMP는 가능한 한 단순하게 설계되었으므로 모든 스크립팅 언어/플랫폼이 최소한의 노력으로 다른 언어/플랫폼에 메시지를 보낼 수 있습니다. STOMP는 메시지 전송 및 수신과 같은 각 요청에 플러그형 헤더를 허용합니다. ActiveMQ에는 Stomp 프로토콜에 대한 여러 확장이 있어 JMS 의미를 Stomp 클라이언트에서 지원할 수 있습니다. OpenWire JMS 프로듀서는 Stomp 컨슈머에 메시지를 보낼 수 있고, Stomp 프로듀서는 OpenWire JMS 컨슈머에 메시지를 보낼 수 있습니다. 그리고 Stomp 대 Stomp 구성은 더 풍부한 JMS 메시지 제어를 사용할 수 있습니다.

STOMP는 SENT 메시지에서 다음 표준 JMS 속성을 지원합니다.

- **CorrelationId:** 좋은 소비자는 보내는 모든 응답에 이 헤더를 추가합니다.
- **Expires:** 메시지의 만료 시간입니다.
- **JMSXGroupID:** Message Group을 지정합니다.
- **JMSXGroupSeq:** Message Groups에서 시퀀스 번호를 지정하는 선택적 헤더.
- **Persistent:** 메시지가 영속적인지 여부.
- **Priority:** 메시지의 우선순위입니다.
- **ReplyTo:** 회신을 보내야 하는 대상입니다.
- **MsgType:** 메시지의 유형입니다.

참조

- 속성 — 구성 요소에서 사용 가능한 9개의 속성.
- 메서드 — 16개의 public 메서드.
- 이벤트 — 처리할 수 있는 6개의 이벤트.

프로토콜 AppRTC

WebRTC(Web Real-Time Communication)는 플러그인 없이 음성 통화, 화상 채팅, P2P 파일 공유를 위한 브라우저 대 브라우저 애플리케이션을 가능하게 하기 위해 World Wide Web Consortium(W3C)에서 초안 작성 중인 API 정의입니다. WebRTC의 RTC는 Real-Time Communications를 의미하며, 브라우저 클라이언트(피어) 간의 오디오/비디오 스트리밍 및 데이터 공유를 가능하게 하는 기술입니다. 표준 집합으로서 WebRTC는 모든 브라우저에 플러그인이나 서드파티 소프트웨어를 설치할 필요 없이 애플리케이션 데이터를 공유하고 피어 투 피어 화상 회의를 수행할 수 있는 기능을 제공합니다.

WebRTC 구성 요소는 JavaScript API로 액세스됩니다. 현재 개발 중인 것은 오디오 또는 비디오 데이터 스트림을 나타내는 Network Stream API와, 두 명 이상의 사용자가 브라우저 간에 통신할 수 있게 하는 PeerConnection API입니다. 또한 실시간 게임, 텍스트 채팅, 파일 전송 등을 위해 다른 유형의 데이터 통신을 가능하게 하는 DataChannel API도 개발 중입니다.

[appr.tc](#)는 Google과 Mozilla가 개발한 WebRTC 데모 애플리케이션으로, 두 브라우저가 WebRTC API를 사용하여 서로 "대화"할 수 있게 합니다.

구성 요소

[TsgcWSPServer_AppRTC](#): Server Protocol AppRTC VCL 구성 요소입니다.

TsgcWSPServer_AppRTC

이것은 Server Protocol AppRTC 구성 요소입니다.

소개

이것은 Server Protocol AppRTC 구성 요소입니다. 이 구성 요소를 폼에 배치하고 Server 속성을 사용하여 [TsgcWebSocketServer](#) 구성 요소를 선택해야 합니다.

Parameters

- **IceServers:** 여기서 WebRTC 연결을 위한 turn/stun 서버를 구성할 수 있습니다.
- **RoomLink:** 방에 접근하기 위한 URL 베이스입니다. 예: <https://mydemo.com/r/>
- **WebSocketURL:** WebSocket 서버에 대한 URL입니다. 예: <wss://mydemo.com>

WebRTC 프로토콜에는 STUN/TURN 서버가 필요하며, 데모는 테스트 목적으로 public STUN/TURN 서버를 사용합니다. 프로덕션 시스템에 적용하려면 전용 STUN/TURN 서버가 필요합니다.

등록된 사용자는 **Windows용 Coturn** 서버의 컴파일된 바이너리를 다운로드할 수 있습니다. [COTURN STUN/TURN](#)에 대해 자세히 알아보십시오.

IceServers 구성

다음 IP 주소에서 STUN/TURN 서버를 실행 중이고: 51.122.4.88이며 포트 3478에서 수신 중입니다. 연결할 사용자는 "apprtc"이고 자격 증명은 "secret"입니다. 다음과 같이 IceServers를 구성하십시오:

```
{
  "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"
}
```

참조

- Properties — 구성 요소에서 사용 가능한 5개의 속성.
- 메서드 — 0개의 public 메서드.
- 이벤트 — 처리할 수 있는 11개의 이벤트.

프로토콜 WebRTC

WebRTC(Web Real-Time Communication)는 플러그인 없이 음성 통화, 화상 채팅, P2P 파일 공유를 위한 브라우저 대 브라우저 애플리케이션을 가능하게 하기 위해 World Wide Web Consortium(W3C)에서 초안 작성 중인 API 정의입니다. WebRTC의 RTC는 Real-Time Communications를 의미하며, 브라우저 클라이언트(피어) 간의 오디오/비디오 스트리밍 및 데이터 공유를 가능하게 하는 기술입니다. 표준 집합으로서 WebRTC는 모든 브라우저에 플러그인이나 서드파티 소프트웨어를 설치할 필요 없이 애플리케이션 데이터를 공유하고 피어 투 피어 화상 회의를 수행할 수 있는 기능을 제공합니다.

WebRTC 구성 요소는 JavaScript API로 액세스됩니다. 현재 개발 중인 것은 오디오 또는 비디오 데이터 스트림을 나타내는 Network Stream API와, 두 명 이상의 사용자가 브라우저 간에 통신할 수 있게 하는 PeerConnection API입니다. 또한 실시간 게임, 텍스트 채팅, 파일 전송 등을 위해 다른 유형의 데이터 통신을 가능하게 하는 DataChannel API도 개발 중입니다.

구성 요소

[TsgcWSPServer_WebRTC](#): 서버 프로토콜 WebRTC VCL 구성 요소.

Parameters

- **IceServers:** 여기서 WebRTC 연결을 위한 turn/stun 서버를 구성할 수 있습니다. 기본적으로 다음 공용 STUN 서버를 사용합니다

```
{"iceServers": [{"url": "stun:stun.l.google.com:19302"}]}
```

Browser Test

즐거 사용하는 웹 브라우저로 이 프로토콜을 테스트하려면 다음 url을 입력하십시오(사용자 정의 호스트와 포트를 정의해야 함)

```
http://host:port/webrtc.esegece.com.html
```

TsgcWSPServer_WebRTC

이것은 Server Protocol WebRTC 구성 요소입니다.

소개

이것은 Server Protocol WebRTC 구성 요소입니다. 이 구성 요소를 폼에 놓고 Server 속성을 사용하여 [TsgcWebSocketServer](#) 구성 요소를 선택해야 합니다.

WebRTC 프로토콜에는 STUN/TURN 서버가 필요하며, 데모는 테스트 목적으로 public STUN/TURN 서버를 사용합니다. 프로덕션 시스템에 적용하려면 전용 STUN/TURN 서버가 필요합니다.

등록된 사용자는 **Windows용 Coturn** 서버의 컴파일된 바이너리를 다운로드할 수 있습니다. [COTURN STUN/TURN](#)에 대해 자세히 알아보십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 5개의 속성.
- Methods — 2개의 public 메서드.
- 이벤트 — 처리할 수 있는 11개의 이벤트.

Protocol WebRTC Javascript

여기에서 사용 가능한 메서드를 찾을 수 있습니다. 필요에 따라 {%host%} 및 {%port%} 변수를 교체해야 합니다. 예제: sgcWebSocket 서버가 www.example.com 웹사이트에서 포트 80을 수신하도록 구성된 경우 다음을 구성해야 합니다:

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
<script src="http://www.example.com:80/webrtc.esegce.com.js"></script>
```

Open Connection

WebSocket 연결이 열리면, 브라우저가 로컬 카메라와 마이크에 대한 액세스를 요청하므로, 액세스를 허용해야 합니다.

```
<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

브라우저가 로컬 카메라 및 마이크에 액세스할 수 있을 때, 'sgcmediastart' 이벤트가 발생하며 그러면 webrtc_connect 프로시저를 사용하여 다른 클라이언트에 연결을 시도할 수 있습니다

```
<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>
```

WebRTC 채널 닫기

```
<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>
```

Protocol WAMP

WAMP은 두 가지 비동기 메시징 패턴인 RPC와 PubSub를 제공하는 개방형 WebSocket 서버프로토콜입니다.

기술적으로, WAMP는 공식적으로 등록된 WebSocket 서버프로토콜 (WebSocket 위에서 실행)이며 JSON을 메시지 직렬화 형식으로 사용합니다.

RPC란 무엇입니까?

Remote Procedure Call(RPC)은 클라이언트와 서버라는 두 가지 역할의 피어를 포함하는 메시징 패턴입니다.

서버는 잘 알려진 엔드포인트에서 호출할 메서드 또는 프로시저를 제공합니다.

클라이언트는 메서드 또는 프로시저 엔드포인트와 호출에 대한 인수를 제공하여 원격 메서드 또는 프로시저를 호출합니다.

서버는 호출에 제공된 인수를 사용하여 메서드 또는 프로시저를 실행하고 호출 결과를 클라이언트에 반환합니다.

PubSub란 무엇입니까?

게시/구독(PubSub)은 세 가지 역할의 피어, 즉 게시자, 구독자, 브로커가 관여하는 메시징 패턴입니다.

게시자는 특정 피어가 아닌 추상 주소로 토픽(채널이라고도 함)을 제공하여 이벤트를 보냅니다(게시합니다).

구독자는 먼저 관심 있는 토픽(채널)을 제공하여 이벤트를 받습니다. 이후 구독자는 해당 토픽에 게시된 모든 이벤트를 받습니다.

브로커는 게시자와 구독자 사이에 위치하며 게시자가 구독자에게 게시하는 메시지를 중재합니다. 브로커는 토픽별 구독자 목록을 유지하여 새로 게시된 이벤트를 적절한 구독자에게 디스패치할 수 있습니다.

브로커는 또한 자체적으로 이벤트를 디스패치할 수 있습니다. 예를 들어 브로커가 RPC 서버 역할도 하고 서버에서 실행된 메서드가 PubSub 이벤트를 트리거해야 하는 경우입니다.

요약하면, PubSub은 중개자인 브로커를 통해 게시자와 수신자를 분리합니다.

구성 요소

[TsgcWSPServer_WAMP](#): Server Protocol WAMP VCL Component.

[TsgcWSPClient_WAMP](#): Client Protocol WAMP VCL 구성 요소.

[Javascript Component](#): 클라이언트 Javascript 참조입니다.

가장 일반적인 용도

- **RPC**
 - [Simple RPC](#)
 - [RPC Progress Results](#)
- **PubSub**
 - [Subscribers](#)
 - [Publishers](#)

Browser Test

즐거 사용하는 웹 브라우저로 이 프로토콜을 테스트하려면 이 URL을 입력하십시오(사용자 지정 host 및 포트를 정의해야 함)

<http://host:port/wamp.esegece.com.html>

TsgcWSPServer_WAMP

이것은 Server Protocol WAMP 구성 요소입니다.

소개

이 것은 Server Protocol WAMP Component 입 니 다. 이 구 성 요 소 를 품 에 놓 고 Server 속 성 을 사 용 하 여 [TsgcWebSocketServer](#) 구성 요소를 선택해야 합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 4개의 속성입니다.
- 메서드 — 4개의 public 메서드.
- 이벤트 — 처리할 수 있는 14개의 이벤트.

TsgcWSPClient_WAMP

이것은 Client Protocol WAMP 구성 요소입니다.

소개

이것은 클라이언트 프로토콜 WAMP 구성 요소입니다. 이 구성 요소를 폼에 넣고 Client 속성을 사용하여 [TsgcWebSocketClient](#) 구성 요소를 선택해야 합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 3개의 속성.
- 메서드 — 7개의 public 메서드.
- 이벤트 — 처리할 수 있는 13개의 이벤트.

Protocol WAMP Javascript

여기에서 사용 가능한 메서드를 찾을 수 있습니다. 필요에 따라 `{%host%}` 및 `{%port%}` 변수를 교체해야 합니다. 예제: sgcWebSocket 서버가 `www.example.com` 웹사이트에서 포트 80을 수신하도록 구성한 경우 다음을 구성해야 합니다:

```
<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>
```

TopicURI 구독

```
<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 from a 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>
```

Publish message

```
<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>
```

수신된 메시지로 Alert 표시

```
<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>
```

OnEvent 시 Alert 표시

```
<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>
```

OnConnect, OnDisconnect, OnError 이벤트에서 Alert 표시

```
<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>
```

Get Connection Status

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  socket.state();
</script>
```

WAMP | Subscribers

구독자는 먼저 관심 있는 토픽(채널)을 제공하여 이벤트를 받습니다. 이후 구독자는 해당 토픽에 게시된 모든 이벤트를 받습니다.
토픽에서 이벤트를 수신하려면, 구독자는 먼저 해당 토픽을 구독해야 합니다.

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

게시자는 특정 피어가 아니라 추상 주소로서 topic(또는 채널)을 제공하여 이벤트를 보냅니다(게시합니다). Publish 메서드를 호출하고 topic 이름과 보내려는 메시지를 인수로 전달하기만 하면 됩니다. 이 메시지는 이 topic의 모든 구독자에게 전달됩니다. 참고로, 해당 topic에 메시지를 게시하기 위해 topic을 구독할 필요는 없습니다.

publish 메시지가 수신될 때 메시지가 클라이언트에 자동으로 브로드캐스트되므로 서버 측에서 구성할 것이 없습니다.

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 | Simple RPC

WAMP 구성 요소의 가장 일반적인 용도는 클라이언트가 서버에 메서드를 요청하고, 서버가 클라이언트에 응답을 보내는 것입니다. 클라이언트는 메서드 이름만 보낼 수 있고/있거나 서버가 결과를 계산하는 데 필요한 일부 매개변수를 전달할 수 있습니다. 서버는 요청을 처리하고 성공하면 결과와 함께 응답을 클라이언트에 보냅니다. 오류가 있으면 서버는 오류 응답을 클라이언트에 보냅니다.

보시다시피, 하나의 요청과 하나의 응답(성공 또는 실패)만 있습니다.

예: 서버에 **GetTime**이라는 메서드가 있어 클라이언트가 이 메서드를 요청할 때마다 서버가 서버 시간을 반환합니다.

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 | RPC Progress Results

때때로 Remote Procedure Call은 요청을 완료하기 위해 둘 이상의 결과가 필요합니다. 기본적으로 WAMP 1.0 프로토콜은 호출에서 Partial 결과를 허용하지 않습니다. 이것은 sgcWebSockets 라이브러리 전용 기능입니다.

흐름은 단순한 RPC와 매우 유사하지만, 여기서는 프로세스를 완료하기 위해 CallResult가 호출되기 전에 1개 이상의 부분 결과가 있습니다.

기본적으로 클라이언트는 서버로부터 프로시저를 요청하고, 서버는 결과 또는 오류를 보낼 수 있습니다. 결과를 보내는 경우, 이것은 최종 결과이거나 나중에 더 많은 결과를 보내야 하는 경우일 수 있습니다. 최종 결과인 경우 **CallResult** 메서드를 호출하고 프로세스가 완료됩니다. 보낼 결과가 더 있는 경우 **CallProgressResult** 메서드를 호출합니다.

예제: 클라이언트가 서버에 매초 서버 시간을 받고 20개의 메시지 후 중지하는 메서드를 요청합니다.

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("yyyymmdd hh:nn:ss", Now));
            }
            else
            {
                oServerWAMP->CallProgressiveResult(CallId, FormatDateTime("yyyymmdd hh:nn:ss", Now));
            }
        }
    }
    else
    {
        oServer->WAMP->CallError(CallId, "Unknown method");
    }
}

oServer = new TsgcWebSocketServer();
oServer->Port = 80;
oServerWAMP = new TsgcWSPServer_WAMP();
oServerWAMP->OnCall = OnServerCallEvent();
oServerWAMP->Server = oServer;
oServer->Active = true;
```

WAMP Client

```
void OnCallResultClient(TsgcWSConnection *Connection, string CallId, string Result);
{
    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");
```

프로토콜 WAMP 2

WAMP는 다양한 언어와 함께 작동하는 개방형 WebSocket 프로토콜에서 통합 애플리케이션 라우팅을 제공합니다.

WAMP를 사용하면 느슨하게 결합되고 (소프트) 실시간으로 통신하는 애플리케이션 구성 요소로 분산 시스템을 구축할 수 있습니다.

WAMP는 핵심적으로 애플리케이션 구성 요소가 서로 통신할 수 있는 두 가지 통신 패턴을 제공합니다:

- 게시 및 구독(PubSub)
- Remote Procedure Calls (RPC)

WAMP은 사용하기 쉽고, 구현하기 간단하며, 최신 웹 표준(WebSocket, JSON 및 URI)을 기반으로 합니다.

구성 요소

[TsgcWSPClient_WAMP2](#): Client Protocol WAMP2 VCL Component.

TsgcWSPClient_WAMP2

이것은 Client Protocol WAMP2 구성 요소입니다.

소개

이것은 Client Protocol WAMP2 구성 요소입니다. 이 구성 요소를 폼에 드롭하고 Client 속성을 사용하여 [TsgcWebSocketClient](#) 구성 요소를 선택해야 합니다.

Session 메서드

- ABORT:** Router와 Client 모두 ABORT 메시지를 보내어 WAMP 세션 열기를 중단할 수 있습니다.
Reason은 URI여야 합니다.
Details는 추가적인 선택적 종료 정보를 제공할 수 있는 dictionary여야 합니다(아래 참조).
 ABORT 메시지에 대한 응답은 예상되지 않습니다.
- GOODBYE:** WAMP 세션은 Router가 Client에 WELCOME 메시지를 보내는 것으로 수명을 시작하고, 기본 전송이 사라지거나 한 Peer가 보낸 GOODBYE 메시지와 그에 대한 응답으로 다른 Peer가 보낸 GOODBYE 메시지에 의해 WAMP 세션이 명시적으로 닫힐 때 종료됩니다.
Reason은 URI여야 합니다.
Details는 추가적인 선택적 종료 정보를 제공할 수 있는 사전이어야 합니다.

Publish/Subscribe Methods

- PUBLISH:** Publisher가 어떤 토픽에 이벤트를 게시하도록 요청할 때, Broker에 PUBLISH 메시지를 보냅니다.
Request는 게시자(Publisher)가 선택한 임의의 임시 ID로, Broker의 응답을 요청과 상관시키는 데 사용됩니다.
Options는 확장 가능한 방식으로 추가 게시 요청 세부 정보를 제공할 수 있는 딕셔너리입니다. 이에 대해서는 아래에서 자세히 설명합니다.
Topic은 게시된 topic입니다.
Arguments는 애플리케이션 수준 이벤트 페이로드 요소의 목록입니다. 목록은 길이가 0일 수 있습니다.
ArgumentsKw는 키워드 인수로 제공되는 애플리케이션 수준 이벤트 페이로드를 담은 선택적 사전입니다. 사전은 비어 있을 수 있습니다.
 브로커가 게시를 충족하고 허용할 수 있는 경우, 브로커는 게시된 이벤트의 토픽의 모든 현재 구독자에게 이벤트를 보냅니다.
 기본적으로 게시는 확인되지 않으며, 게시가 실제로 성공했는지 여부에 관계없이 브로커는 응답하지 않습니다.
- SUBSCRIBE:** Subscriber는 SUBSCRIBE 메시지를 보내 토픽에 대한 관심을 Broker에 전달합니다.
Request는 Subscriber가 선택한 무작위 임시 ID여야 하며 브로커의 응답을 요청과 연관시키는 데 사용됩니다.
Options는 확장 가능한 방식으로 추가 구독 요청 세부 정보를 제공할 수 있는 딕셔너리여야 합니다.
Topic은 구독자가 구독하려는 토픽이며 반드시 URI여야 합니다.
- UNSUBSCRIBE:** Subscriber가 더 이상 구독에 대한 이벤트 수신에 관심이 없을 때 UNSUBSCRIBE 메시지를 보냅니다.
Request는 Subscriber가 선택한 무작위 임시 ID여야 하며 브로커의 응답을 요청과 연관시키는 데 사용됩니다.

SUBSCRIBED.Subscription은 구독 취소할 구독의 ID여야 하며, 원래 Broker가 Subscriber에게 전달한 것입니다.

RPC Methods

- CALL:** Caller가 원격 프로시저를 호출하려고 할 때 Dealer에게 CALL 메시지를 보냅니다.

Request는 Caller가 선택한 무작위 임시 ID이며 Dealer의 응답을 요청과 연관시키는 데 사용됩니다.

Options는 확장 가능한 방식으로 추가 호출 요청 세부 정보를 제공할 수 있는 딕셔너리입니다. 이는 아래에서 더 자세히 설명합니다.

Procedure는 호출될 프로시저의 URI입니다.

Arguments는 위치 호출 인수의 목록입니다(각각 임의의 유형). 목록의 길이는 0일 수 있습니다.

ArgumentsKw는 키워드 호출 인수(각각 임의의 유형)의 사전입니다. 사전은 비어 있을 수 있습니다.
- REGISTERCALL:** Callee는 REGISTER 메시지를 보내 Dealer에게 프로시저를 구현하는 엔드포인트의 가용성을 알립니다.

Request는 Callee가 선택한 무작위의 임시 ID로, Dealer의 응답을 요청과 연관시키는 데 사용됩니다.

Options는 확장 가능한 방식으로 추가 등록 요청 세부 정보를 제공할 수 있는 사전입니다. 이에 대해서는 아래에서 자세히 설명합니다.

Procedure는 Callee가 등록하려는 프로시저입니다.
- UNREGISTERCALL:** Callee가 더 이상 등록된 절차의 구현을 제공할 의향이 없으면, Dealer에게 UNREGISTER 메시지를 보냅니다.

Request는 Callee가 선택한 무작위의 임시 ID로, Dealer의 응답을 요청과 연관시키는 데 사용됩니다.

REGISTERED.Registration은 취소할 등록의 ID이며, 원래 Dealer가 Callee에게 전달한 것입니다.
- INVOCATION:** Dealer가 호출을 수행(중재)할 수 있고 호출을 허용하면, 프로시저를 구현하는 해당 Callee에 INVOCATION 메시지를 보냅니다.

Request는 Dealer가 선택한 무작위의 임시 ID로, Callee의 응답을 요청과 연결하는 데 사용됩니다.

REGISTERED.Registration은 프로시저가 Dealer에 등록된 등록 ID입니다.

Details는 확장 가능한 방식으로 추가 호출 요청 세부 정보를 제공할 수 있는 딕셔너리입니다. 이는 아래에서 더 자세히 설명합니다.

CALL.Arguments는 Caller가 제공한 위치 호출 인수의 원본 목록입니다.

CALL.ArgumentsKw는 Caller가 제공한 키워드 호출 인수의 원래 딕셔너리입니다.
- YIELD:** Callee가 호출의 실행을 성공적으로 처리하고 완료할 수 있는 경우, Dealer에게 YIELD 메시지를 보내 응답합니다.

INVOCATION.Request는 원래 invocation 요청의 ID입니다.

Options는 추가 옵션을 제공할 수 있는 dictionary입니다.

Arguments는 위치 기반 결과 요소(각각 임의의 유형)의 목록입니다. 목록의 길이가 0일 수 있습니다.

ArgumentsKw는 키워드 결과 요소의 딕셔너리입니다(각각 임의의 유형). 딕셔너리는 비어 있을 수 있습니다.

참조

- 속성 — 구성 요소에서 사용 가능한 3개의 속성.
- 메서드 — 11개의 public 메서드.
- 이벤트 — 처리할 수 있는 20개의 이벤트.

Protocol Default

이것은 "JSONRPC 2.0" 메시지를 사용하여 구현된 기본 서브프로토콜입니다. 이 프로토콜을 사용하여 메시지를 전송할 때마다 다음 속성을 가진 JSON 객체가 생성됩니다:

jsonrpc: JSON-RPC 프로토콜의 버전을 지정하는 String입니다. 정확히 "2.0"이어야 합니다.

method: 호출할 메서드의 이름을 포함하는 String입니다. rpc라는 단어 뒤에 마침표 문자(U+002E 또는 ASCII 46)가 오는 메서드 이름은 rpc 내부 메서드 및 확장을 위해 예약되어 있으며 다른 용도로 사용해서는 안 됩니다.

params: 메서드 호출 중에 사용될 매개변수 값을 보유하는 구조화된 값입니다. 이 멤버는 생략될 수 있습니다.

id: 포함되는 경우 String, Number 또는 NULL 값을 포함해야 하는, 클라이언트가 설정한 식별자입니다. 포함되지 않으면 알림으로 간주됩니다. 값은 일반적으로 Null이 아니어야 하며[1] Number는 소수 부분을 포함해서는 안 됩니다[2]

JSON 객체 예제:

```
{"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "id": 1}
```

Features

- **Publish/subscribe** 메시지 패턴은 일대다 메시지 분배와 애플리케이션의 분리를 제공합니다. Wildcard 문자를 지원하므로, 채널의 계층 구조를 구독할 수 있습니다. 예: 'news'로 시작하는 모든 채널을 구독하려면 Subscribe('news*')를 호출하십시오.
- 페이로드의 내용에 **독립적인** 메시징 전송
- **Acknowledgment** 전송된 메시지의 확인.
- 서버 로컬 트랜잭션을 통해 **트랜잭션 메시지**를 지원합니다. 클라이언트가 트랜잭션을 커밋하면 서버가 큐에 추가된 모든 메시지를 처리합니다. 클라이언트가 트랜잭션을 롤백하면 모든 메시지가 삭제됩니다.
- 메시지 전달을 위한 **QoS** (Quality of Service)를 구현합니다.

구성 요소

[TsgcWSPClient_sgc](#): 서버 프로토콜 기본 VCL 구성 요소.

[TsgcWSPClient_sgc](#): Client Protocol Default VCL Component.

[Javascript 구성 요소](#): 클라이언트 Javascript 참조.

Browser Test

즐거 사용하는 웹 브라우저로 이 프로토콜을 테스트하려면 이 URL을 입력하십시오(사용자 지정 host 및 포트를 정의해야 함)

<http://host:port/esegece.com.html>

TsgcWSPServer_sgc

이것은 Server Protocol Default Component입니다.

소개

이것은 Server Protocol Default 구성 요소입니다. 이 구성 요소를 폼에 놓고 Server 속성을 사용하여 [TsgcWebSocketServer](#) 구성 요소를 선택해야 합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 6개의 속성.
- 메서드 — 6개의 public 메서드.
- 이벤트 — 처리할 수 있는 16개의 이벤트.

TsgcWSPClient_sgc

이것은 *Client Protocol Default* 구성 요소입니다.

소개

이것은 클라이언트 프로토콜 기본 구성 요소입니다. 폼에 이 구성 요소를 놓고 Client 속성을 사용하여 [TsgcWebSocketClient](#) 구성 요소를 선택해야 합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 5개의 속성.
- 메서드 — 11개의 public 메서드.
- 이벤트 — 처리할 수 있는 15개의 이벤트.

TsgclWWSPClient_sgc

이것은 Inraweb Client Protocol Default Component입니다.

소개

이것은 Inraweb Client Protocol Default 구성 요소입니다. 이 구성 요소를 form에 놓고 Client 속성을 사용하여 [TsgclWWebSocketClient](#) 구성 요소를 선택해야 합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 2개의 속성.
- Methods — 3개의 public 메서드.
- 이벤트 — 처리할 수 있는 6개의 이벤트입니다.

Protocol Default Javascript

기본 프로토콜 Javascript sgcWebSockets는 **sgcWebSocket.js** 및 **esegece.com.js** 파일을 사용합니다.

여기에서 사용 가능한 메서드를 찾을 수 있습니다. 필요에 따라 `{%host%}` 및 `{%port%}` 변수를 교체해야 합니다. 예제: sgcWebSocket 서버가 `www.example.com` 웹사이트에서 포트 80을 수신하도록 구성한 경우 다음을 구성해야 합니다:

```
<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>
```

메시지 보내기

```
<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>
```

수신된 메시지로 Alert 표시

```
<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>
```

Publish Message to test channel

```
<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>
```

Event Message Received와 함께 Alert 표시

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcs('ws://{%host%}:{%port%}');
  socket.on('sgcevent', function(event)
  {
    alert('channel:' + event.channel + '. message: ' + event.message);
  }
</script>
```

Call RPC

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcs('ws://{%host%}:{%port%}');
  var params = {param:10};
  socket.rpc(GUID(), 'test', JSON.stringify(params));
</script>
```

RPC 응답 처리

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcs('ws://{%host%}:{%port%}');
  socket.on('sgcrpcresult', function(event)
  {
    alert('result:' + event.result);
  }
  socket.on('sgcrpcerror', function(event)
  {
    alert('error:' + event.code + ' ' + event.message);
  }
</script>
```

Call Notify

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcs('ws://{%host%}:{%port%}');
  var params = {param:10};
  socket.notify('test', JSON.stringify(params));
</script>
```

Send Messages in a Transaction

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcs('ws://{%host%}:{%port%}');

  socket.starttransaction('sgc:test');
  socket.publish('Message1', 'sgc:test');
  socket.publish('Message2', 'sgc:test');
```

```
socket.publish('Message3', 'sgc:test');
socket.commit('sgc:test');
</script>
```

채널에 대한 OnSubscribe 또는 OnUnSubscribe 시 Alert 표시

```
<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>
```

OnConnect, OnDisconnect, OnError 이벤트에서 Alert 표시

```
<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>
```

Get Connection Status

```
<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>
```

Protocol Dataset

이 프로토콜은 Protocol Default에서 상속되며, 이 프로토콜에 연결된 클라이언트에 데이터셋 변경 사항을 브로드캐스트하려는 경우 유용합니다. 2가지 모드로 사용할 수 있습니다:

1. 데이터베이스 복제: 데이터베이스 변경 사항이 모든 클라이언트 데이터베이스에 복제됩니다. 예: 서버에 주식 시세 데이터베이스가 있고 연결된 모든 클라이언트가 시세 변경 사항을 수신합니다. 단일 데이터베이스(서버에 있음)가 있고 모든 클라이언트는 자체 데이터베이스를 가집니다. 시세가 업데이트될 때마다 이 변경 사항은 연결된 모든 클라이언트에 브로드캐스트되고 모든 클라이언트는 자체 레코드 데이터베이스를 업데이트합니다. 이 모드 유형에는 UpdateMode: upWhereAll 또는 upWhereChanged를 사용하십시오.

2. 데이터베이스 업데이트: 여기에서는 서버와 클라이언트가 공유하는 단일 데이터베이스가 있으며, 클라이언트가 데이터베이스의 레코드를 업데이트할 때마다 다른 모든 클라이언트가 이 업데이트에 대해 알림을 받기를 원합니다. 이 모드에는 UpdateMode: upRefreshAll을 사용하십시오.

Most common uses

- **Update Mode**
 - [테이블 복제 방법](#)
 - [Notify Updates 방법](#)

"JSON-RPC 2.0" 객체를 사용하며, 데이터셋 변경이 있을 때마다 Dataset 객체를 사용하여 모든 필드 값(* 지원되는 필드만)을 보냅니다.

구성 요소가 데이터셋에서 레코드를 검색할 수 있도록 하려면 어떤 필드가 Key인지 지정해야 합니다. 예: 데이터셋에서 ID 필드가 키이면 다음과 같은 코드를 작성해야 합니다

```
void OnAfterOpenDataSet(TDataSet *DataSet)
{
    DataSet->FieldByName("ID")->ProviderFlags =
        DataSet->FieldByName("ID")->ProviderFlags + [pfInKey];
}
```

구성 요소

[TsgcWSPServer_Dataset](#): Server Protocol Dataset VCL 구성 요소.

[TsgcWSPClient_Dataset](#): 클라이언트 프로토콜 Dataset VCL 구성 요소입니다.

[Javascript Component](#): 클라이언트 Javascript 참조.

Browser Test

즐거 사용하는 웹 브라우저로 이 프로토콜을 테스트하려면 이 URL을 입력하십시오(사용자 지정 host 및 포트를 정의해야 함)

`http://host:port/dataset.esegece.com.html`

TsgcWSPServer_Dataset

이것은 Server Protocol Dataset 구성 요소입니다.

소개

이것은 Server Protocol Dataset 구성 요소입니다. 이 구성 요소를 폼에 드롭하고 Server 속성을 사용하여 [TsgcWebSocketServer](#) 구성 요소를 선택하고 Dataset 속성을 사용하여 Dataset 구성 요소를 선택해야 합니다.

이 구성 요소는 [TsgcWSProtocol_Server_sgc](#)에서 모든 메서드와 속성을 상속합니다.

참조

- 속성 — 구성 요소에서 사용할 수 있는 13개의 속성.
- 메서드 — 9개의 public 메서드.
- 이벤트 — 처리할 수 있는 22개의 이벤트.

TsgcWSPClient_Dataset

이것은 Client Protocol Dataset 구성 요소입니다.

소개

이것은 Client Protocol Dataset 구성 요소입니다. 이 구성 요소를 폼에 놓고 Client 속성을 사용하여 [TsgcWebSocketClient](#) 구성 요소를 선택하고 Dataset 속성을 사용하여 Dataset 구성 요소를 선택해야 합니다.

이 구성 요소는 [TsgcWSProtocol_Client_sgc](#)에서 모든 메서드와 속성을 상속합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 13개의 속성.
- 메서드 — 14개의 public 메서드.
- 이벤트 — 처리할 수 있는 25개의 이벤트입니다.

TsgclWWSPClient_Dataset

이것은 Inraweb Client Protocol Dataset 구성 요소입니다.

소개

이것은 Inraweb Client Protocol Dataset 구성 요소입니다. 이 구성 요소를 폼에 드롭하고 Client 속성을 사용하여 [TsgclWWebSocketClient](#) 구성 요소를 선택하고 Dataset 속성을 사용하여 Dataset 구성 요소를 선택해야 합니다.

이 구성 요소는 [TsgclWWSPClient_sgc](#)로부터 모든 메서드와 속성을 상속합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 3개의 속성.
- 메서드 — 6개의 public 메서드.
- 이벤트 — 처리할 수 있는 9개의 이벤트.

Protocol Dataset Javascript

Dataset Protocol Javascript sgcWebSockets는 **sgcWebSocket.js** 및 **dataset.esegece.com.js** 파일을 사용합니다.

여기에서 사용 가능한 메서드를 찾을 수 있습니다. 필요에 따라 `{%host%}` 및 `{%port%}` 변수를 교체해야 합니다. 예제: sgcWebSocket 서버가 `www.example.com` 웹사이트에서 포트 80을 수신하도록 구성한 경우 다음을 구성해야 합니다:

```
<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>
```

메시지 보내기

```
<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>
```

수신된 메시지로 Alert 표시

```
<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>
```

수신된 Dataset과 함께 Alert 표시

```
<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>
```

채널에 대한 OnSubscribe 또는 OnUnSubscribe 시 Alert 표시

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
  var socket = new sgcs_dataset('ws://{%host%}:{%port%}');
  socket.on('sgcsubscribe', function(event)
  {
    alert('subscribed: ' + event.channel);
  }
  socket.on('sgcunsubscribe', function(event)
  {
    alert('unsubscribed: ' + event.channel);
  }
</script>
```

OnConnect, OnDisconnect, OnError 이벤트에서 Alert 표시

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
  var socket = new sgcs_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>
```

모든 데이터셋 변경 구독

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
  socket.subscribe_all();
</script>
```

모든 Dataset 변경 구독 취소

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
  socket.unsubscribe_all();
</script>
```

데이터셋 변경 처리

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
```

```

<script>
var socket = new sgcdataset('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>

```

Get Connection Status

```

<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
    socket.state();
</script>

```

Protocol Dataset | Replicate Table

이 모드는 모든 연결된 클라이언트에 대해 테이블이 복제되는 일반적인 시나리오를 해결하려고 합니다. 예제로, 주식 시세 테이블이 있는 서버가 있고 모든 클라이언트에게 주식 변경 사항을 브로드캐스트하고 싶지만, 클라이언트가 데이터베이스에 연결할 수 없게 하고 싶다면, 어떤 주식 시세에 변경이 있을 때마다 레코드 정보가 브로드캐스트되어 모든 연결된 클라이언트에게 전달됩니다. 모든 클라이언트는 레코드를 읽고 자신의 테이블을 업데이트합니다.

Demos 폴더의 SQLite/MultipleDatabase 데모에서 확인할 수 있습니다.

Configure Dataset Server

새 Dataset Protocol Server를 생성하고 다음 속성을 사용하여 구성하십시오

- **ApplyUpdates: True**로 설정하면, 변경이 있을 때마다 클라이언트에 브로드캐스트됩니다.
- **AutoSynchronize: True**로 설정하면, 새 클라이언트가 서버에 연결할 때마다 서버가 모든 레코드(metadata 및 data)를 보내므로, 클라이언트가 서버에서 최신 정보를 얻습니다.
- **UpdateMode: upWhereAll** 또는 **upWhereChanged**로 설정하십시오. 차이점은 첫 번째는 레코드의 모든 필드를 전송하고 두 번째는 업데이트에서 변경된 필드만 전송한다는 것입니다.

```
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

새 Dataset Protocol Client를 생성하고 다음 속성을 사용하여 구성하십시오.

- **ApplyUpdates: True**로 설정하면 변경이 있을 때마다 서버로 전송됩니다.
- **AutoSubscribe: True**로 설정하면, 새 클라이언트가 서버에 연결할 때마다 클라이언트가 update, delete 및 new record를 자동으로 구독합니다.
- **UpdateMode: upWhereAll** 또는 **upWhereChanged**로 설정하십시오. 차이점은 첫 번째는 레코드의 모든 필드를 전송하고 두 번째는 업데이트에서 변경된 필드만 전송한다는 것입니다.

```
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

이 모드는 서버와 클라이언트가 단일 데이터베이스를 공유하고(서버와 클라이언트가 동일한 물리적 데이터베이스에 연결됨) 클라이언트가 다른 클라이언트가 dataset에 변경을 수행할 때마다 알림을 받기를 원하는 시나리오를 해결하려고 합니다.

Demos 폴더의 SQLite/SingleDatabase 데모에서 확인할 수 있습니다.

Configure Dataset Server

새 Dataset Protocol Server를 생성하고 다음 속성을 사용하여 구성하십시오.

- **ApplyUpdates: True**로 설정하면, 변경이 있을 때마다 클라이언트에 브로드캐스트됩니다.
- **AutoSynchronize: False**로 설정하십시오. 클라이언트가 서버와 동일한 데이터베이스에 연결되어 있으므로 여기서는 true로 설정할 필요가 없습니다.
- **UpdateMode: upRefreshAll**로 설정하십시오.

```
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

새 Dataset Protocol Client를 생성하고 다음 속성을 사용하여 구성하십시오.

- **ApplyUpdates: True**로 설정하면 변경이 있을 때마다 서버로 전송됩니다.
- **AutoSubscribe: True**로 설정하면, 새 클라이언트가 서버에 연결할 때마다 클라이언트가 update, delete 및 new record를 자동으로 구독합니다.
- **UpdateMode: upRefreshAll**로 설정하십시오.

```
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

이 프로토콜은 바이너리 WebSocket 전송을 사용하여 파일을 보낼 수 있게 합니다. 낮은 메모리 사용량으로 큰 파일을 처리할 수 있습니다.

Features

- **Publish/subscribe** 메시지 패턴은 일대다 메시지 분배와 애플리케이션의 분리를 제공합니다.
- **Acknowledgment** 전송된 메시지의 확인.
- 파일 전달을 위한 **QoS** (Quality of Service)를 구현합니다.
- 선택적으로 받은 파일에 대해 **Authorization** 을 요청할 수 있습니다.
- 낮은 메모리 사용량.

구성 요소

[TsgcWSPServer_Files](#): Server Protocol Files VCL 구성 요소.

[TsgcWSPClient_Files](#): Client Protocol Files VCL Component.

Classes

[TsgcWSMessageFile](#): 파일 패킷 정보를 캡슐화하는 객체입니다.

Most common uses

- 파일 보내기
 - [How Send Files To Server](#)
 - 클라이언트에게 파일을 보내는 방법
- 대용량 파일
 - [큰 파일을 보내는 방법](#)

TsgcWSPServer_Files

이것은 Server Files Protocol 구성 요소입니다.

소개

이것은 Server Files Protocol 구성 요소입니다. 이 구성 요소를 폼에 놓고 Server 속성을 사용하여 [TsgcWebSocketServer](#) 구성 요소를 선택해야 합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 5개의 속성.
- 메서드 — 3개의 공개 메서드.
- Events — 처리할 수 있는 20개의 이벤트.

TsgcWSPClient_Files

이것은 Client Files Protocol 구성 요소입니다.

소개

이것은 Client Files Protocol 구성 요소입니다. 이 구성 요소를 폼에 놓고 Client 속성을 사용하여 [TsgcWebSocketClient](#) 구성 요소를 선택해야 합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 5개의 속성.
- 메서드 — 4개의 public 메서드.
- 이벤트 — 처리할 수 있는 19개의 이벤트입니다.

TsgcWSMessageFile

이 객체는 파일 프로토콜 이벤트가 발생할 때마다 매개변수로 전달됩니다.

Properties

- **BufferSize:** 패킷의 기본 크기입니다.
- **Channel:** 지정된 경우 이 파일은 특정 채널을 구독한 클라이언트에만 전송됩니다.
- **Method:** 내부 메서드입니다.
- **FileId:** 파일의 식별자. 수신/송신된 모든 파일에 대해 고유합니다.
- **Data:** 사용자 사용자 지정 데이터입니다. 여기에서 사용자는 원하는 텍스트를 설정할 수 있습니다.
- **FileName:** 파일의 이름입니다.
- **FilePosition:** 파일 위치(바이트)입니다.
- **FileSize:** 바이트 단위의 전체 파일 크기.
- **Id:** 패킷의 식별자입니다. 모든 패킷에 대해 고유합니다.
- **QoS:** 메시지의 서비스 품질입니다.
- **Streaming:** 내부용입니다.
- **Text:** 내부 사용을 위한 것입니다.

Protocol Files | How Send Files To Server

서버에 파일을 보내려면 Files Protocol의 **SendFile** 메서드를 호출하고 전체 **FileName**을 인수로 전달하기만 하면 됩니다. 서버가 수신한 파일은 기본적으로 서버 실행 파일이 위치한 동일한 디렉터리 또는 Files.SaveDirectory 속성에 설정된 Path에 저장됩니다.

```
// ... 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 | How Send Files To Clients

클라이언트에 파일을 보내려면 Files 프로토콜의 **SendFile** 메서드를 호출하고 연결의 **Guid**와 전체 **FileName**을 인수로 전달하기만 하면 됩니다. 클라이언트 연결의 Guid는 Server Protocol Files의 OnConnect 이벤트에서 캡처할 수 있습니다. 클라이언트가 수신한 파일은 기본적으로 클라이언트 실행 파일이 있는 동일한 디렉터리 또는 Files.SaveDirectory 속성에 설정된 Path에 저장됩니다.

```
// ... 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");
```

Protocol Files | How Send Big Files

서버나 클라이언트에 큰 파일, 예를 들어 수 기가바이트의 파일을 보내려고 할 때 전체 파일을 로드하려고 하면 메모리 문제가 발생할 수 있습니다. Protocol Files를 사용하면 다른 피어가 수신하면 단일 파일로 재조립되는 더 작은 패킷으로 파일을 보낼 수 있습니다. **SendFile** 메서드의 **Size** 매개변수를 사용하여 각 단일 패킷의 크기(바이트)를 설정하기만 하면 됩니다.

```
// ... 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

Presence 프로토콜을 사용하면 누가 채널을 구독하고 있는지 알 수 있으므로 채팅 애플리케이션을 더 쉽게 만들고 누가 온라인인지 알 수 있습니다. 예: 게임 사용자, 채팅방, 동일한 문서를 보고 있는 사용자...

Features

- 기본적으로 사용자는 **이름**으로 식별되지만, 이는 더 많은 데이터(email, company, twitter...)를 전달하여 **사용자 정의**할 수 있습니다.
- **Channel**을 생성할 수 있는지, **member**가 허용되는지 등을 **Authorize**하는 이벤트입니다...
- 새 **멤버가 가입**할 때마다 채널의 모든 멤버에게 **알림**이 전송됩니다.
- 모든 채널 구독자에게 **메시지**를 게시합니다.
- 낮은 메모리 사용량.

구성 요소

[TsgcWSPServer_Presence](#): Server Protocol Presence VCL 구성 요소.

[TsgcWSPClient_Presence](#): Client Protocol Presence VCL 구성 요소.

Classes

[TsgcWSPresenceMessage](#): presence 패킷 정보를 캡슐화하는 객체입니다.

TsgcWSPServer_Presence

이것은 Server Presence Protocol Component입니다.

소개

이것은 Server Presence Protocol 구성 요소입니다. 이 구성 요소를 form에 놓고 Server 속성을 사용하여 [TsgcWebSocketServer](#) 구성 요소를 선택해야 합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 6개의 속성.
- 메서드 — 1개의 public 메서드.
- 이벤트 — 처리할 수 있는 17개의 이벤트.

TsgcWSPresenceMessage

이 객체는 서버와 클라이언트 presence 프로토콜이 교환하는 모든 내부 메시지를 캡슐화합니다.

TsgcWSPresenceMember

ID: 내부 식별자

Name: 멤버 이름, 클라이언트가 제공합니다.

Info: 클라이언트가 제공한 멤버 추가 정보.

Data: 서버 목적으로 사용할 수 있는 TObject입니다.

TsgcWSPresenceMemberList

Member[i]: index별 목록의 멤버

Count: 목록의 멤버 수입니다

TsgcWSPresenceChannel

Name: 클라이언트가 제공한 채널 이름입니다.

TsgcWSPresenceMsg

Text: 텍스트 메시지로, 클라이언트가 Publish 메서드를 호출할 때 제공됩니다.

TsgcWSPresenceError

Code: 오류를 식별하는 정수 값

Text: 오류 설명입니다.

TsgcWSPClient_Presence

이것은 Client Presence Protocol 구성 요소입니다.

소개

이것은 Client Presence Protocol 구성 요소입니다. 이 구성 요소를 폼에 드롭하고 Client 속성을 사용하여 [TsgcWebSocketClient](#) 구성 요소를 선택해야 합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 7개의 속성.
- 메서드 — 5개의 public 메서드.
- Events — 처리할 수 있는 16개의 이벤트.

Protocol Presence Javascript

Presence Protocol Javascript sgcWebSockets는 **sgcWebSocket.js** 및 **presence.esegece.com.js** 파일을 사용합니다.

여기에서 사용 가능한 메서드를 찾을 수 있습니다. 필요에 따라 `{%host%}` 및 `{%port%}` 변수를 교체해야 합니다. 예: sgcWebSocket 서버를 `www.example.com` 웹사이트에서 포트 80에서 수신하도록 구성한 경우 다음과 같이 구성해야 합니다:

```
<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>
```

연결 후 새 멤버

```
<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>
```

Topic 1 채널 구독

```
<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>
```

Topic 1 채널에서 구독 해제

```
<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>
```

Topic 1 채널에 메시지 게시

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegce.com.js"></script>
<script>
  var socket = new sgcsws_presence('ws://{%host%}:{%port%}');
  socket.publish('Hello sgcWebSockets!', 'Topic 1');
</script>
```

Receive Message

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegce.com.js"></script>
<script>
  var socket = new sgcsws_presence('ws://{%host%}:{%port%}');
  socket.on('sgcpublishmsg', function(event)
  {
    console.log('#publishmsg: ' + event.message.text);
  });
</script>
```

연결된 모든 멤버 가져오기

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegce.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>
```

멤버가 구독/구독 취소할 때 경고 표시

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegce.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>
```

OnConnect, OnDisconnect, OnError 이벤트에서 Alert 표시

```
<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

종단간 암호화(E2EE)는 메시지가 발신자 장치에서 암호화되고 수신자 장치에서만 복호화될 수 있음을 의미합니다. 서버는 패킷을 라우팅하지만 평문 콘텐츠를 읽을 수 없습니다.

이 주제에서는 다음에 대한 기술적 흐름을 설명합니다:

- 두 피어 간의 **직접 메시지(1:1)**입니다.
- 멤버십 기반 키 회전을 사용하는 **sender-key** 모델을 통해 **그룹 메시지**를 전송합니다.

암호화 구성 요소

- **신원 / 키 합의:** Elliptic Curve Diffie-Hellman (ECDH) P-256.
- **대칭 암호화:** AES-256-GCM.
- **키 도출:** 공유 비밀에서 암호화 키를 도출하기 위한 HKDF-SHA-256입니다.
- **Message authentication:** AEAD 인증 태그(AES-GCM에서 제공).
- **Replay protection:** 메시지별 nonce/IV와 송신자 시퀀스 카운터입니다.

다이렉트 메시지(1:1) 기술 흐름

1. **공개 키 검색**
각 피어는 공개 ID 키를 게시합니다. 서버는 공개 키를 배포할 수 있지만 개인 키는 클라이언트 장치를 절대 떠나지 않습니다.
2. **공유 시크릿 설정**
발신자와 수신자가 ECDH(개인 키 + 피어 공개 키)를 수행하고 동일한 공유 시크릿을 얻습니다.
3. **세션 키 파생**
HKDF-SHA-256은 ECDH 출력에서 하나 이상의 대칭 키(암호화 키, 선택적 헤더 키)를 파생합니다.
4. **메시지 암호화**
평문은 고유한 nonce/IV를 사용하여 AES-256-GCM으로 암호화됩니다. 출력에는 ciphertext + 인증 태그가 포함됩니다.
5. **전송**
서버는 평문 액세스 없이 암호화된 페이로드와 메타데이터(예: 발신자 id, 키 id, 카운터, 타임스탬프)를 전달합니다.
6. **수신자 복호화**
수신자는 동일한 세션 키를 파생하고, 인증 태그를 검증하고, 복호화합니다. 변조가 있으면 인증 실패가 발생합니다.

그룹 메시지 기술 흐름(Sender Keys)

그룹의 경우 모든 멤버에 대해 각 메시지를 개별적으로 암호화하는 것은 비용이 많이 듭니다. 일반적인 최적화는 **sender key** 설계입니다:

- 각 발신자는 그룹별로 **Sender Key State**를 유지합니다.
- 상태는 대칭 **chain key**와 메시지 카운터를 포함합니다.
- 보내는 모든 그룹 메시지에 대해 발신자는 체인 키에서 일회용 메시지 키를 파생하고 체인을 진행합니다(해시 래치).
- 메시지 키는 AES-256-GCM으로 페이로드를 암호화합니다.

sender 키가 배포되는 방법

1. 송신자가 그룹에 조인하면 새로운 송신자 키 상태를 생성합니다.
2. 발신자 키 상태는 기존 1:1 암호화 세션(쌍별 E2EE)을 통해 각 현재 멤버에게 공유됩니다.
3. 배포 후, 일반 그룹 페이로드는 발신자 키 빠른 경로(메시지당 단일 암호화)를 사용합니다.

멤버십 변경 및 sender key 교체

전방 및 후방 비밀성을 유지하기 위해, 그룹 sender 키는 멤버십 이벤트 시 교체되어야 합니다:

- **사용자가 그룹에 조인:** 새 멤버가 명시적으로 공유되지 않는 한 이전 기록을 복호화할 수 없도록 sender 키를 회전하십시오.
- **사용자가 나가거나 제거됨:** 제거된 멤버가 새 트래픽을 복호화할 수 없도록 sender 키를 즉시 회전하십시오.
- **주기적 교체:** 선택적인 시간 기반 또는 메시지 수 기반 교체는 키 손상의 영향을 줄입니다.

일반적인 회전 시퀀스:

1. 새 발신자 키 상태(새 키 id, 새 체인 키)를 생성합니다.
2. 페어와이즈 E2EE 채널을 통해 현재 권한이 부여된 멤버에게만 배포하십시오.
3. 새 키 id로 새 그룹 메시지 암호화를 시작합니다.

4. 짧은 전환 윈도우 동안에만 이전 키 id를 수락한 다음 폐기합니다.

보안 속성

- 기밀성: 유효한 키가 있는 클라이언트만 복호화할 수 있습니다.
- 무결성 및 진위성: AEAD 검증은 변조 및 위조된 암호문을 감지합니다.
- 서버 무지(**blindness**): 릴레이 서버는 메시지 내용에 대한 평문 키를 보유하지 않습니다.
- 멤버십 이후 보호: 회전된 키는 이전 멤버가 향후 그룹 메시지에 접근하는 것을 차단합니다.

구성 요소

[TsgcWSPServer_E2EE](#): Server Protocol E2EE 구성 요소입니다. 메시지 내용을 알지 못한 채 클라이언트 간에 암호화된 메시지를 전달합니다.

[TsgcWSPClient_E2EE](#): Client Protocol E2EE 구성 요소. 피어 디바이스에서 키 교환, 암호화 및 복호화를 관리합니다.

TsgcWSPServer_E2EE

E2EE 서버 프로토콜 구성 요소는 페이로드를 불투명하게 유지하면서 클라이언트 간 암호화된 직접 메시지와 그룹 메시지를 라우팅합니다.

소개

E2EE 서버 프로토콜 구성 요소는 페이로드를 불투명하게 유지하면서 클라이언트 간에 암호화된 직접 및 그룹 메시지를 라우팅합니다. 폼에 놓고, [TsgcWebSocketHTTPServer](#)(또는 다른 서버 구성 요소)를 **Server** 속성에 할당한 다음, E2EE 옵션을 구성하십시오. 구현은 **sgcWebSocket_Protocol_E2EE_Server** 유닛에 있습니다.

구성

구성 요소를 사용하려면:

- **Server**를 구성된 **TsgcWebSocketHTTPServer** 인스턴스로 설정하십시오.
- **E2EE_Options.Ack**에서 확인 처리를 구성하십시오.
- **E2EE_Options.PublicKeys.ReceiveNewPublicKey**로 새 공개 키를 처리하는 방법을 선택하십시오.
- 수신 대기를 시작하려면 **Server.Active := True**를 설정하십시오.

참조

- 속성 — 구성 요소에서 사용할 수 있는 5개의 속성.
- 메서드 — 0개의 public 메서드.
- 이벤트 — 처리할 수 있는 8개의 이벤트.

TsgcWSPClient_E2EE

E2EE 클라이언트 프로토콜 구성 요소는 암호화된 직접 메시지 및 암호화된 그룹 메시지를 포함하여 WebSocket 연결을 통한 종단간 암호화를 추가합니다.

소개

E2EE 클라이언트 프로토콜 구성 요소는 암호화된 직접 메시지 및 암호화된 그룹 메시지를 포함하여 WebSocket 연결에 종단간 암호화를 추가합니다. 구성 요소를 폼에 드롭하고, [TsgcWebSocketClient](#)를 **Client** 속성에 할당한 다음, 연결하기 전에 E2EE 옵션을 구성하십시오. 구현은 **sgcWebSocket_Protocol_E2EE_Client** 유닛에 있습니다.

구성

구성 요소를 사용하려면:

- **Client**를 구성된 **TsgcWebSocketClient** 인스턴스로 설정하십시오.
- **E2EE_Options.UserId**를 사용하여 고유한 사용자 식별자를 할당하십시오.
- 전달 상태 콜백을 원할 때 **E2EE_Options.Ack.RcvDirectMessage** 및 **E2EE_Options.Ack.RcvGroupMessage**로 메시지 승인을 활성화하십시오.
- WebSocket 연결을 열려면 **Client.Active := True**로 설정하십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 5개의 속성.
- 메서드 — 9개의 public 메서드.
- 이벤트 — 처리할 수 있는 19개의 이벤트입니다.

WebSocket APIs

WebSocket을 기반으로 하는 여러 구현이 있습니다: finance, 메시지 게시, 큐... sgcWebSockets는 WebSocket 프로토콜을 기반으로 하는 가장 중요한 API를 구현합니다. API를 사용하려면 API 구성 요소를 클라이언트에 연결하기만 하면 모든 메시지가 API 구성 요소에 의해 처리됩니다(클라이언트에는 하나의 API 구성 요소만 연결할 수 있습니다).

Client APIs

API	설명
Binance	국제적인 다국어 암호화폐 거래소입니다.
Binance Futures	Binance Futures WebSocket / REST Market Streams에 연결할 수 있게 합니다.
Coinbase	Coinbase는 미국 기반 암호화 거래소입니다. Bitcoin(BTC), Ethereum(ETH) 등을 U WebSocket API 및 REST API를 지원합니다.
SignalR	은 WebSockets를 전송 수단으로 사용하여 실시간 웹 기능을 쉽게 개발할 수 있게 해주는 A
SignalRCore	ASP.NET Core SignalR은 앱에 실시간 웹 기능을 추가하는 것을 간소화하는 오픈 소스 라
SocketIO	는 실시간 웹 애플리케이션을 위한 JavaScript 라이브러리입니다. 웹 클라이언트와 서버 간
Kraken	미국 기반 cryptocurrency 거래소입니다.
Kraken Futures	Kraken Futures WebSocket / REST Market 데이터에 연결할 수 있게 합니다.
Pusher	Pusher는 유연한 pub/sub 메시징, 라이브 사용자 목록, 인증 등을 갖춘 쉽고 신뢰할 수 있
Bitfinex	Bitfinex는 세계에서 가장 크고 진보된 암호화폐 거래 플랫폼 중 하나입니다. 사용자는 Bitc
Bitstamp	Bitstamp는 2011년부터 blockchain 생태계를 지원해 온 세계에서 가장 오래된 암호화 거
Huobi	국제적인 다국어 암호화폐 거래소입니다.
Cex	는 cryptocurrency exchange이자 이전 Bitcoin 클라우드 마이닝 제공업체입니다.
Cex Plus	CEX.IO Exchange Plus는 깊은 유동성과 고급 거래 도구를 갖춘 중국의 암호화폐 거래 플
Bitmex	는 cryptocurrency exchange이자 파생상품 거래 플랫폼입니다.
3Commas	암호화폐 트레이딩 봇입니다.
Kucoin	는 BTC, ETH, KCS, SHIB, DOGE와 같은 cryptocurrency를 사고, 팔고, 저장할 수 있는 cr
Kucoin Futures	Kucoin Futures 서버(WebSocket 및 REST)에 연결할 수 있도록 합니다
OKX	이전에 OKEx로 알려졌으며, 가장 큰 암호화폐 현물 및 파생상품 거래소 중 하나입니다.
XTB	2000개 이상의 금융 시장에 대한 접근을 제공하는 FX 및 CFD 거래.
Discord	은(는) 온라인 게임 및 스트리밍을 위한 가장 인기 있는 통신 도구 중 하나입니다.
Bybit	암호화폐 거래소 및 트레이딩 플랫폼
OpenAI API	OpenAI Realtime API는 음성 대 음성 대화 경험 및 실시간 전사를 포함한 저지연 멀티모달
MEXC	중앙화된 암호화폐 거래소 및 거래 플랫폼으로, 이 구성 요소는 SPOT API를 구현합니다.
MEXC Futures	중앙화된 암호화폐 거래소 및 거래 플랫폼으로, 이 구성 요소는 Futures API를 구현합니다.
Bitget	Spot 및 Futures 시장을 지원하는 암호화폐 거래소 및 거래 플랫폼입니다.
Gate.io	WebSocket 및 REST API로 현물 및 선물 거래를 지원하는 암호화폐 거래소입니다.
Deribit	Bitcoin 및 Ethereum에 대한 선물 및 옵션 거래를 제공하는 암호화폐 파생상품 거래소.
Crypto.com	WebSocket 및 REST API와 함께 Market 및 User 채널을 지원하는 암호화폐 거래소입니다
HTX	시장 데이터 및 거래를 위한 REST API를 갖춘 국제 암호화폐 거래소(이전 Huobi).

WebSocket API는 런타임에 등록할 수 있습니다. **RegisterAPI** 메서드를 호출하고 API 구성 요소를 매개변수로 전달하기만 하면 됩니다.

Other Client APIs

API	설명
Telegram	클라우드 기반 인스턴트 메시징 및 voice over IP 서비스입니다. 사용자는 메시지를 보내고 수신하는 다양한 유형의 파일을 교환할 수 있습니다.
Whatsapp	국제적으로 이용 가능한 미국의 무료 크로스 플랫폼 중앙화된 인스턴트 메시징 및 voice-over IP 서비스입니다.
RCON	서버에 콘솔 명령을 실행할 수 있게 하는 TCP/IP 기반 통신 프로토콜입니다.
CryptoHopper	crypto 트레이딩 봇 및 포트폴리오 관리자입니다.
CryptoRobotics	암호화폐 거래 로봇입니다.

Server APIs

API	설명
RTCMultiConnection	RTCMultiConnection은 peer-to-peer 애플리케이션(화면 공유, 오디오/비디오 스트리밍 등)을 위한 WebRTC JavaScript 라이브러리입니다.
WebPush	Web Push 프로토콜을 사용하면 웹 애플리케이션이 열려 있거나 활성 상태가 아니더라도 메시지를 보낼 수 있습니다.
WebAuthn	생체 인식, 보안 키, passkey를 사용한 비밀번호 없는 인증을 위한 FIDO2/WebAuthn API입니다.

API Binance

Binance

Binance는 국제 다국어 암호화폐 거래소입니다. Binance 데이터에 액세스하기 위한 일부 API를 제공합니다. 다음 API가 지원됩니다:

1. **WebSocket streams:** 일부 메서드를 구독하여 데이터를 실시간으로 받을 수 있게 합니다. 이벤트는 서버에 의해 구독자에게 푸시됩니다. 프로토콜로 WebSocket을 사용합니다.
2. **UserData stream:** 구독한 클라이언트가 계정 세부 정보를 받습니다. 인증하려면 API 키가 필요하며 프로토콜로 WebSocket을 사용합니다.
3. **REST API:** 인증을 위해 API 키와 Secret이 필요하며 프로토콜로 HTTPs를 사용합니다.
 1. **Market Data**
 2. **Account and Trading Data**
 3. **Wallet**
4. **Futures:** WebSocket Futures Market Data Streams은 [Binance Futures Client API](#)를 통해 지원됩니다.

클라이언트는 [Binance.us](#)도 지원합니다. 다음 API가 지원됩니다:

1. **WebSocket streams:** 일부 메서드를 구독하여 데이터를 실시간으로 받을 수 있게 합니다. 이벤트는 서버에 의해 구독자에게 푸시됩니다. 프로토콜로 WebSocket을 사용합니다.
2. **UserData stream:** 구독한 클라이언트가 계정 세부 정보를 받습니다. 인증하려면 API 키가 필요하며 프로토콜로 WebSocket을 사용합니다.
3. **REST API:** 클라이언트가 서버에 시장 및 계정 데이터를 요청할 수 있습니다. 인증을 위해 API Key와 Secret이 필요하며 프로토콜로 HTTPs를 사용합니다.

Properties

Binance API에는 2가지 유형의 메서드가 있습니다: 공개 및 비공개. 공개 메서드는 인증 없이 액세스할 수 있습니다(예: 티커 가격 가져오기). 일부는 비공개이며 사용자 데이터와 관련됩니다. 이러한 메서드는 Binance API 키 사용이 필요합니다.

- **ApiKey:** binance 계정에서 새 api 키를 요청할 수 있습니다. 값을 이 속성에 복사하기만 하면 됩니다.
- **ApiSecret:** API secret은 REST_API에만 필요하며, websocket api는 일부 메서드에 대해 ApiKey만 필요합니다.
- **TestNet:** 활성화하면 Binance Demo Account에 연결합니다(기본값 false).
 - **HTTPLogOptions:** HTTP 요청 로그를 텍스트 파일에 저장합니다.
 - **Enabled:** 활성화하면 WebSocket API의 모든 HTTP 요청을 저장합니다.
 - **FileName:** 로그가 저장될 파일 이름의 전체 경로입니다.
 - **REST:** REST API 요청 로그를 텍스트 파일에 저장합니다.
 - **Enabled:** 활성화하면 REST API의 모든 HTTP 요청을 저장합니다.
 - **FileName:** 로그가 저장될 파일 이름의 전체 경로입니다.
- **UserStream:** 활성화되면 클라이언트는 Account, Orders 또는 Balance Update에 대한 알림을 받습니다(기본적으로 true).
- **BinanceUS:** 활성화되면 Binance.us 서버에 연결합니다(기본값인 Binance.com 서버 대신).
- **ListenKeyOnDisconnect:** 이 속성은 클라이언트가 활성 ListenKey를 가지고 Binance 서버에서 연결 해제할 때 수행 작업을 지정합니다.
 - **blkodDeleteListenKey:** Binance 서버에 HTTP 요청을 수행하여 활성 ListenKey를 삭제합니다(기본값).
 - **blkodClearListenKey:** Binance 서버에서 ListenKey를 삭제하지 않고 필드 값만 지웁니다.
 - **blkodDoNothing:** 아무것도 하지 않으므로, 다음에 Binance에 연결할 때 동일한 ListenKey를 사용하려고 시도합니다.
- **UseCombinedStreams:** 활성화되면 스트림을 다음과 같이 결합합니다:


```
{ "stream": "<streamName>", "data": <rawPayload> }
```

 (기본적으로 비활성화됨)

Most common uses

- **WebSockets API**
 - [WebSocket API에 연결하는 방법](#)

- [WebSocket 채널을 구독하는 방법](#)
- **REST API**
 - [시장 데이터를 가져오는 방법](#)
 - [Private REST API를 사용하는 방법](#)
 - [Spot 거래 방법](#)
 - [Private Requests Time](#)
 - [Withdraw](#)

WebSocket Stream API

기본 엔드포인트는 wss://stream.binance.com:9443이며, 클라이언트는 성공적으로 연결된 후 이벤트를 구독/구독 해제할 수 있습니다.

다음 Subscription / Unsubscription 메서드가 지원됩니다.

Method	매개변수	설명
AggregateTrades	Symbol	단일 taker 주문에 대해 집계된 거래 정보를 푸시합니다
Trades	Symbol	원시 거래 정보를 푸시합니다. 각 거래에는 고유한 구매자와 판매자가 있습니다
KLine	Symbol, Interval	매초, 매분, 매시간마다 현재 klines/캔들스틱에 업데이트를 푸시합니다...
MiniTicker	Symbol	24시간 롤링 윈도우 미니 티커 통계입니다. 이는 UTC 일자의 통계가 아니라 이전 24시간에 대한 24시간 롤링 윈도우입니다.
AllMiniTickers		배열로 변경된 모든 심볼에 대한 24시간 롤링 윈도우 미니 시세 통계입니다. 이것은 UTC 일자의 통계가 아니라 이전 24시간에 대한 24시간 롤링 윈도우입니다. 변경된 시세만 배열에 존재합니다.
Ticker	Symbol	단일 심볼에 대한 24시간 롤링 윈도우 티커 통계입니다. 이것은 UTC 일자의 통계가 아니라 이전 24시간에 대한 24시간 롤링 윈도우입니다.
AllMarketTickers		배열로 변경된 모든 심볼에 대한 24시간 롤링 윈도우 티커 통계입니다. 이것은 UTC 일자의 통계가 아니라 이전 24시간에 대한 24시간 롤링 윈도우입니다. 변경된 티커만 배열에 존재함에 유의하십시오.
BookTicker	Symbol	지정된 symbol에 대한 최우선 매수 또는 매도 호가의 가격 또는 수량에 대한 모든 업데이트를 실시간으로 푸시합니다.
AllBookTickers		모든 심볼에 대해 최고 매수 또는 매도의 가격이나 수량에 대한 모든 업데이트를 실시간으로 푸시합니다.
PartialBookDepth	Symbol, Depth	상위 <levels>개의 bid와 ask로, 매초 푸시됩니다. 유효한 <levels>는 5, 10 또는 20입니다.
DiffDepth	Symbol	주문장을 로컬에서 관리하는 데 사용되는 주문장 가격 및 수량 깊이 업데이트입니다.

성공적인 구독 / 구독 취소 후 클라이언트는 이에 대한 메시지를 수신하며, 여기서 id는 Subscribed / Unsubscribed 메서드의 결과입니다.

```
{
  "result": null,
  "id": 1
}
```

User Data Stream API

binance 계정에서 얻은 유효한 ApiKey가 필요하며, ApiKey는 구성 요소의 Binance.ApiKey 속성에 설정해야 합니다.

다음 데이터는 변경이 있을 때마다 클라이언트에 푸시됩니다. 어떤 메서드도 구독할 필요가 없으며, 유효한 ApiKey를 설정하면 자동으로 수행됩니다.

Method	설명
Account Update	계정 상태는 outboundAccountInfo 이벤트로 업데이트됩니다.
Balance Update	Balance Update는 다음 중에 발생합니다: - 계정에서의 입금 또는 출금 - 계정 간 자금 이체(예: 현물에서 마진으로)
Order Update	주문은 executionReport 이벤트로 업데이트됩니다.

REST API

기본 엔드포인트는 <https://api.binance.com> 입니다. 모든 엔드포인트는 JSON 객체 또는 배열을 반환합니다. 데이터는 오름차순으로 반환됩니다. 가장 오래된 것이 먼저, 가장 최신 것이 마지막입니다.

REST_API.BinanceOptions 속성을 사용하여 REST API Options에 접근합니다.

Public API EndPoints

이러한 엔드포인트는 권한 부여 없이 접근할 수 있습니다.

General EndPoints

Method	매개변수	설명
Ping		Rest API에 대한 연결을 테스트합니다.
GetServerTime		Rest API에 대한 연결성을 테스트하고 현재 서버 시간을 가져옵니다.
GetExchangeInformation		현재 거래소 거래 규칙 및 심볼 정보

Market Data EndPoints

Method	매개변수	설명
GetOrderBook	Symbol	호가창을 가져옵니다.
GetTrades	Symbol	최근 거래를 가져옵니다
GetHistoricalTrades	Symbol	이전 거래를 가져옵니다.
GetAggregateTrades	Symbol	압축된 집계 거래를 가져옵니다. 동일한 주문에서 동일한 가격으로 동시에 체결되는 거래는 수량이 집계됩니다.
GetKLines	Symbol, Interval	심볼에 대한 Kline/캔들스틱 바입니다. Klines는 open time으로 고유하게 식별됩니다.
GetAveragePrice	Symbol	심볼의 현재 평균 가격입니다.
Get24hrTicker	Symbol	24시간 롤링 창 가격 변동 통계. 심볼 없이 이에 접근할 때는 주의하십시오.
GetPriceTicker	Symbol	symbol의 최신 가격.
GetPriceTickers	Symbols	심볼 배열의 최신 가격입니다. 예: ["BTCUSDT", "BNBUSDT"]
GetBookTicker	Symbol	심볼 또는 심볼들에 대한 오더북의 최적 가격/수량.

GetUIKLines	Symbol, Interval	심볼에 대한 캔들스틱 봉입니다. 응답은 GetKLines와 유사하며, 캔들스틱 차트 표시에 최적화되어 있습니다.
GetRollingWindowTicker	Symbol, Symbols, WindowSize	롤링 윈도우 가격 변동 통계입니다. 참고: 지정하지 않으면 WindowSize 기본값은 1d입니다.
GetTradingDayTicker	Symbol, Symbols, Type	거래일의 가격 변동 통계.

Private API EndPoints

서버의 권한을 받으려면 APIKey와 APISecret이 필요합니다.

Account Data EndPoints

Method	매개변수	설명
NewOrder	Symbol, Side, Type	새 주문을 보냅니다.
PlaceMarketOrder	Side, Symbol, Quantity	새 Market Order를 배치합니다
PlaceMarketQuoteOrder	Side, Symbol, QuoteOrderQty	새 마켓 견적 주문을 등록합니다
PlaceLimitOrder	Side, Symbol, Quantity, LimitPrice	새 지정가 주문을 실행합니다
PlaceStopOrder	Side, Symbol, Quantity, StopPrice, LimitPrice	새 Stop Order를 넣습니다
PlaceStopTrailingOrder	Side, Symbol, Quantity, TrailingDelta, LimitPrice	새 Stop Trailing 주문을 배치합니다
PlaceTakeProfitOrder	Side, Symbol, Quantity, StopPrice, LimitPrice	새 Take Profit Order를 배치합니다
PlaceTakeProfitTrailingOrder	Side, Symbol, Quantity, TrailingDelta, LimitPrice	새 Take ProfitTrailing Order를 합니다
PlaceLimitMakerOrder	Side, Symbol, Quantity	새 Limit Market Order를 넣습니다
TestNewOrder	Symbol, Side, Type	새 주문 생성 및 signature/recvWindow long을 테스트합니다. 새 주문을 생성하고 검증하지만 매칭 엔진으로 보내지는 않습니다.
QueryOrder	Symbol	주문의 상태를 확인합니다.
CancelOrder	Symbol	활성 주문을 취소합니다. 활성 주문을 취소합니다. OrderId 또는 OrigClientOrderId 중 하나를 전송해야 합니다.
CancelAllOpenOrders	Symbol (선택적)	
GetOpenOrders		심볼의 모든 미결제 주문을 가져옵니다. 심볼 없이 이에 접근할 때 주의하십시오.
GetAllOrders	Symbol	모든 계정 주문을 가져옵니다. 활성, 취소됨 또는 체결됨.
NewOCO	Symbol, Side, Quantity, Price, StopPrice	새 OCO를 전송합니다.
CancelOCO	Symbol	전체 Order List를 취소합니다
QueryOCO	Symbol	제공된 선택적 매개변수를 기반으로 특정 OCO를 검색합니다
GetAllOCO		제공된 선택적 매개변수를 기반으로 모든 OCO를 검색합니다
GetOpenOCO		모든 미결제 OCO를 가져옵니다.
GetAccountInformation		현재 계정 정보를 가져옵니다.

GetAccountTradeList	Symbol	특정 계정과 심볼에 대한 거래를 가져옵니다.
CancelReplaceOrder	Symbol, Side, Type, CancelReplaceMode	기존 주문을 취소하고 동일한 심볼에 새 주문을 실행합니다.
NewOrderListOCO	Symbol, Side, Quantity, AboveType, BelowType	새 OCO order list를 넣습니다.
NewOrderListOTO	Symbol, WorkingType, WorkingSide, WorkingQuantity, WorkingPrice, PendingType, PendingSide, PendingQuantity	새 OTO(One-Triggers-the-Other) 주문 목록을 배치합니다.
NewOrderListOTOCO	Symbol, WorkingType, WorkingSide, WorkingQuantity, WorkingPrice, PendingSide, PendingAboveType, PendingBelowType, PendingQuantity	새 OTOCO(One-Triggers-a-One-Cancels-the-Other) 주문 목록을 발주합니다.
NewSOROrder	Symbol, Side, Type, Quantity	Smart Order Routing(SOR)을 사용하여 주문을 배치합니다.
TestSOROrder	Symbol, Side, Type, Quantity	Smart Order Routing(SOR)을 사용하여 새 주문을 테스트합니다. 새 주문을 생성하고 검증하지만 매칭 엔진으로 보내지는 않습니다.
GetOrderRateLimitUsage		모든 간격에 대한 사용자의 현재 주문 수 사용량을 표시합니다.
GetPreventedMatches	Symbol	STP(Self Trade Prevention)로 인해 만료된 주문 목록을 표시합니다.
GetAllocations	Symbol	SOR 주문 배치로 인한 allocation을 검색합니다.
GetAccountCommission	Symbol	현재 계정 수수료를 가져옵니다.

Convert EndPoints

Method	매개변수	설명
GetAllConvertPairs	FromAsset, ToAsset	변환 가능한 모든 token 쌍과 token의 각 상한/하한을 쿼리합니다
GetConvertAssetInfo		지원되는 자산의 정밀도 정보를 쿼리합니다
SendConvertQuoteRequest	FromAsset, ToAsset	요청한 토큰 쌍에 대한 견적을 요청합니다
AcceptConvertQuote	QuoteId	견적 ID로 제공된 견적을 수락합니다.
GetConvertOrderStatus	OrderId 또는 QuoteId	주문 ID로 주문 상태를 쿼리합니다.
PlaceConvertLimitOrder	BaseAsset, QuoteAsset, Side, LimitPrice	사용자가 지정가 주문을 발주할 수 있도록 합니다. baseAsset 또는 quoteAsset은 exchangeInfo 엔드 포인트를 통해 결정할 수 있습니다. Limit price는 baseAsset에서 quoteAsset으로 정의됩니다. baseAmount 또는 quoteAmount 중 하나가 사용됩니다.
CancelConvertLimitOrder	OrderId	사용자가 limit 주문을 취소할 수 있도록 합니다
GetConvertLimitOpenOrders		사용자가 기존의 모든 지정가 주문을 쿼리할 수 있도록 합니다.

GetConvertTradeHistory	StartTime, EndTime	startTime과 endTime 사이의 최대 간격은 30일입니다.

Wallet EndPoints

(*wallet 엔드포인트는 데모가 아닌 프로덕션 서버에서만 작동합니다)

Method	설명
GetWalletSystemStatus	시스템 상태를 가져옵니다.
GetWalletAllCoinsInformation	사용자가 입금 및 출금할 수 있는 코인 정보를 가져옵니다.
GetWalletDailyAccountSnapshot	Type: "SPOT", "MARGIN", "FUTURES" - 쿼리 기간은 30일 미만이어야 합니다 - 최근 1개월 이내의 쿼리만 지원합니다 - startTime과 endTime이 전송되지 않으면 기본적으로 최근 7일의 레코드를 반환합니다
SetWalletDisableFastWithdrawSwitch	이 요청은 계정의 fastwithdraw 스위치를 비활성화합니다. 이 엔드포인트를 요청하는 api 키에 대해 "trade" 옵션을 활성화해야 합니다.
SetWalletEnableFastWithdrawSwitch	이 요청은 계정에서 fastwithdraw 스위치를 활성화합니다. 이 엔드포인트를 요청하는 api 키에 대해 "trade" 옵션을 활성화해야 합니다. Fast Withdraw Switch가 켜져 있으면 Binance 계정으로의 자금 이체가 즉시 완료됩니다. 온체인 거래, 거래 ID, 출금 수수료가 없습니다.
WalletWithdraw	출금 요청을 제출합니다.
GetWalletDepositHistory	입금 기록을 가져옵니다.
GetWalletWithdrawHistory	출금 내역을 가져옵니다.
GetWalletDepositAddress	네트워크와 함께 입금 주소를 가져옵니다.
GetWalletAccountStatus	계정 상태 세부 정보를 가져옵니다.
GetWalletAccountAPITradingStatus	계정 api 거래 상태 세부 정보를 가져옵니다.
GetWalletDustLog	마지막 100개 레코드만 반환 2020/12/01 이후 레코드만 반환
GetWalletAssetsConvertedBNB	
WalletDustTransfer	dust 자산을 BNB로 변환합니다. 이 엔드포인트를 요청하는 API Key에 대해 Spot & Margin Trading 권한을 활성화해야 합니다.
GetWalletAssetDividendRecord	자산 배당 기록을 조회합니다.
GetWalletAssetDetail	Binance에서 지원하는 자산의 세부 정보를 가져옵니다.
GetWalletTradeFee	거래 수수료 가져오기
WalletUserUniversalTransfer	이 엔드포인트를 요청하는 API Key에 대해 Permits Universal Transfer 옵션을 활성화해야 합니다.MAIN_UMFUTURE Spot 계정을 USDⓈ-M Futures 계정으로 이체 Type의 ENUM: - MAIN_CMFUTURE Spot 계정을 COIN-M Futures 계정으로 이체 - MAIN_MARGIN Spot 계정을 Margin (cross) 계정으로 이체 - UMFUTURE_MAIN USDⓈ-M Futures 계정을 Spot 계정으로 이체 - UMFUTURE_MARGIN USDⓈ-M Futures 계정을 Margin (cross) 계정으로 이체 - CMFUTURE_MAIN COIN-M Futures 계정을 Spot 계정으로 이체 - CMFUTURE_MARGIN COIN-M Futures 계정을 Margin(cross) 계정으로 이체

	• MARGIN_MAIN Margin (cross) 계정을 Spot 계정으로 이체 • MARGIN_UMFUTURE Margin (cross) 계정을 USDⓈ-M Futures로 이체 • MARGIN_CMFUTURE Margin (cross) 계정을 COIN-M Futures로 이체 • ISOLATEDMARGIN_MARGIN Isolated margin 계정을 Margin(cross) 계정으로 이체 • MARGIN_ISOLATEDMARGIN Margin(cross) 계정을 Isolated margin 계정으로 이체 • ISOLATEDMARGIN_ISOLATEDMARGIN Isolated margin 계정을 Isolated margin 계정으로 이체 • MAIN_FUNDING Spot 계정을 Funding 계정으로 이체 • FUNDING_MAIN Funding 계정을 Spot 계정으로 이체 • FUNDING_UMFUTURE Funding 계정을 UMFUTURE 계정으로 이체 • UMFUTURE_FUNDING UMFUTURE 계정을 Funding 계정으로 이체 • MARGIN_FUNDING MARGIN 계정을 Funding 계정으로 이체 • FUNDING_MARGIN Funding 계정을 Margin 계정으로 이체 • FUNDING_CMFUTURE Funding 계정을 CMFUTURE 계정으로 이체 • CMFUTURE_FUNDING CMFUTURE 계정을 Funding 계정으로 이체
GetWalletQueryUserUniversalTransferHistory	• type이 ISOLATEDMARGIN_MARGIN 및 ISOLATEDMARGIN_ISOLATEDMARGIN일 때 fromSymbol을 보내야 합니다 • type이 MARGIN_ISOLATEDMARGIN 및 ISOLATEDMARGIN_ISOLATEDMARGIN일 때 toSymbol을 보내야 합니다 • 최근 6개월 이내의 쿼리만 지원합니다 • startTime와 endTime이 전송되지 않으면, 기본적으로 최근 7일 간의 레코드를 반환합니다
GetWalletFundingWallet	현재 다음 비즈니스 자산 쿼리를 지원합니다 : Binance Pay, Binance Card, Binance Gift Card, Stock Token
GetWalletUserAsset	사용자 자산을 가져옵니다. 양수 데이터만 해당됩니다.
GetWalletApiKeyPermission	

Events

Binance 메시지는 TsgcWebSocketClient 구성 요소에서 수신되며, 다음 이벤트를 사용할 수 있습니다:

OnConnect

Binance 서버에 성공적으로 연결된 후.

OnDisconnect

Binance 서버와의 연결이 끊긴 후

OnMessage

서버가 클라이언트에 보낸 메시지는 이 이벤트에서 처리됩니다.

OnError

프로토콜에 오류가 있으면 이 이벤트가 호출됩니다.

OnException

처리되지 않은 예외가 있으면 이 이벤트가 호출됩니다.

또한 Binance API 구성 요소에는 **OnBinanceHTTPException**이라는 특정 이벤트가 있으며, HTTP 요청(REST API 또는 WebSocket User Stream)을 호출하는 중 오류가 있을 때마다 발생합니다.

(*) Binance 서버의 변경으로 인해 Rad Studio 10.1 이전의 Indy 버전은 Test Servers에 연결할 수 없습니다. 이 문제는 Enterprise Edition이나 Indy 버전이 최신으로 업그레이드된 경우에는 영향을 주지 않습니다.

Binance | Connect WebSocket API

Binance WebSocket API에 연결하려면 새 Binance API 클라이언트를 생성하고 TsgcWebSocketClient에 연결하기만 하면 됩니다.

아래 예제를 참조하십시오:

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient(NULL);
TsgcWSAPI_Binance *oBinance = new TsgcWSAPI_Binance(NULL);
oBinance->Client = oClient;
oClient->Active = true;
```

Binance | Subscribe WebSocket Channel

Binance는 시장 데이터, 주문 등의 실시간 업데이트를 받기 위해 구독할 수 있는 다양한 채널을 제공합니다... 아래에서 Ticker를 구독하는 방법의 샘플을 확인하십시오:

```
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 | Get Market Data

Binance는 REST 엔드포인트를 통해 public 시장 데이터를 제공하며, 이러한 엔드포인트 중 하나를 호출하면 요청된 시장 데이터의 스냅샷을 얻게 됩니다.

Market Data 엔드포인트는 인증이 필요하지 않으므로 모든 사용자가 자유롭게 사용할 수 있습니다.

Example: ticker BNBBTC의 스냅샷을 가져오려면 다음 호출을 수행하십시오:

```
TsgcWSAPI_Binance *oBinance = new TsgcWSAPI_Binance(this);  
ShowMessage(oBinance->REST_API->GetPriceTicker("BNBBTC"));
```

Binance | Private REST API

Binance REST API는 public 및 private 엔드포인트를 제공합니다. Private 엔드포인트는 거래의 보안을 높이기 위해 메시지에 서명할 것을 요구합니다.

먼저 Binance 계정에 로그인하여 새 API를 생성해야 합니다. 다음 값을 얻게 됩니다:

- ApiKey
- ApiSecret

이 필드들은 Binance API 클라이언트 구성 요소의 Binance 속성에서 구성해야 합니다.

구성이 완료되면 Binance Pro REST API에 비공개 요청을 시작할 수 있습니다

*Private Requests는 로컬 컴퓨터의 로컬 시간이 동기화되어 있어야 합니다. 그렇지 않으면 요청이 Binance 서버에 의해 거부됩니다. 이에 대한 다음 글을 확인하십시오, [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

Binance를 사용하면 REST API를 사용하여 spot 거래를 할 수 있습니다.

구성

먼저 binance 계정에서 **API Key**를 생성하고 Spot 거래 권한을 추가해야 합니다.

이 작업이 완료되면 현물 거래를 시작할 수 있습니다.

먼저, **Binance Client** 구성 요소에서 **ApiKey**와 **ApiSecret**을 설정하십시오. 이는 Binance 서버로 전송되는 요청에 서명하는 데 사용됩니다.

Place an Order

새 주문을 발주하려면 Binance Client 구성 요소의 **REST_API.NewOrder** 메서드를 호출하기만 하면 됩니다.

주문 유형(시장가, 지정가...)에 따라 API는 더 많거나 적은 필드를 요구합니다.

Mandatory Fields

- **Symbol:** 제품 id 심볼입니다, 예: BNBBTC
- **Side:** BUY 또는 SELL
- **type:** 주문 유형
 - LIMIT
 - MARKET
 - STOP_LOSS
 - STOP_LOSS_LIMIT
 - TAKE_PROFIT
 - TAKE_PROFIT_LIMIT
 - LIMIT_MAKER

Type에 따른 추가 필수 필드

- **LIMIT:** timeInForce, quantity, price
- **MARKET:** quantity 또는 quoteOrderQty
- **STOP_LOSS / TAKE_PROFIT:** quantity, stopPrice
- **STOP_LOSS_LIMIT / TAKE_PROFIT_LIMIT:** timeInForce, quantity, price, stopPrice
- **LIMIT_MAKER:** quantity, price

주문을 보낼 때 두 가지 가능성이 있습니다:

1. **성공:** NewOrder 함수는 binance 서버가 보낸 메시지를 반환합니다.
2. **Error:** 예외가 OnBinanceHttpException 이벤트에서 반환됩니다.

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 | 비공개 요청 시간

Binance에 프라이빗 요청을 할 때, 요청의 보안을 높이기 위해 메시지에 서명합니다. 메시지는 로컬 시간을 가져와 서명된 메시지 안에 보냅니다. 로컬 시간이 Binance 서버와 5초보다 큰 차이가 나면 요청이 거부됩니다. 따라서 로컬 시간이 동기화되었는지 확인하는 것이 중요하며, OS의 시간 동기화 메서드를 사용하여 이를 수행할 수 있습니다.

로직은 다음과 같습니다:

```
if (timestamp < (serverTime + 1000) && (serverTime - timestamp) <= recvWindow) {
// process request
} else {
// reject request
}
```

5000 이하의 작은 recvWindow를 사용하는 것이 좋습니다! 최대값은 60000밀리초를 초과할 수 없습니다.

GetServerTime 메서드를 호출하여 Binance 서버 시간을 확인할 수 있으며, 이는 Binance 서버의 시간을 반환합니다

RecvWindow의 기본값은 **5000**이며, 이 값은 **REST_API.BinanceOptions.RecvWindow** 속성을 사용하여 늘릴 수 있습니다.

Binance | Withdraw

Binance를 사용하면 Wallet API를 사용하여 Withdraw 요청을 제출할 수 있으며, 다음 매개변수만 필수입니다.

- 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

Binance는 국제적인 다국어 cryptocurrency 거래소입니다. Binance 데이터에 액세스하기 위한 일부 API를 제공합니다. 이 구성 요소를 사용하면 Binance Futures WebSocket Market Streams를 가져올 수 있습니다.

<https://binance-docs.github.io/apidocs/futures/en>
<https://binance-docs.github.io/apidocs/delivery/en>

Futures Contracts

Binance API에는 public과 private 두 가지 유형의 메서드가 있습니다. Public 메서드는 인증 없이 액세스할 수 있습니다(예: ticker 가격 가져오기). 일부는 private이며 사용자 데이터와 관련이 있습니다. 이러한 메서드는 Binance API 키 사용을 요구합니다.

- **ApiKey:** binance 계정에서 새 api 키를 요청할 수 있습니다. 값을 이 속성에 복사하기만 하면 됩니다.
- **ApiSecret:** API secret은 REST_API에만 필요하며, websocket api는 일부 메서드에 대해 ApiKey만 필요합니다.
- **TestNet:** 활성화하면 Binance Demo Account에 연결합니다(기본값 false).
 - **HTTPLogOptions:** HTTP 요청 로그를 텍스트 파일에 저장합니다.
 - **Enabled:** 활성화하면 WebSocket API의 모든 HTTP 요청을 저장합니다.
 - **FileName:** 로그가 저장될 파일 이름의 전체 경로입니다.
 - **REST:** REST API 요청 로그를 텍스트 파일에 저장합니다.
 - **Enabled:** 활성화하면 REST API의 모든 HTTP 요청을 저장합니다.
 - **FileName:** 로그가 저장될 파일 이름의 전체 경로입니다.
- **UserStream:** 활성화되면 클라이언트는 Account, Orders 또는 Balance Update에 대한 알림을 받습니다(기본적으로 true).
- **ListenKeyOnDisconnect:** 이 속성은 활성 ListenKey가 있는 상태에서 클라이언트가 Binance 서버와의 연결을 끊을 때 수행할 작업을 지정합니다.
 - **blkodDeleteListenKey:** Binance 서버에 HTTP 요청을 수행하여 활성 ListenKey를 삭제합니다(이것이 기본값입니다).
 - **blkodClearListenKey:** Binance 서버에서 ListenKey를 삭제하지 않고 필드의 값만 지웁니다.
 - **blkodDoNothing:** 아무 작업도 하지 않으므로, 다음에 Binance에 연결할 때 동일한 ListenKey를 사용하려고 시도합니다.
- **UseCombinedStreams:** 활성화되면 스트림을 다음과 같이 결합합니다:
 {"stream": "<streamName>", "data": <rawPayload>} (기본적으로 비활성화됨)

클라이언트는 **USDT** 또는 **COIN** Binance Futures에 연결할 수 있습니다. **FuturesContracts** 속성을 사용하여 거래하려는 계약을 설정하십시오:

- **bfcUSDT:** USD-M Futures API에 연결합니다.
- **bfcCOIN:** COIN-M Futures API에 연결합니다.

클라이언트는 Production 또는 Demo Binance 계정에 연결할 수 있습니다. **TestNet** 속성이 활성화되면 Demo 계정에 연결되고, 그렇지 않으면 production Binance 서버에 연결됩니다.

WebSocket Stream API

클라이언트는 성공적인 연결 후 이벤트를 구독/구독 취소할 수 있습니다. 다음 Subscription / Unsubscription 메서드가 지원됩니다.

Method	매개변수	설명
AggregateTrades	Symbol	Aggregate Trade Stream은 100밀리초마다 단일 taker 주문에 대해 집계된 거래 정보를 푸시합니다.

MarkPrice	Symbol, UpdateSpeed	단일 심볼에 대한 마크 가격과 펀딩 비율이 3초마다 또는 1초마다 푸시됩니다.
AllMarkPrice	UpdateSpeed	모든 심볼에 대한 mark price와 펀딩 비율을 3초마다 또는 매초마다 푸시합니다.
KLine	Symbol, Interval	Kline/Candlestick Stream은 현재 kline/candlestick에 대한 업데이트를 250밀리초마다 푸시합니다(존재하는 경우).
MiniTicker	Symbol	단일 심볼에 대한 24시간 롤링 윈도우 미니 티커 통계입니다. 이는 UTC 일의 통계가 아니라 requestTime부터 24시간 전까지의 24시간 롤링 윈도우입니다.
AllMiniTicker		모든 심볼에 대한 24시간 롤링 윈도우 미니 ticker 통계입니다. 이는 UTC 날짜의 통계가 아니라 requestTime부터 24시간 전까지의 24시간 롤링 윈도우입니다. 변경된 ticker만 배열에 존재한다는 점에 유의하십시오.
Ticker	Symbol	단일 심볼에 대한 24시간 롤링 윈도우 티커 통계입니다. 이는 UTC 일자의 통계가 아니라 requestTime부터 그 24시간 전까지의 24시간 롤링 윈도우입니다.
AllMarketTickers		모든 심볼에 대한 24시간 롤링 윈도우 티커 통계입니다. 이것은 UTC 일자의 통계가 아니라, requestTime에서 24시간 전까지의 24시간 롤링 윈도우입니다. 변경된 티커만 배열에 표시됩니다.
BookTicker	Symbol	지정된 symbol에 대한 최우선 매수 또는 매도 호가의 가격 또는 수량에 대한 모든 업데이트를 실시간으로 푸시합니다.
AllBookTickers		모든 심볼에 대해 최고 매수 또는 매도의 가격이나 수량에 대한 모든 업데이트를 실시간으로 푸시합니다.
LiquidationOrders	Symbol	Liquidation Order Stream은 특정 심볼에 대한 강제 청산 주문 정보를 푸시합니다.
AllLiquidationOrders		All Liquidation Order Streams는 시장의 모든 심볼에 대한 강제 청산 주문 정보를 푸시합니다.
PartialBookDepth	Symbol, Depth	최상위 bids 및 asks, 유효 값은 5, 10, 20입니다.
DiffDepth	Symbol	매수 및 매도 호가로, 250밀리초, 500밀리초, 100밀리초마다 또는 실시간으로(존재하는 경우) 푸시됩니다.

구독 / 구독 해지가 성공한 후, 클라이언트는 이에 대한 메시지를 받으며, 여기서 id는 Subscribed / Unsubscribed 메서드의 결과입니다.

```
{
  "result": null,
  "id": 1
}
```

User Data Stream API

binance 계정에서 얻은 유효한 ApiKey가 필요하며, ApiKey는 구성 요소의 Binance.ApiKey 속성에 설정해야 합니다.

다음 데이터는 변경이 있을 때마다 클라이언트에 푸시됩니다. 어떤 메서드도 구독할 필요가 없으며, 유효한 ApiKey를 설정하면 자동으로 수행됩니다.

Method	설명
Margin Call	사용자의 포지션 위험 비율이 너무 높으면 이 스트림이 푸시됩니다. 이 메시지는 위험 안내 정보로만 사용되며 투자 전략에는 권장되지 않습니다. 변동성이 큰 시장의 경우, 이 스트림이 푸시될 때 사용자의 포지션이 동시에 청산될 가능성이 있을 수 있습니다.
Balance and Position Update	Balance Update는 다음 중에 발생합니다: - 잔액 또는 포지션이 업데이트되면 이 이벤트가 푸시됩니다. "FUNDING FEE"가 사용자의 잔액으로 변경될 때.
Order Update	새 주문이 생성되거나 주문 상태가 변경되면 이러한 이벤트가 푸시됩니다.

REST API

모든 엔드포인트는 JSON 객체 또는 배열을 반환합니다. 데이터는 오름차순으로 반환됩니다. 가장 오래된 것이 먼저, 가장 새로운 것이 나중입니다.

Public API EndPoints

이러한 엔드포인트는 권한 부여 없이 접근할 수 있습니다.

General EndPoints

Method	매개변수	설명
Ping		Rest API에 대한 연결을 테스트합니다.
GetServerTime		Rest API에 대한 연결성을 테스트하고 현재 서버 시간을 가져옵니다.
GetExchangeInformation		현재 거래소 거래 규칙 및 심볼 정보

Market Data EndPoints

Method	매개변수	설명
GetOrderBook	Symbol	호가창을 가져옵니다.
GetTrades	Symbol	최근 거래를 가져옵니다
GetHistoricalTrades	Symbol	이전 거래를 가져옵니다.
GetAggregateTrades	Symbol	압축된 집계 거래를 가져옵니다. 동일한 주문에서 동일한 가격으로 동시에 체결되는 거래는 수량이 집계됩니다.
GetKLines	Symbol, Interval	심볼에 대한 Kline/캔들스틱 바입니다. Klines는 open time으로 고유하게 식별됩니다.
Get24hrTicker	Symbol	24시간 롤링 창 가격 변동 통계. 심볼 없이 이에 접근할 때는 주의하십시오.
GetPriceTicker	Symbol	심볼 또는 심볼들의 최신 가격입니다.
GetBookTicker	Symbol	심볼 또는 심볼들에 대한 오더북의 최적 가격/수량.
GetMarkPrice	Symbol	Mark Price 및 Funding Rate
GetFundingRateHistory	Symbol	
GetOpenInterest	Symbol	특정 심볼의 현재 미결제약정을 가져옵니다.
GetOpenInterestStatistics	Symbol, Period	
GetTopTraderAccountRatio	Symbol, Period	
GetTopTraderPositionRatio	Symbol, Period	
GetGlobalAccountRatio	Symbol, Period	
GetTakerVolume	Symbol, Period	
GetContinuousKLines	Pair, ContractType, Interval	특정 계약 유형에 대한 Kline/캔들스틱 바입니다.
GetIndexPriceKLines	Pair, Interval	페어의 인덱스 가격에 대한 Kline/캔들스틱 바입니다.
GetMarkPriceKLines	Symbol, Interval	심볼의 mark price에 대한 Kline/캔들스틱 바입니다.
GetPremiumIndexKLines	Symbol, Interval	심볼의 premium index kline 바입니다.
GetFundingInfo		모든 심볼에 대한 펀딩 비율 정보를 가져옵니다.
GetPriceTickerV2	Symbol	심볼 또는 심볼들의 최신 가격입니다(V2).
GetIndexInfo	Symbol	인덱스 정보를 가져옵니다.

GetAssetIndex	Symbol	멀티 에셋 모드의 자산 인덱스를 가져옵니다.
GetConstituents	Symbol	인덱스 구성 종목을 가져옵니다.
GetDeliveryPrice	Pair	delivery price를 가져옵니다.
GetBasis	Pair, ContractType, Period	기준 데이터를 가져옵니다.

Private API EndPoints

서버의 권한을 받으려면 APIKey와 APISecret이 필요합니다.

Account and Trades EndPoints

Method	매개변수	설명
ChangePositionMode	DualPosition	모든 심볼에서 사용자의 포지션 모드(Hedge Mode 또는 One-way Mode)를 변경합니다
GetCurrentPositionMode		모든 심볼에서 사용자의 포지션 모드(Hedge Mode 또는 One-way Mode)를 가져옵니다
NewOrder	Symbol, Side, PositionSide, Type	새 주문을 보냅니다.
PlaceMarketOrder	Side, Symbol, Quantity	
PlaceLimitOrder	Side, Symbol, Quantity, LimitPrice	
PlaceStopOrder	Side, Symbol, Quantity, StopPrice, LimitPrice	
PlaceTrailingStopOrder	Side, Symbol, Quantity, aActivationPrice, aCallbackRate	
QueryOrder	Symbol	주문의 상태를 확인합니다.
CancelOrder	Symbol	활성 주문을 취소합니다. OrderId 또는 OrigClientOrderId 중 하나를 보내야 합니다.
CancelAllOpenOrders	Symbol	
AutoCancelAllOpenOrders	Symbol, CountdownTimer	지정된 카운트다운이 끝날 때 지정된 심볼의 모든 미결제 주문을 취소합니다.
QueryCurrentOpenOrder	Symbol	
GetOpenOrders	Symbol	심볼의 모든 미결제 주문을 가져옵니다. 심볼 없이 이에 접근할 때 주의하십시오.
GetAllOrders	Symbol	모든 계정 주문을 가져옵니다. 활성, 취소됨 또는 체결됨.
GetAccountBalance		
GetAccountInformation		현재 계정 정보를 가져옵니다.
ChangeInitialLeverage	Symbol, Leverage	특정 심볼 시장에 대한 사용자의 초기 레버리지를 변경합니다.
ChangeMarginType	Symbol, MarginType	
ModifyIsolatedPositionMargin	Symbol, Amount, Type	
GetPositionMarginChangeHistory	Symbol	
GetPositionInformation	Symbol	
GetAccountTradeList	Symbol	
GetIncomeHistory	Symbol	
GetNotionalLeverageBracket	Symbol	

TestNewOrder	Symbol, Side, PositionSide, Type	새 주문 생성 및 signature/recvWindow long 을 테스트합니다. 새 주문을 생성하고 검증하지만 매칭 엔진으로 보내지는 않습니다.
ModifyOrder	Symbol	기존 주문 수정.
NewBatchOrders	BatchOrders	여러 주문을 배치합니다.
ModifyBatchOrders	BatchOrders	여러 주문을 수정합니다.
CancelBatchOrders	Symbol	여러 주문을 취소합니다.
GetOrderAmendment	Symbol	주문 수정 기록을 가져옵니다.
CountdownCancelAll	Symbol, CountdownTime	지정된 카운트다운이 끝날 때 지정된 심볼의 모든 미체결 주문을 취소합니다.
GetForceOrders	Symbol	사용자의 강제 청산 주문을 가져옵니다.
GetADLQuantile	Symbol	포지션에 대한 ADL 분위수 추정값을 가져옵니다.
GetAccountBalanceV3		선물 계정 잔고를 가져옵니다(V3).
GetAccountInformationV3		현재 계정 정보를 가져옵니다(V3).
GetPositionInformationV3	Symbol	현재 포지션 정보를 가져옵니다(V3).
GetCommissionRate	Symbol	사용자 수수료를 가져옵니다.
GetAccountConfig		현재 계정 구성을 가져옵니다.
GetSymbolConfig	Symbol	심볼 구성을 가져옵니다.
GetOrderRateLimit		사용자의 주문 rate limit을 가져옵니다.
GetApiTradingStatus	Symbol	API 거래 정량 규칙 지표를 가져옵니다.
ChangeMultiAssetsMode	MultiAssetsMargin	사용자의 multi-assets 모드를 변경합니다. Multi-Assets Mode: true; Single-Asset Mode: false.
GetMultiAssetsMode		사용자의 현재 다중 자산 모드를 가져옵니다.
SetFeeBurn	FeeBurn	사용자의 BNB fee burn 상태를 변경합니다.
GetFeeBurn		사용자의 BNB 수수료 소각 상태를 가져옵니다.
CreateListenKey		새 user data stream을 시작합니다. keepalive 가 전송되지 않으면 60분 후에 스트림이 닫힙니다.
KeepAliveListenKey		타임아웃을 방지하기 위해 user data stream을 keepalive합니다.
CloseListenKey		사용자 데이터 스트림을 닫습니다.

Events

Binance Futures 메시지는 TsgcWebSocketClient 구성 요소에서 수신되며, 다음 이벤트를 사용할 수 있습니다.

OnConnect

Binance 서버에 성공적으로 연결된 후.

OnDisconnect

Binance 서버와의 연결이 끊긴 후

OnMessage

서버가 클라이언트에 보낸 메시지는 이 이벤트에서 처리됩니다.

OnError

프로토콜에 오류가 있으면 이 이벤트가 호출됩니다.

OnException

처리되지 않은 예외가 있으면 이 이벤트가 호출됩니다.

또한 Binance API 구성 요소에는 **OnBinanceHTTPException**이라는 특정 이벤트가 있으며, HTTP 요청(REST API 또는 WebSocket User Stream)을 호출하는 중 오류가 있을 때마다 발생합니다.

(*) Binance 서버의 변경으로 인해 Rad Studio 10.1 이전의 Indy 버전은 Test Server에 연결할 수 없습니다. 이 문제는 Enterprise Edition이나 Indy 버전이 최신으로 업그레이드된 경우에는 영향을 미치지 않습니다.

API Binance Futures | Trade

Binance를 사용하면 REST API를 사용하여 선물을 거래할 수 있습니다.

구성

먼저 binance 계정에서 **API Key**를 생성하고 Futures 거래 권한을 추가해야 합니다.

이 작업이 완료되면 선물 거래를 시작할 수 있습니다.

먼저 **USDT** 또는 **COIN** 선물 중 어느 것으로 거래할지 선택해야 합니다. FuturesContracts라는 속성이 있으며, 여기에서 거래하려는 선물 계약을 설정할 수 있습니다.

그런 다음, Binance Futures Client 구성 요소에서 **ApiKey**와 **ApiSecret**을 설정하십시오. 이것은 Binance 서버로 전송되는 요청에 서명하는 데 사용됩니다.

Place an Order

새 주문을 하려면, Binance Futures Client 구성 요소의 **REST_API.NewOrder** 메서드를 호출하기만 하면 됩니다.

주문 유형(market, limit...)에 따라 API는 더 많거나 적은 필드를 요구합니다.

Mandatory Fields

- **Symbol:** 제품 id 심볼, 예: BTCUSD_210326
- **Side:** BUY 또는 SELL
- **type:** 주문 유형
 - LIMIT
 - MARKET
 - STOP
 - TAKE_PROFIT
 - STOP_MARKET
 - TAKE_PROFIT_MARKET
 - TRAILING_STOP_MARKET

Type에 따른 추가 필수 필드

- **LIMIT:** timeInForce, quantity, price
- **MARKET:** quantity
- **STOP/TAKE_PROFIT:** quantity, price, stopPrice
- **STOP_MARKET/TAKE_PROFIT_MARKET:** stopPrice
- **TRAILING_STOP_MARKET:** callbackRate

주문을 보낼 때 두 가지 가능성이 있습니다:

1. **성공:** NewOrder 함수는 binance 서버가 보낸 메시지를 반환합니다.
2. **Error:** 예외가 OnBinanceHttpException 이벤트에서 반환됩니다.

API SocketIO

SocketIO

Socket.IO는 실시간 웹 애플리케이션을 위한 JavaScript 라이브러리입니다. 웹 클라이언트와 서버 간의 실시간 양방향 통신을 가능하게 합니다. 두 부분으로 구성됩니다: 브라우저에서 실행되는 클라이언트 측 라이브러리와 Node.js용 서버 측 라이브러리. 두 구성 요소는 거의 동일한 API를 가지고 있습니다. Node.js와 마찬가지로 이벤트 기반입니다.

Messages Types

0: open (새 전송이 열릴 때 서버에서 전송됨(재확인))

1: close (이 전송을 닫도록 요청하지만 연결 자체를 종료하지는 않습니다.)

2: ping(클라이언트가 전송. 서버는 동일한 데이터를 포함한 pong 패킷으로 응답해야 합니다)

example

client sends: 2probe

server sends: 3probe

3: pong(ping 패킷에 응답하기 위해 서버가 전송.)

4: string message(실제 메시지로, 클라이언트와 서버는 데이터와 함께 콜백을 호출해야 합니다.)

example:

42/chat,[{"join"}, {"room:1}"]

4는 engine.io 프로토콜의 message packet 유형입니다

2는 socket.io 프로토콜의 EVENT 유형입니다

/chat은 socket.io가 처리하는 데이터입니다

socket.io는 "join" 이벤트를 발생시킵니다

"room: 1" 데이터를 전달합니다. namespace가 /일 때만 namespace를 생략할 수 있습니다.

5: upgrade (engine.io가 전송을 전환하기 전에 서버와 클라이언트가 이 전송을 통해 통신할 수 있는지 테스트합니다. 이 테스트가 성공하면 클라이언트는 서버가 이전 전송의 캐시를 플러시하고 새 전송으로 전환하도록 요청하는 upgrade 패킷을 보냅니다.)

6: noop (noop 패킷입니다. 주로 들어오는 WebSocket 연결이 수신될 때 폴 사이클을 강제하는 데 사용됩니다.)

Properties

API: SocketIO 버전을 지정합니다.

ioAPI0: socket.io 0.* 서버를 지원합니다(기본적으로 선택됨)

ioAPI1: socket.io 1.* 서버를 지원합니다

ioAPI2: socket.io 2.* 서버를 지원합니다.

ioAPI3: socket.io 3.* 서버를 지원합니다

ioAPI4: socket.io 4.* 서버를 지원합니다

Base64: 활성화되면 바이너리 메시지가 base64로 수신됩니다.

HandShakeCustomURL: socket.io 세션을 가져오는 URL을 사용자 정의할 수 있도록 합니다.

HandShakeTimestamp: 새 세션을 요청할 때 timestamp를 매개변수로 보내려는 경우에만 활성화하십시오(gevent-socketio python 서버에 액세스하려는 경우 이 속성을 활성화하십시오).

HandShakeAuthToken: 서버가 인증을 위한 토큰을 요구하면 여기에 인증 토큰을 설정하십시오.

Namespace: 서버에 연결할 때 namespace를 설정할 수 있습니다.

Polling: 이 속성을 비활성화하면 클라이언트는 websocket을 전송으로 사용하여 서버에 직접 연결합니다.

Parameters: 연결 매개변수를 설정할 수 있습니다.

EncodeParameters: 활성화되면 매개변수가 인코딩됩니다.

Methods

WriteData 메서드를 사용하여 socket.io 서버로 메시지를 보냅니다(Message Types 섹션을 따라).

1. 사용자 이름으로 John을 사용하여 매개변수 하나와 함께 "add user" 메서드를 호출하십시오.

```
WriteData("42[\"add user\", \"John\"]");
```

Events

OnHTTPRequest

새 websocket 연결이 설정되기 전에 socket.io 서버는 클라이언트가 새 세션 id를 가져오기 위해 새 HTTP 연결을 열도록 요구합니다. 경우에 따라 socket.io 서버는 HTTP 헤더를 사용한 인증을 요구합니다. 이 이벤트를 사용하여 Basic 권한 부여 또는 Bearer 토큰 인증과 같은 사용자 지정 HTTP 헤더를 추가할 수 있습니다.

OnAfterConnect

이 이벤트는 socket.io 연결이 성공하고 클라이언트가 서버에 메시지를 보낼 수 있을 때 호출됩니다. 여기서 예를 들어 네임스페이스를 구독할 수 있습니다.

OnHTTPConnectionSSL

WebSocket 서버가 보안 연결을 요구할 때, 클라이언트가 서버에 연결하려고 시도하면 다음과 같은 오류 메시지가 표시될 수 있습니다:

Error connecting with SSL. error:XXXXXXXX:SSL routines:ssl3_read_bytes:tlsv1 alert protocol version

이 오류는 클라이언트가 서버에서 지원하지 않는 TLS 버전을 사용하여 연결하려고 함을 의미합니다.

이 오류를 해결하려면 WebSocket 클라이언트 구성 요소의 OnSSLAfterCreateHandler를 처리하고 최신 TLS 버전을 설정해야 합니다.

예: 여기에서 프로토콜 버전으로 TLS 1.2를 설정합니다.

```
void OnHTTPConnectionSSL(TObject *Sender; TIdSSLIOHandlerSocketBase *aSSLHandler)
{
    static_cast<tidssliohandlersocketopenssl*>(aSSLHandler)->SSLOptions->Method = sslvTLSv1_2;
}
```

API Coinbase

Coinbase

지원되는 API

- **WebSockets API**: 공개 websocket 서버에 연결하고 실시간 시장 데이터 업데이트를 제공합니다.
- **REST API**: REST API에는 계정 및 주문 관리뿐만 아니라 공개 시장 데이터를 위한 엔드포인트가 있습니다.

Most common uses

- **WebSockets API**
 - **WebSocket API에 연결하는 방법**
 - **WebSocket 채널을 구독하는 방법**
- **REST API**
 - **Market Data를 가져오는 방법**
 - **Private REST API 사용 방법**
 - **주문하는 방법**
 - **SandBox 계정 사용 방법**
 - **Private Requests Time**

WebSockets API

WebSocket 피드는 공개적으로 사용 가능하며 주문 및 거래에 대한 실시간 시장 데이터 업데이트를 제공합니다. 프로덕션에서 두 개의 엔드포인트가 지원됩니다:

- **Market Data**는 주문과 거래 모두에 대한 업데이트를 제공하는 피드입니다. 이제 대부분의 채널은 인증 없이 사용할 수 있습니다.
- **User Order Data**는 사용자의 주문에 대한 업데이트를 제공합니다.

다음 채널을 구독할 수 있습니다:

메서드	Arguments	설명
SubscribeHeartBeat		모든 연결을 열린 상태로 유지하기 위한 실시간 서버 ping
SubscribeStatus	aProductId : 제품의 id 입니다	미리 설정된 간격으로 모든 상품과 통화를 전송합니다
SubscribeCandles	aProductId : 제품의 id 입니다	product candle에 대한 실시간 업데이트
SubscribeTicker	aProductId : 제품의 id 입니다	매칭이 발생할 때마다 실시간 가격 업데이트
SubscribeTickerBatch	aProductId : 제품의 id 입니다	5000밀리초마다 실시간 가격 업데이트
SubscribeLevel2	aProductId : 제품의 id 입니다	모든 업데이트 및 주문 장부 스냅샷을 유지하는 가장 쉬운 방법
SubscribeMarketTrades	aProductId : 제품의 id 입니다	시장 거래가 발생할 때마다 실시간 업데이트
SubscribeUser	aProductId : 제품의 id 입니다	인증된 사용자를 포함하는 메시지만 전송합니다

User 및 FuturesBalanceSummary 채널에는 인증이 필요하므로, 먼저 Coinbase 계정에서 API 키를 요청한 다음 구성 요소의 Coinbase 속성에 값을 설정하십시오:

- ApiKey
- ApiSecret

인증은 다음과 같은 몇 가지 이점을 제공합니다.

1. 당사자 중 한 명인 메시지는 확장되며 더 유용한 필드를 가집니다
2. 귀하가 넣은 stop order에 대한 lifecycle 정보와 같은 비공개 메시지를 받게 됩니다

REST API

Private Endpoints

프라이빗 엔드포인트는 주문 관리 및 계정 관리에 사용할 수 있습니다.

요청에 서명할 수 있으려면 먼저 Coinbase Pro 웹사이트를 통해 API 키를 생성해야 합니다. API 키는 특정 프로필로 범위가 지정됩니다. 키를 생성하면 반드시 기억해야 하는 3개의 정보를 얻게 됩니다:

- Key
- Secret
- Passphrase

Key와 Secret은 무작위로 생성되어 Coinbase Pro에서 제공됩니다. Passphrase는 API 액세스를 더욱 안전하게 하기 위해 사용자가 제공합니다. Coinbase Pro는 검증을 위해 passphrase의 솔트된 해시를 저장하지만, 잊어버린 경우 passphrase를 복구할 수 없습니다.

API 키의 기능을 제한할 수 있습니다. 키를 생성하기 전에 키가 가질 권한을 선택해야 합니다. 권한은 다음과 같습니다:

- View - 키에 읽기 권한을 허용합니다. 여기에는 모든 GET 엔드포인트가 포함됩니다.
- Transfer - 키가 계정을 대신하여 통화를 이체할 수 있도록 허용합니다(입금 및 출금 포함). 주의하여 활성화하십시오 — API 키 이체는 2단계 인증을 우회합니다.
- Trade - 키가 주문을 입력하고 거래 데이터를 검색할 수 있도록 합니다. 여기에는 POST /orders 및 여러 GET 엔드포인트가 포함됩니다.

Accounts

메서드	Arguments	Description
ListAccounts		API 키의 프로필에서 거래 계정 목록을 가져옵니다.
GetAccount	aAccountId: 계정의 id	단일 계정에 대한 정보. account_id를 알고 있을 때 이 엔드포인트를 사용하십시오. API 키는 계정과 동일한 프로필에 속해야 합니다.

Orders

메서드	Arguments	Description
PlaceNewOrder	aOrder: 주문의 가능한 모든 필드를 포함하는 클래스	새 주문을 배치합니다. 고급 주문 옵션에 액세스해야 하는 경우에만 사용하십시오.
PlaceMarketOrder	aSide: buy 또는 sell aProductId: 제품의 id aQuoteSize: Trading Pair의 두 번째 Asset의 양입니다.	새 Market 주문을 넣습니다.

	aBaseSize: 거래 페어에서 첫 번째 Asset의 수량 aClient_oid: 주문을 식별하기 위해 사용자가 선택한 Order ID	
PlaceLimitOrder	aSide: buy 또는 sell aProductId: 제품의 id aQuoteSize: Trading Pair에서 두 번째 Asset의 수량입니다. aBaseSize: Trading Pair에서 첫 번째 Asset의 양 aLimitPrice: 가격 한도 Client_oid: 주문을 식별하기 위해 직접 선택한 주문 ID	새 Limit 주문을 배치합니다.
PlaceStopOrder	aSide: buy 또는 sell ProductId: 제품의 id aBaseSize: Trading Pair에서 첫 번째 Asset의 양 StopPrice: stop의 가격 aLimitPrice: 가격 한도 aStopDirection: loss 또는 entry Client_oid: 주문을 식별하기 위해 선택한 Order ID	새 Stop Order를 합니다
CancelOrder	aOrderId: 주문의 id	이전에 넣은 주문을 취소합니다. 주문은 API 키가 속한 프로필에 속해야 합니다.
EditOrder	aOrderId: 주문의 id aPrice: 가격 aSize: 수량	지정된 새 크기 또는 새 가격으로 주문을 편집합니다
EditOrderPreview	aOrderId: 주문의 id aPrice: 가격 aSize: 수량	지정된 새 크기 또는 새 가격으로 주문 편집 요청을 미리 봅니다.
ListOrders		선택적 쿼리 매개변수(product_id, order_status 등)로 필터링된 주문 목록을 가져옵니다.
GetOrder	aOrderId: 주문의 id	주문 ID로 단일 주문을 가져옵니다.
PreviewOrder		주문을 미리 봅니다.
ClosePosition	aOrderId: 주문의 id aProductId: 제품의 id aSize: 수량	지정된 product_id에 대한 모든 미체결 포지션을 청산하는 주문을 실행합니다.

Market Data

메서드	Arguments	Description
GetPublicProducts		거래에 사용 가능한 통화 쌍 목록을 가져옵니다.
GetPublicProduct	aProductId: 제품의 id	product ID로 단일 상품에 대한 정보를 가져옵니다.
GetPublicProductBook	aProductId: 상품의 id입니다.	단일 product에 대한 매수/매도 목록을 가져옵니다. 표시되는 세부 정보의 양은 limit 매개변수로 사용자 지정할 수 있습니다.
GetPublicProductCandles	aProductId: 상품의 id입니다. aStart: 시간 간격의 시작입니다. aEnd: 시간 간격의 끝 aGranularity: 각 candle이 나타내는 시간 프레임입니다.	제품 ID로 단일 제품의 요율을 버킷으로 그룹화하여 가져옵니다.
GetTrades	aProductId: 상품의 id입니다.	제품 ID로 최근 거래(틱)와 최우선 매수/매도 호가에 대한 스냅샷 정보를 가져옵니다.
GetTime		Coinbase Advanced API에서 현재 시간을 가져옵니다.

Fills

메서드	Arguments	Description
GetFillsByOrderId		주문 id로 필터링된 체결 목록을 가져옵니다
GetFillsByProductId		product id로 필터링된 체결 목록을 가져옵니다
GetFillsByTradeId		trade id로 필터링된 fills 목록을 가져옵니다

Convert

메서드	Arguments	Description
CreateConvertQuote		통화 간 환전 견적을 생성합니다.
CommitConvertTrade		convert 거래를 commit합니다.
GetConvertTrade		convert 거래 세부 정보를 가져옵니다.

Fees

메서드	Arguments	Description
GetTransactionSummary		거래 수수료 요약을 가져옵니다.

Products (인증됨)

메서드	Arguments	Description
ListProducts		사용 가능한 product를 나열합니다.
GetProduct	aProductId: 제품의 id입니다	특정 제품을 가져옵니다.
GetProductBook	aProductId: 제품의 id입니다	상품 주문서를 가져옵니다.
GetProductCandles	aProductId: 제품의 id입니다	상품 OHLCV 캔들을 가져옵니다.
GetMarketTrades	aProductId: 제품의 id입니다	최근 시장 거래를 가져옵니다.
GetBestBidAsk		최우선 매수/매도 가격을 가져옵니다.

Portfolios

메서드	Arguments	Description
ListPortfolios		모든 포트폴리오를 나열합니다.
CreatePortfolio		새 포트폴리오를 생성합니다.
DeletePortfolio		포트폴리오를 삭제합니다.
GetPortfolioBreakdown		포트폴리오 분석 세부 정보를 가져옵니다.
MovePortfolioFunds		포트폴리오 간에 자금을 이동합니다.

Perpetuals

메서드	Arguments	Description
GetPerpetualsPortfolioSummary		perpetuals 포트폴리오 요약을 가져옵니다.
ListPerpetualsPositions		무기한 포지션을 나열합니다.
GetPerpetualsPosition		특정 perpetuals 포지션을 가져옵니다.

Coinbase | Connect WebSocket API

Coinbase WebSocket API에 연결하려면 새 Coinbase API 클라이언트를 만들고 TsgcWebSocketClient에 연결하기만 하면 됩니다. 아래 예를 참조하십시오:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase();
oCoinbase->Client = oClient;
oClient->Active = true;
```

Coinbase | Subscribe WebSocket Channel

Coinbase는 시장 데이터, 주문 등의 실시간 업데이트를 받기 위해 구독할 수 있는 다양한 채널을 제공합니다... 아래에서 Ticker를 구독하는 방법의 샘플을 확인하십시오:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase();
oCoinbase->Client = oClient;
oCoinbase->SubscribeTicker("ETH-USD");

void OnCoinbaseMessage(TObject *Sender, string aType, string aRawMessage)
{
    // here you will receive the ticker updates
}
```

Coinbase Pro | Get Market Data

Coinbase는 REST Endpoint를 통해 공개 Market Data를 제공하며, 이러한 엔드포인트 중 하나를 호출하면 요청한 시장 데이터의 snapshot을 받습니다.
Market Data 엔드포인트는 인증이 필요하지 않으므로 모든 사용자가 자유롭게 사용할 수 있습니다.

예제: BTC-USD 호가창의 스냅샷을 가져오려면 다음 호출을 수행하십시오

```
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase(this);  
ShowMessage(oCoinbase->REST_API->GetPublicProductBook("BTC-USD"));
```


Coinbase Pro | Private REST API

Coinbase REST API는 public 및 private 엔드포인트를 제공합니다. Private 엔드포인트는 거래의 보안을 강화하기 위해 메시지에 서명할 것을 요구합니다.

먼저 Coinbase 계정에 로그인하여 새 API를 생성해야 하며, 다음 값을 얻게 됩니다:

- ApiKey
- ApiSecret

이러한 필드는 Coinbase API 클라이언트 구성 요소의 Coinbase 속성에 구성해야 합니다. 구성되면 Coinbase REST API에 private 요청을 시작할 수 있습니다

```
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

Coinbase에 private 요청을 할 때 요청의 보안을 높이기 위해 메시지에 서명합니다. 메시지는 로컬 시간을 가져와 서명된 메시지 내부에 보냅니다. 로컬 시간이 Coinbase 서버와 30초보다 큰 차이가 있으면 요청이 거부됩니다. 따라서 로컬 시간이 동기화되었는지 확인하는 것이 중요하며, OS의 시간 동기화 방법을 사용하여 이를 수행할 수 있습니다.

GetTime 메서드를 호출하여 Coinbase Pro 서버 시간을 확인할 수 있으며, 이는 Coinbase Pro 서버의 시간을 반환합니다

Coinbase Pro | Place Orders

Coinbase에서 새 주문을 배치하려면 먼저 비공개 데이터에 액세스하기 위한 API 키가 필요합니다. 다음 문서를 확인하십시오: [How to Use Private REST API](#).

API 키를 구성하면 주문을 실행할 수 있습니다.

Market Order

새 시장가 주문 실행, BTC-USD 계약 0.002개 매수

```
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

새 지정가 주문을 넣어 BTC-USD 0.002 계약을 가격 한도 10000에 매수합니다

```
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

Coinbase는 실제 자금 없이 거래할 수 있는 SandBox 계정 사용을 허용합니다. 이 계정에는 프로덕션 계정과 다른 API 키 생성이 필요합니다.

SandBox 계정을 사용하려면 REST API에 요청하기 전에 **Coinbase.SandBox** 속성을 **true**로 설정하기만 하면 됩니다.

```
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

ASP.NET Core SignalR은 앱에 실시간 웹 기능을 추가하는 것을 단순화하는 오픈 소스 라이브러리입니다. 실시간 웹 기능은 서버 측 코드가 클라이언트에 콘텐츠를 즉시 푸시할 수 있게 합니다.

SignalR에 적합한 후보:

- 서버로부터 고빈도 업데이트가 필요한 앱입니다. 예로는 게임, 소셜 네트워크, 투표, 경매, 지도 및 GPS 앱이 있습니다.
- 대시보드 및 모니터링 앱입니다. 예로는 회사 대시보드, 즉각적인 판매 업데이트 또는 여행 알림이 있습니다.
- 협업 앱. 화이트보드 앱과 팀 회의 소프트웨어가 협업 앱의 예입니다.
- 알림이 필요한 앱입니다. 소셜 네트워크, 이메일, 채팅, 게임, 여행 알림 및 기타 많은 앱이 알림을 사용합니다.

SignalRCore sgcWebSockets 구성 요소는 WebSocket을 전송으로 사용하여 SignalRCore 서버에 연결하며, 이 전송이 지원되지 않으면 오류가 발생합니다.

Hubs

SignalRCore는 허브를 사용하여 클라이언트와 서버 간에 통신합니다. SignalRCore는 두 가지 허브 프로토콜을 제공합니다: JSON 기반 텍스트 프로토콜과 MessagePack 기반 바이너리 프로토콜입니다. sgcWebSockets 구성 요소는 SignalRCore 서버와 통신하기 위해 JSON 텍스트 프로토콜만 구현합니다.

클라이언트가 사용할 Hub를 구성하려면, 클라이언트가 서버에 연결하기 전에 **SignalRCore/Hub** 속성에 Hub 이름을 설정하기만 하면 됩니다.

연결

클라이언트가 서버에 새 연결을 열면, 형식 protocol과 버전을 포함하는 요청 메시지를 보냅니다. sgcWebSockets는 항상 형식 protocol을 JSON으로 보냅니다. 서버는 protocol이 서버에서 지원되지 않는 경우 오류로 응답하며, 이 오류는 **OnSignalRCoreError** 이벤트를 사용하여 처리할 수 있고, 연결이 성공하면 **OnSignalRCoreConnect** 이벤트가 호출됩니다.

클라이언트가 SignalRCore 서버에 연결하면 세션 간 클라이언트를 식별하는 ConnectionId를 보낼 수 있으므로, 연결 해제가 발생하면 클라이언트는 동일한 이전 connection id를 전달하여 서버에 다시 연결할 수 있습니다. 새 connection id를 얻으려면 서버에 정상적으로 연결하기만 하면 되며 **OnBeforeConnectEvent**를 사용하여 ConnectionId를 알 수 있습니다. 서버에 다시 연결하고 이전 connection id를 전달하려면 **ReConnect** 메서드를 사용하고 **ConnectionId**를 매개변수로 전달하십시오.

SignalRCore Protocol

SignalR Protocol은 모든 Message 기반 전송을 통한 양방향 RPC를 위한 프로토콜입니다. 연결의 어느 당사자든 다른 당사자에서 프로시저를 호출할 수 있으며, 프로시저는 0개 이상의 결과 또는 오류를 반환할 수 있습니다. 예제: 클라이언트는 서버에서 메서드를 요청할 수 있고 서버는 클라이언트에서 메서드를 요청할 수 있습니다. 다음 메시지가 서버와 클라이언트 간에 교환됩니다:

- **HandshakeRequest:** 클라이언트가 메시지 형식에 합의하기 위해 서버에 보냅니다.
- **HandshakeResponse:** 서버가 클라이언트에 이전 HandshakeRequest 메시지의 확인으로 응답합니다. 핸드셰이크가 실패하면 오류를 포함합니다.
- **Close:** 연결이 닫힐 때 클라이언트 또는 서버가 호출합니다. 오류로 인해 연결이 닫힌 경우 오류를 포함합니다.
- **Invocation:** 클라이언트 또는 서버가 인수가 있거나 없는 메서드를 호출하기 위해 다른 피어에 메시지를 보냅니다.
- **StreamInvocation:** 클라이언트 또는 서버가 인수가 있거나 없는 스트리밍 메서드를 호출하기 위해 다른 피어에게 메시지를 보냅니다. 응답은 다른 항목으로 분할됩니다.
- **StreamItem:** 이전 StreamInvocation의 응답입니다.
- **Completion:** 이전 invocation 또는 StreamInvocation이 완료되었음을 의미합니다. 프로세스가 성공한 경우 결과를 포함하거나 오류가 있는 경우 오류를 포함할 수 있습니다.
- **CancellInvocation:** 이전 StreamInvocation 요청을 취소합니다.

- **Ping:** 연결이 여전히 살아 있는지 확인하는 메시지입니다.

SignalRCore Encoding

SignalRCore를 사용하면 다음 인코딩을 사용할 수 있습니다:

- **JSON:** 현재 지원되는 유일한 인코딩입니다.
- **MessagePack**

현재 JSON만 지원되지만, 외부 messagepack 라이브러리를 사용하여 전송되는 메시지를 인코딩함으로써 MessagePack을 사용할 수 있습니다. 자세한 내용은 아래 MessagePack 섹션을 참조하십시오.

Encoding Protocol의 구성은 SignalRCore.Protocol 속성에 정의됩니다. 기본적으로 값은 **srcpJSON**입니다.

권한 부여

각 연결에 사용자를 연결하고 어떤 사용자가 리소스에 액세스할 수 있는지 필터링하기 위해 Authentication을 활성화할 수 있습니다. Authentication은 Bearer Token을 사용하여 구현됩니다: 클라이언트가 access token을 제공하고 서버가 이 token을 검증하여 사용자를 식별하는 데 사용합니다.

표준 웹 API에서는 bearer 토큰이 HTTP 헤더로 전송되지만, websocket을 사용할 때는 토큰이 쿼리 문자열 매개변수로 전송됩니다.

다음 메서드가 지원됩니다:

srcpRequestToken

Authentication이 활성화된 경우, 흐름은 다음과 같습니다:

1. 먼저 서버에서 유효한 토큰을 가져오려고 시도합니다. Authentication.RequestToken.URL에 대해 HTTP 연결을 열고 User와 Password 데이터를 사용하여 POST를 수행합니다.
2. 이전 단계가 성공하면 토큰이 반환됩니다. 그렇지 않으면 오류가 반환됩니다.
3. 토큰이 반환되면 새 연결을 협상하기 위해 새 HTTP 연결을 엽니다. 여기서 토큰은 HTTP 헤더로 전달됩니다.
4. 이전 단계가 성공하면, websocket 연결을 열고 토큰을 쿼리 문자열 매개변수로 전달합니다.

- **Authentication.Enabled:** 활성화되면 websocket 연결이 설정되기 전에 권한 부여가 사용됩니다.
- **Authentication.Username:** 인증을 위해 서버에 제공되는 사용자 이름.
- **Authentication.Password:** 인증을 위해 서버에 제공되는 비밀번호입니다.
- **Authentication.RequestToken.PostFieldUsername:** 사용자 이름을 전송할 필드의 이름입니다(구성에 따라 다름). 어떤 이름이 사용되는지 확인하려면 http javascript 페이지를 참조하십시오).
- **Authentication.RequestToken.PostFieldPassword:** 비밀번호를 전송하는 필드의 이름입니다(구성에 따라 다름, 사용되는 이름을 보려면 http javascript 페이지를 확인하십시오).
- **Authentication.RequestToken.URL:** 토큰을 요청하는 url입니다.
- **Authentication.RequestToken.QueryFieldToken:** websocket 연결에 사용되는 쿼리 문자열 매개변수의 이름입니다.

srcpSetToken

여기서는 토큰을 SignalRCore 서버에 직접 전달합니다(토큰이 다른 서버에서 획득되었기 때문).

- **Authentication.Enabled:** 활성화된 경우, websocket 연결이 설정되기 전에 권한 부여가 사용됩니다.
- **Authentication.SetToken.Token:** 얻은 토큰 값입니다.

액세스 토큰은 쿼리 매개변수로 전송하거나(기본 옵션) HTTP 헤더에 Bearer Token으로 전송할 수 있습니다. 이 동작을 구성하려면 Authentication.TokenParam 속성을 사용하십시오.

- **srcpQuery:** access_token이 websocket 연결의 쿼리 url에 전달됩니다.
- **srcpHeader:** access_token이 Bearer Token으로 http 헤더로 전달됩니다.

srcpBasic

이 옵션은 **Basic Authentication**을 사용하며, 이 인증 방법은 SignalRCore 구성 요소와 [TsgcWebSocketClient](#) 구성이 필요합니다.

예: 서버가 basic 인증을 요구하고 사용자 이름이 "user", 비밀번호가 "secret"인 경우, 아래와 같이 구성 요소를 구성하십시오.

```
// 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;
```

클라이언트와 서버 간의 통신

서버와 클라이언트 간에는 세 가지 종류의 상호 작용이 있습니다:

Invocations

Caller는 Callee에게 메시지를 보내고 호출이 완료되었음을 나타내는 메시지와 선택적으로 호출 결과를 기대합니다

예: 클라이언트가 SendMessage 메서드를 호출하고 매개변수로 사용자 이름과 텍스트 메시지를 전달합니다. 서버로부터 결과 메시지를 받기 위해 Invocation Id를 보냅니다.

```
SignalRCore->Invoke("SendMessage", ARRAYOFCONST(("John", "Hello All.")), "id-000001");

void OnSignalRCoreCompletion(TObject *Sender, TSignalRCore_Completion *Completion)
{
    if (Completion->Error != "")
    {
        ShowMessage("Something goes wrong.");
    }
    else
    {
        ShowMessage("Invocation Successful!");
    }
}
```

Non-Blocking Invocations

Caller는 Callee에 메시지를 보내고 이 호출에 대한 추가 메시지를 기대하지 않습니다. Invocation ID 값 없이 호출을 보낼 수 있습니다. 이는 호출이 "non-blocking"임을 나타냅니다.

예제: 클라이언트가 SendMessage 메서드를 호출하고 매개변수로 사용자 이름과 텍스트 메시지를 전달합니다. 클라이언트는 호출 결과에 대한 서버의 응답을 기대하지 않습니다.

```
SignalRCore->Invoke("SendMessage", ARRAYOFCONST(("John", "Hello All.")));
```

Streaming Invocations

Caller는 Callee에 메시지를 보내고, Callee가 반환하는 하나 이상의 결과와 그 뒤에 호출 종료를 나타내는 메시지를 기대합니다.

예: 클라이언트가 Counter 메서드를 호출하고 500밀리초 간격으로 10개의 숫자를 요청합니다.

```
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

단일 호출을 수행하기 위해 Caller는 다음 기본 흐름을 따릅니다:

```
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)
```

호출을 나타내기 위해 고유한 Invocation ID 값(Caller가 선택한 임의의 문자열)을 할당하십시오. 호출되는 Target, Arguments 및 InvocationId(InvocationId를 보내지 않으면 완료 결과를 받지 못합니다)를 포함하여 Invoke 또는 InvokeStream 메서드를 호출하십시오.

Invocation이 논블로킹으로 표시된 경우(아래의 "Non-Blocking Invocations" 참조) 여기서 중지하고 즉시 애플리케이션으로 제어를 반환하십시오. 일치하는 Invocation ID를 가진 StreamItem 또는 Completion 메시지를 처리하십시오.

```
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!");
        }
    }
}
```

단일 호출을 실행하고 완료를 기다릴 수 있습니다.

```
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);
```


호출을 나타내기 위해 고유한 Invocation ID 값(Caller가 선택한 임의 문자열)을 할당하십시오. 호출되는 Target, Arguments 및 InvocationId를 포함하는 InvokeAndWait 또는 InvokeStreamAndWait 메서드를 호출하십시오. 프로그램은 완료 이벤트가 호출되거나 시간 초과를 초과할 때까지 기다립니다.

```
{
    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

클라이언트가 서버가 Completion 메시지를 보내기 전에 StreamItem 메시지 수신을 중지하려는 경우, 스트림을 시작한 StreamInvocation 메시지에 사용된 것과 동일한 InvocationId로 CancelInvocation 메시지를 보낼 수 있습니다.

```
void OnSignalRCoreStreamItem(TObject *Sender, TSignalRCore_StreamItem *StreamItem, bool &Cancel)
{
    if (StreamItem->InvocationId == "id-000002")
    {
        Cancel = true;
    }
}
```

Client Results

Invocation은 Completion 메시지가 수신될 때만 완료된 것으로 간주됩니다. 클라이언트가 서버로부터 Invocation을 받으면 OnSignalRCoreInvocation 이벤트가 호출됩니다.

```
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

연결이 닫힐 때 클라이언트가 보냅니다. 오류로 인해 연결이 닫힌 경우 오류 사유를 포함합니다.

```
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

SignalR Hub 프로토콜은 기본 전송 연결이 활성 상태로 유지되도록 보장하는 데 사용되는 "Keep Alive" 메시지를 지원합니다. 이러한 메시지는 다음을 보장하는 데 도움이 됩니다:

프록시가 유휴 시간(메시지가 거의 전송되지 않을 때) 동안 기본 연결을 닫지 않습니다. 기본 연결이 정상적으로 종료되지 않고 끊어진 경우, 애플리케이션에 가능한 한 빨리 알립니다.

Keep alive 동작은 Ping 메시지를 호출하거나 WebSocket 클라이언트에서 HeartBeat를 활성화하여 달성됩니다. 서버가 클라이언트에 ping을 보내면 클라이언트는 자동으로 응답을 보내고 OnSignalRCoreKeepAlive 이벤트가 호출됩니다.

```
void OnSignalRCoreKeepAlive(TObject *Sender)
{
    DoLog("#keepalive");
}
```

MessagePack

SignalR 프로토콜의 MsgPack 인코딩에서, 각 Message는 주어진 hub 프로토콜 메시지의 속성에 해당하는 항목을 포함하는 단일 MsgPack 배열로 표현됩니다. 배열 항목은 기본 값, 배열(예: 메서드 인수) 또는 객체(예: 인수 값)일 수 있습니다. 배열의 첫 번째 항목은 메시지 유형입니다.

전송되는 메시지를 인코딩하는 방법을 보려면 [MessagePack 문서](#)를 참조하십시오.

새 메시지가 수신될 때마다, 이는 OnSignalRCoreMessagePack 이벤트에서 디스패치됩니다. 메시지는 Data Stream 매개 변수를 읽어 액세스할 수 있습니다. 매개 변수 JSON은 기본적으로 비어 있으며, MessagePack 메시지를 JSON으로 변환하면 구성 요소는 JSON 메시지를 인코딩이 JSON을 사용하는 것처럼 처리합니다(따라서 OnSignalRCoreCompletion, OnSignalRCoreInvocation... 이벤트가 디스패치됩니다).

API SignalR

SignalR

SignalR 구성 요소는 SignalR 서버에 연결하기 위한 전송으로 WebSocket을 사용합니다. 이 전송이 지원되지 않으면 오류가 발생합니다.

SignalR 클라이언트 구성 요소에는 다음 데이터를 설정할 수 있는 `SignalR`이라는 속성이 있습니다:

- **Hubs:** 클라이언트가 구독하는 허브 목록을 포함합니다.
- **ProtocolVersion:** 클라이언트가 사용하는 프로토콜 버전으로, 1.2에서 1.5까지의 프로토콜 버전을 지원합니다
- **UserAgent:** SignalR 서버에 연결하는 데 사용되는 사용자 에이전트입니다.

클라이언트는 Text 또는 Binary 데이터 전송을 지원합니다.

Hubs Messages

Hubs API를 사용하면 클라이언트에서 서버 메시지를, 서버에서 클라이언트 메시지를 호출할 수 있습니다. 영구 연결에 사용되는 프로토콜은 RPC(원격 프로시저 호출) 의미론을 표현할 만큼 풍부하지 않습니다. 그러나 이것이 허브 연결에 사용되는 프로토콜이 영구 연결에 사용되는 프로토콜과 완전히 다르다는 것을 의미하지는 않습니다. 오히려 허브 연결에 사용되는 프로토콜은 대부분 영구 연결 프로토콜의 확장입니다.

클라이언트가 서버 메시지를 호출할 때, 영구 연결에서처럼 더 이상 자유 형식 문자열을 전송하지 않습니다. 대신 메시지를 호출하는 데 필요한 모든 정보를 포함하는 JSON 문자열을 전송합니다. 다음은 클라이언트가 서버 메시지를 호출하기 위해 전송하는 샘플 메시지입니다:

```
WriteData("{\"H\":\"chathub\",\"M\":\"Send\",\"A\":[\"CBuilder Client\",\"Test message\"],\"I\":0}
```

페이로드에는 다음 속성이 있습니다:

I – invocation 식별자 – 응답을 요청과 일치시킬 수 있게 합니다

H – 허브의 이름

M – 메시지의 이름

A – arguments(배열, 메시지에 매개변수가 없는 경우 비어 있을 수 있음)

문자열 인수에 **큰따옴표**가 있으면 "를 \"로 교체하십시오.

예: 인수가 {"test":1}인 경우, 인수를 {"test\\":1}로 보내십시오.

```
WriteData("{\"H\":\"chathub\",\"M\":\"Send\",\"A\":{\"test\\\":1},\"I\":0}");
```

권한 부여

각 연결에 사용자를 연결하고 어떤 사용자가 리소스에 액세스할 수 있는지 필터링하기 위해 Authentication을 활성화할 수 있습니다. Authentication은 Bearer Token을 사용하여 구현됩니다: 클라이언트가 access token을 제공하고 서버가 이 token을 검증하여 사용자를 식별하는 데 사용합니다.

현재 Bearer Tokens만 지원됩니다:

여기서는 토큰을 Signal 서버에 직접 전달합니다(토큰이 다른 서버에서 얻어졌기 때문입니다).

- **Authentication.Enabled:** 활성화된 경우, websocket 연결이 설정되기 전에 권한 부여가 사용됩니다.
- **Authentication.Authentication:** 2가지 유형의 인증이 지원됩니다: bearer 토큰 또는 쿠키. 둘 다 필요한 값을 가져 오는 외부 방법이 필요합니다.

- **BearerToken:** 획득한 토큰 값.
- **Cookie:** 필요한 쿠키의 값을 설정합니다.

```
TsgcWSAPI_Signal *oSignalR = new TsgcWSAPI_Signal(NULL);
oSignalR->SignalR->Enabled = true;
oSignalR->SignalR->Authentication = srcBearerToken;
oSignalR->SignalR->BearerToken->Token = "token here";
```

구성 요소에는 다음 이벤트가 있습니다:

OnSignalRConnect

이 이벤트는 클라이언트가 서버에 성공적으로 연결될 때 발생합니다.

OnSignalRDisconnect

이 이벤트는 클라이언트가 서버에서 연결이 끊어질 때 발생합니다.

OnSignalRError

WebSocket 연결에 오류가 있을 때 이 이벤트가 호출됩니다.

OnSignalRMessage

persistent 연결에 사용되는 프로토콜은 매우 간단합니다. 서버에 전송되는 메시지는 raw 문자열일 뿐입니다. 특정 형식이 필요하지 않습니다. 클라이언트에 전송되는 메시지는 더 구조화되어 있습니다. 메시지에서 찾을 수 있는 속성은 다음과 같습니다:

C – 메시지 id, 모든 비-KeepAlive 메시지에 존재함
M – 실제 데이터를 포함하는 배열입니다.

```
{"C": "d-9B7A6976-B, 2|C, 2", "M": ["Welcome!"]}
```

OnSignalRBinary

이 이벤트는 서버로부터 바이너리 데이터가 수신될 때 호출됩니다.

OnSignalRResult

서버 메시드가 호출되면, 서버는 호출이 완료되었다는 확인을 클라이언트에 invocation id를 보냄으로써 반환하고, 메시드가 값을 반환한 경우 반환 값을, 메시드 호출이 실패한 경우 오류를 반환합니다. 다음은 서버 메시드 호출의 샘플 결과입니다:

```
{"I": "0"}
```

호출 식별자가 "0"이었던 서버 void 메시드가 성공적으로 완료되었습니다.

```
{"I": "0", "R": 42}
```

호출 식별자가 "0"인 숫자를 반환하는 서버 메서드가 성공적으로 완료되어 값 42를 반환했습니다.

```
{"I": "0", "E": "Error occurred"}
```

OnSignalRKeepAlive

이 이벤트는 서버로부터 KeepAlive 메시지가 수신될 때 발생합니다.

API Kraken

Kraken

개요

WebSockets API는 실시간 시장 데이터 업데이트를 제공합니다. WebSockets는 가장 빠른 실시간 데이터를 제공하는 양방향 프로토콜로, 실시간 애플리케이션을 구축하는 데 도움이 됩니다. 아래에 제시된 공개 메시지 유형은 인증이 필요하지 않습니다. 비공개 데이터 메시지는 별도의 인증된 엔드포인트에서 구독할 수 있습니다.

Kraken은 공개 시장 데이터와 (인증이 필요한) 비공개 사용자 데이터를 제공하는 REST API도 제공합니다.

구성

Private API는 Kraken 계정에서 API 키를 생성해야 합니다.

Kraken은 WebSocket 프로토콜에서 테스트 환경을 허용합니다. 이 베타 기능을 사용하려면 Kraken 속성에서 Beta 속성을 활성화하십시오.

지원되는 API

- [WebSockets Public API](#): 공개 WebSocket 서버에 연결합니다.
- [WebSockets Private API](#): 비공개 WebSocket 서버에 연결하며 서버에 대해 인증하기 위해 API Key 및 API Secret이 필요합니다.
- [REST Public API](#): 공개 REST 서버에 연결합니다.
- [REST Private API](#): public REST 서버에 연결하며 서버에 대해 인증하기 위해 API Key와 API Secret이 필요합니다.

Kraken 예제

Public WebSocket 서버에 연결하는 방법

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient(NULL);
TsgcWSAPI_Kraken *oKraken = new TsgcWSAPI_Kraken(NULL);
oKraken->Client = oClient;
oClient->Active = true;
```

비공개 WebSocket 서버에 연결하는 방법

```
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;
```

REST API에서 Ticker를 가져오는 방법

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient(NULL);
TsgcWSAPI_Kraken *oKraken = new TsgcWSAPI_Kraken(NULL);
```

```
oKraken->Client = oClient;
ShowMessage(oKraken->GetTicker(ARRAYOFCNST(("XBTUSD"))));
```

REST API 메서드

Public Endpoints

메서드	Arguments	설명
GetSystemStatus		현재 시스템 상태를 가져옵니다.

Private Endpoints

메서드	Arguments	설명
GetExtendedBalance		확장 잔고 정보를 가져옵니다.
AmendOrder		기존 주문 수정.
CancelAllOrders		모든 미체결 주문을 취소합니다.
CancelAllOrdersAfter		Dead man's switch - 시간 초과 후 모든 주문을 취소합니다.
EditOrder		기존 주문을 편집합니다.
AddOrderBatch		여러 주문을 배치로 추가합니다.
CancelOrderBatch		여러 주문을 일괄 취소합니다.
GetWithdrawalMethods		사용 가능한 출금 방법을 가져옵니다.
GetWithdrawalAddresses		출금 주소를 가져옵니다.

REST API에서 계정 잔액을 가져오는 방법

```
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

연결

URL: wss://ws.kraken.com (v1) 또는 wss://ws.kraken.com/v2 (v2, 권장)

구성 요소는 이제 **Version** 속성(기본값: 2)을 통해 WebSocket API v2를 지원합니다. 레거시 v1 엔드포인트를 사용하려면 `Kraken.Version := 1`을 설정하십시오.

소켓이 열리면 subscribe 요청 메시지를 보내 공개 채널을 구독할 수 있습니다.

일반 고려 사항

- WebSocket을 통해 송수신되는 모든 메시지는 JSON 형식으로 인코딩됩니다
- 모든 부동 소수점 필드(타임스탬프 포함)는 정밀도를 보존하기 위해 따옴표로 묶입니다.
- 각 거래 가능한 쌍의 형식은 A/B이며, 여기서 A와 B는 표준화된 자산의 경우 ISO 4217-A3이고 표준화되지 않은 경우 인기 있는 고유 심볼입니다.
- 타임스탬프는 고유한 것으로 간주되어서는 안 되며 트랜잭션 id의 별칭으로 간주되어서는 안 됩니다. 또한 타임스탬프의 세분성은 트랜잭션 속도를 대표하지 않습니다.

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

Methods

Ping

클라이언트는 서버에 ping을 보내 연결이 살아 있는지 확인할 수 있으며, 서버는 pong으로 응답합니다. 이것은 서버에서 시작되는 WebSockets 표준의 기본 ping과 달리 애플리케이션 수준 ping입니다

Ticker

Ticker 정보에는 지정된 통화 쌍에 대한 최우선 매도 및 최우선 매수 가격, 24시간 거래량, 마지막 거래 가격, 거래량 가중 평균 가격 등이 포함됩니다. ticker 메시지는 거래 또는 일련의 거래가 발생할 때마다 게시됩니다.

SubscribeTicker 메서드를 호출하여 ticker를 구독하십시오:

```
SubscribeTicker(["XBT/USD"]);
```

구독이 성공하면 **OnKrakenSubscribed** 이벤트가 호출됩니다:

```
void OnKrakenSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription, string ChannelName, int ReqID)
{
    DoLog("#subscribed: " + Subscription + " " + Pair + " " + ChannelName);
}
```


UnSubscribeTicker 메서드를 호출하여 구독 해제:

```
UnSubscribeTicker(ARRAYOFCONST((L"XBT/USD")));
```

구독 취소가 성공하면, OnKrakenUnSubscribed 이벤트가 호출됩니다:

```
void OnKrakenUnSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#unsubscribed: " + Subscription + " " + Pair);
}
```

구독 / 구독 취소를 시도하는 동안 오류가 있으면 OnKrakenSubscriptionError 이벤트가 호출됩니다.

```
void OnKrakenSubscriptionError(TObject *Sender, string ErrorMessage, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#subscription error: " + ErrorMessage);
}
```

Ticker 업데이트는 OnKrakenData 이벤트에서 알립니다.

```
[
    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

OHLC를 구독한 경우, (endtime에 관계없이) 마지막 유효 캔들의 스냅샷이 전송된 후 진행 중인 캔들에 대한 업데이트가 이어 집니다. 예를 들어, 1분 캔들을 구독하고 5분 동안 거래가 없었다면, 5분 전의 마지막 1분 캔들 스냅샷이 게시됩니다. endtime을 사용하여 그것이 오래된 캔들인지 확인할 수 있습니다.

SubscribeOHLC 메서드를 호출하여 OHLC를 구독하십시오. pair와 interval을 전달해야 합니다.

```
SubscribeOHLC(ARRAYOFCONST((L"XBT/USD")), kin1min);
```

구독이 성공하면 OnKrakenSubscribed 이벤트가 호출됩니다.

```
void OnKrakenSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription, string ChannelName,
int ReqID)
{
    DoLog("#subscribed: " + Subscription + " " + Pair + " " + ChannelName);
}
```

UnSubscribeOHLC 메서드를 호출하여 구독을 해제합니다.

```
UnSubscribeOHLC(ARRAYOFCONST(("XBT/USD")), kin1min);
```

구독 취소가 성공하면, OnKrakenUnSubscribed 이벤트가 호출됩니다:

```
void OnKrakenUnSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
int ReqID)
{
    DoLog("#unsubscribed: " + Subscription + " " + Pair);
}
```

구독 / 구독 취소를 시도하는 동안 오류가 있으면 OnKrakenSubscriptionError 이벤트가 호출됩니다.

```
void OnKrakenSubscriptionError(TObject *Sender, string ErrorMessage, string Pair, string Subscription,
int ReqID)
{
    DoLog("#subscription error: " + ErrorMessage);
}
```

OHLC 업데이트는 OnKrakenData 이벤트에서 알립니다.

```
[
    42,
    [
        "1542057314.748456",
        "1542057360.435743",
        "3586.70000",
        "3586.70000",
        "3586.60000",
        "3586.60000",
        "3586.68894",
        "0.03373000",
        2
    ],
    "ohlcv-5",
    "XBT/USD"
]
```

Trade

통화 쌍에 대한 거래 피드입니다.

SubscribeTrade 메서드를 호출하여 Trade feed를 구독합니다.

```
SubscribeTrade(ARRAYOFCONST((L"XBT/USD")));
```

구독이 성공하면 OnKrakenSubscribed 이벤트가 호출됩니다.

```
void OnKrakenSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
string ChannelName, int ReqID)
{
    DoLog("#subscribed: " + Subscription + " " + Pair + " " + ChannelName);
}
```

UnSubscribeTrade 메서드를 호출하여 UnSubscribe:

```
UnSubscribeTrade(ARRAYOFCONST((L"XBT/USD")));
```

구독 취소가 성공하면, OnKrakenUnSubscribed 이벤트가 호출됩니다:

```
void OnKrakenUnSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#unsubscribed: " + Subscription + " " + Pair);
}
```

구독 / 구독 취소를 시도하는 동안 오류가 있으면 OnKrakenSubscriptionError 이벤트가 호출됩니다.

```
void OnKrakenSubscriptionError(TObject *Sender, string ErrorMessage, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#subscription error: " + ErrorMessage);
}
```

거래 업데이트는 OnKrakenData 이벤트에서 알림이 전달됩니다.

```
[
  0,
  [
    [
      "5541.20000",
      "0.15850568",
      "1534614057.321597",
      "s",
      "l",
      ""
    ],
    [
      "6060.00000",
      "0.02455000",
      "1534614057.324998",
      "b",
      "l",
      ""
    ]
  ],
  "trade",
  "XBT/USD"
]
```

Book

주문서 레벨입니다. 구독 시 지정된 깊이로 스냅샷이 게시되며, 스냅샷에 이어 레벨 업데이트가 게시됩니다. SubscribeBook 메서드를 호출하여 Book을 구독하십시오. pair와 depth를 전달해야 합니다.

```
SubscribeBook(ARRAYOFCONST((L"XBT/USD")), kde10);
```

구독이 성공하면 OnKrakenSubscribed 이벤트가 호출됩니다.

```
void OnKrakenSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription, string ChannelName,
    int ReqID)
{
    DoLog("#subscribed: " + Subscription + " " + Pair + " " + ChannelName);
}
```

UnSubscribeBook 메서드를 호출하여 구독 해제:

```
UnSubscribeBook(ARRAYOFCONST((L"XBT/USD")), kde10);
```

구독 취소가 성공하면, OnKrakenUnSubscribed 이벤트가 호출됩니다:

```
void OnKrakenUnSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
    int ReqID)
```

```
{
  DoLog("#unsubscribed: " + Subscription + " " + Pair);
}
```

구독 / 구독 취소를 시도하는 동안 오류가 있으면 OnKrakenSubscriptionError 이벤트가 호출됩니다.

```
void OnKrakenSubscriptionError(TObject *Sender, string ErrorMessage, string Pair, string Subscription,
  int ReqID)
{
  DoLog("#subscription error: " + ErrorMessage);
}
```

Book 업데이트는 OnKrakenData 이벤트에서 알림이 표시됩니다.

```
[
  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

구독한 asset 쌍에 대한 최고 bid 및 ask 가격을 표시하는 Spread 피드입니다. Bid 거래량과 ask 거래량도 메시지의 일부입니다.

SubscribeSpread 메서드를 호출하여 Spread 피드를 구독합니다.

```
SubscribeSpread(ARRAYOFCNST(("XBT/USD")));
```

구독이 성공하면 OnKrakenSubscribed 이벤트가 호출됩니다.

```
void OnKrakenSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription, string ChannelName,
  int ReqID)
{
  DoLog("#subscribed: " + Subscription + " " + Pair + " " + ChannelName);
}
```

UnSubscribeSpread 메서드를 호출하여 구독 취소:

```
UnSubscribeSpread(ARRAYOFCONST((L"XBT/USD")));
```

구독 취소가 성공하면, OnKrakenUnSubscribed 이벤트가 호출됩니다:

```
void OnKrakenUnSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#unsubscribed: " + Subscription + " " + Pair);
}
```

구독 / 구독 취소를 시도하는 동안 오류가 있으면 OnKrakenSubscriptionError 이벤트가 호출됩니다.

```
void OnKrakenSubscriptionError(TObject *Sender, string ErrorMessage, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#subscription error: " + ErrorMessage);
}
```

Spread 업데이트는 OnKrakenData 이벤트에서 알립니다.

```
[
    0,
    [
        "5698.40000",
        "5700.00000",
        "1542057299.545897",
        "1.01234567",
        "0.98765432"
    ],
    "spread",
    "XBT/USD"
]
```

기타 메서드

한 가지 메서드로 모든 채널을 구독/구독 해제할 수 있습니다.

```
SubscribeAll(ARRAYOFCONST((L"XBT/USD")));
UnSubscribeAll(ARRAYOFCONST((L"XBT/USD")));
```

모든 채널이 구독된 경우 OHLC 간격 값은 1입니다.

Events

OnConnect: websocket 클라이언트가 서버에 연결될 때입니다.

OnKrakenConnect: 성공적인 websocket 연결 후 그리고 서버가 시스템 상태를 보낼 때 호출됩니다.

OnKrakenSystemStatus: 시스템 상태가 변경될 때 호출됩니다.

OnKrakenSubscribed: 채널 구독이 성공한 후 호출됩니다.

OnKrakenUnSubscribed: 채널에서 구독 취소에 성공한 후 호출됩니다.

OnKrakenSubscriptionError: 구독/구독 취소를 시도하는 중 오류가 있을 경우 호출됩니다.

OnKrakenData: 채널 구독에 업데이트가 있을 때마다 호출됩니다.

API Kraken | WebSockets Private API

연결

URL: wss://ws-auth.kraken.com (v1) 또는 wss://ws-auth.kraken.com/v2 (v2, 권장)
이제 구성 요소는 **Version** 속성(기본값: 2)을 통해 WebSocket API v2를 지원합니다.

소켓이 열리면 인증된 subscribe 요청 메시지를 보내 비공개 데이터 채널을 구독할 수 있습니다.

Authentication

API 클라이언트는 WebSockets Private 엔드포인트에 연결하기 위해 다음 REST API 엔드포인트 "GetWebSocketsToken"을 통해 인증 "token"을 요청해야 합니다. 토큰은 생성 후 15분 이내에 사용해야 합니다. WebSockets API private 메시지(openOrders 또는 ownTrades)에 대한 연결이 유지되는 한 토큰은 만료되지 않습니다. Websockets Token을 얻으려면 Kraken Options Component에 API Key와 API Secret을 설정해야 합니다. 계정에서 Kraken이 제공한 api 키입니다.

```
Kraken->ApiKey = "api key";
Kraken->ApiSecret = "api secret";
```

Methods

OwnTrades

자신의 거래 목록을 가져옵니다. 첫 구독 시 최신 50건의 거래 목록을 가져옵니다

```
SubscribeOwnTrades();
```

나중에 UnSubscribeOwnTrades 메서드를 호출하여 OwnTrades에서 구독을 취소할 수 있습니다

```
UnSubscribeOwnTrades();
```

서버의 응답 예제

```
[
  [
    {
      "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"
]
```

Open Orders

인증된 사용자 API 키에 속하는 모든 미결 주문을 표시하는 피드입니다. 초기 스냅샷은 모든 미결 주문 목록을 제공하고 그 다음 미결 주문 목록에 대한 모든 업데이트가 전송됩니다. 'closed'와 같은 상태 변경 업데이트의 경우, 페이로드에 orderid 및 status 필드가 존재합니다

```
SubscribeOpenOrders();
```

나중에 UnSubscribeOpenOrders 메서드를 호출하여 OpenOrders에서 구독 해제할 수 있습니다

```
UnSubscribeOpenOrders();
```

서버의 응답 예제

```
[
  [
    {
      "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

Send a new Order to Kraken

```
TsgcWSKrakenOrder *oKrakenOrder = new TsgcWSKrakenOrder();
oKrakenOrder->Pair = "XBT/USD";
oKrakenOrder->_Type = kosBuy;
oKrakenOrder->OrderType = kotMarket;
oKrakenOrder->Volume = 1;
AddOrder(oKrakenOrder);
```

Order 매개변수 목록

```
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

```

서버의 응답 예제

```

{
  "descr": "buy 0.01770000 XBTUSD @ limit 4000",
  "event": "addOrderStatus",
  "status": "ok",
  "txid": "ONPNXH-KMKMU-F4MR5V"
}

```

주문 취소

주문 취소

```
CancelOrder("Order Id");
```

서버의 응답 예제

```

{
  "event": "cancelOrderStatus",
  "status": "ok"
}

```


API Kraken | REST Public API

연결

URL: <https://api.kraken.com>

Kraken Public API는 인증이 필요하지 않습니다.

구성

유일한 구성은 REST HTTP 요청에 대한 로그를 활성화할지 여부입니다. 모든 HTTP 요청/응답을 텍스트 파일 로그에 저장하려면 HTTPLogOptions를 활성화하십시오

Events

OnKrakenHTTPException: 이 이벤트는 REST Api에서 HTTP 요청을 수행하는 동안 예외가 있을 경우 호출됩니다.

Methods

GetServerTime

이 메서드는 서버와 클라이언트 간의 시간 편차를 근사하는 데 도움이 됩니다. 시간을 Unix 형식으로 반환합니다.

```
{"error": [], "result": {"unixtime": 1586705546, "rfc1123": "Sun, 12 Apr 20 15:32:26 +0000"}}
```

GetAssets

Assets에 대한 정보를 반환합니다

```
{"error": [], "result": {"ADA": {"aclass": "currency", "altname": "ADA", "decimals": 8, "display_decimals": 6}}}}
```

GetAssetPairs

한 쌍의 자산에 대한 정보를 반환합니다

```
Kraken->REST_API->GetAssetPairs(ARRAYOFCNST((L"XBTUSD")));
```

GetTicker

ticker 정보를 반환합니다

```
Kraken->REST_API->GetTicker(ARRAYOFCNST((L"XBTUSD")));
```

GetOHLC

Open-High-Low-Close 데이터를 반환합니다.

```
Kraken->REST_API->GetOHLC("XBTUSD");
```

GetOrderBook

Array 페어 이름과 시장 깊이를 반환합니다.

```
Kraken->REST_API->GetOrderBook("XBTUSD");
```

GetTrades

페어의 최근 거래 데이터를 반환합니다.

```
Kraken->REST_API->GetTrades("XBTUSD");
```

GetSpread

페어의 최근 스프레드 데이터를 반환합니다.

```
Kraken->REST_API->GetSpread("XBTUSD");
```

API Kraken | REST Private API

연결

URL: <https://api.kraken.com>

Authentication

REST Private API는 API Key와 API Secret을 필요로 하며, 이 값들은 계정에서 Kraken이 제공합니다.

```
Kraken->ApiKey = "api key";  
Kraken->ApiSecret = "api secret";
```

Methods

GetAccountBalance

계정 잔액을 반환합니다.

```
Kraken->REST_API->GetAccountBalance();
```

GetTradeBalance

거래에 대한 정보를 반환합니다.

```
Kraken->REST_API->GetTradeBalance();
```

GetOpenOrders

미체결 주문 목록을 반환합니다.

```
Kraken->REST_API->GetOpenOrders();
```

GetClosedOrders

종료된 주문 목록을 반환합니다.

```
Kraken->REST_API->GetClosedOrders();
```

QueryOrders

주문에 대한 정보를 쿼리합니다.

```
Kraken->REST_API->QueryOrders("1234");
```

GetTradesHistory

거래 정보의 배열을 반환합니다.

```
Kraken->REST_API->GetTradesHistory();
```

QueryTrades

거래에 대한 정보를 쿼리합니다.

```
Kraken->REST_API->QueryTrades("1234");
```

GetOpenPositions

포지션 정보를 반환합니다.

```
Kraken->REST_API->GetOpenPositions("1234");
```

GetLedgers

원장 정보의 연관 배열을 반환합니다.

```
Kraken->REST_API->GetLedgers();
```

QueryLedgers

원장 정보의 연관 배열을 반환합니다.

```
Kraken->REST_API->QueryLedgers("1234");
```

GetTradeVolume

거래량 정보를 반환합니다.

```
Kraken->REST_API->GetTradeVolume();
```

AddExport

새 보고서 내보내기를 추가합니다.

```
Kraken->REST_API->AddExport("Report All Trades");
```

ExportStatus

보고서 상태 가져오기

```
Kraken->REST_API->ExportStatus();
```

RetrieveExport

report id로 Report를 가져옵니다.

```
Kraken->REST_API->RetrieveExport("GOCO");
```

RemoveExport

report id로 Report를 제거합니다.

```
Kraken->REST_API->RemoveExport("GOCO");
```

Add Order

새 주문을 추가합니다

```
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

id로 미체결 주문을 취소합니다

```
Kraken->REST_API->CancelOrder("1234");
```

API Kraken Futures

Kraken Futures

개요

REST API를 사용하면 Kraken Futures 계정의 메서드에 안전하게 액세스할 수 있습니다. REST API 메서드의 예:

- 현재 또는 과거 가격 정보 요청
- 계정 잔액 및 PnL 확인
- 마진 매개변수와 예상 청산 임계값
- 주문 발주 또는 취소(개별 또는 일괄)
- 미체결 주문 보기
- 미결제 포지션 또는 거래 내역
- 디지털 자산 출금을 요청합니다

이러한 메서드를 "endpoint"라고 하며 REST API 섹션에서 설명합니다.

Websocket API를 사용하면 Kraken Futures 플랫폼에 안전하게 통신 채널을 설정하여 실시간으로 정보를 수신할 수 있습니다. 이를 통해 지속적으로 요청을 전송하는 대신 업데이트를 수신할 수 있습니다. 이러한 채널을 subscription이라고 합니다.

일부 엔드포인트는 디지털 자산 출금 시작과 같은 민감한 작업을 수행할 수 있도록 합니다. 이러한 엔드포인트에 안전하게 액세스하기 위해, API는 National Security Agency가 개발한 암호화 기술을 사용합니다.

구성

API를 사용하려면, 고유한 **API 키** 쌍을 생성해야 합니다(비공개 API에 접근하려는 경우):

1. **Kraken Futures** 계정에 로그인합니다.
2. 오른쪽 상단 모서리에서 이름을 클릭하십시오.
3. 드롭다운 메뉴에서 "Settings"를 선택하십시오.
4. API 패널에서 "Create Key" 탭을 선택하십시오.
5. "Create Key" 버튼을 누르십시오.
6. 공개 및 개인 키를 확인하고 안전한 곳에 기록하십시오.

공개 키와 비공개 키를 구성 요소의 **KrakenOptions** 속성에 복사하십시오.

```
KrakenOptions.ApiKey
KrakenOptions.ApiSecret
```

지원되는 API

- **WebSockets Public API**: 공개 WebSocket 서버에 연결합니다.
- **WebSockets Private API**: private WebSocket 서버에 연결하며 서버에 대해 인증하기 위해 API Key와 API Secret이 필요합니다.
- **REST Public API**: 공개 REST 서버에 연결합니다.
- **REST Private API**: 공개 REST 서버에 연결하며 서버에 대해 인증하기 위해 API Key와 API Secret이 필요합니다.

REST API 메서드

Private Endpoints

메서드	Arguments	설명
BatchOrder		배치 주문을 제출합니다(실행/취소/편집).
GetOrderStatus		특정 주문의 상태를 가져옵니다.
GetPNLCurrencyPreferences		PNL 통화 환경 설정을 가져옵니다.
SetPNLCurrencyPreference		심볼에 대한 PNL 통화 환경 설정을 지정합니다.
GetLeverageSettings		레버리지 환경 설정을 가져옵니다.
SetLeverageSettings		symbol에 대한 leverage를 설정합니다.

API Kraken Futures | WebSockets Public API

연결

URL: wss://futures.kraken.com/ws/v1

소켓이 열리면 subscribe 요청 메시지를 보내 공개 채널을 구독할 수 있습니다.

Methods

Ticker

이 엔드포인트는 현재 상장된 모든 Futures 계약과 인덱스에 대한 현재 시장 데이터를 반환합니다. 인증이 필요하지 않습니다. **SubscribeTicker** 메서드를 호출하여 ticker를 구독하십시오:

```
SubscribeTicker(ARRAYOFCONST(("PI_XBTUSD")));
```

구독이 성공하면 **OnKrakenFuturesSubscribed** 이벤트가 호출됩니다:

```
void OnKrakenFuturesSubscribed(TObject *Sender, string Feed, string ProductId)
{
    DoLog("#subscribed: " + Feed + " " + ProductId);
}
```

UnSubscribeTicker 메서드를 호출하여 UnSubscribe합니다:

```
UnSubscribeTicker(ARRAYOFCONST(("PI_XBTUSD")));
```

구독 취소가 성공하면 **OnKrakenFuturesUnSubscribed** 이벤트가 호출됩니다:

```
void OnKrakenFuturesUnSubscribed(TObject *Sender, string Feed, string ProductId)
{
    DoLog("#unsubscribed: " + Feed + " " + ProductId);
}
```

구독 / 구독 취소를 시도하는 중 오류가 발생하면 **OnKrakenFuturesError** 이벤트가 호출됩니다.

```
void OnKrakenFuturesError(TObject *Sender, string Error)
{
    DoLog("#error: " + Error);
}
```

Ticker 업데이트는 **OnKrakenData** 이벤트에서 알림됩니다.

```
{    "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

trade 피드는 체결된 거래에 대한 정보를 반환합니다
SubscribeTrade 메서드를 호출하여 Trade feed를 구독하십시오.

```
SubscribeTrade(ARRAYOFCONST(("PI_XBTUSD")));
```

구독이 성공하면 **OnKrakenFuturesSubscribed** 이벤트가 호출됩니다:

```
void OnKrakenFuturesSubscribed(TObject *Sender, string Feed, string ProductId,)
{
  DoLog("#subscribed: " + Feed + " " + ProductId);
}
```

UnSubscribeTrade 메서드를 호출하여 구독을 해제하십시오:

```
UnSubscribeTrade(ARRAYOFCONST(("PI_XBTUSD")));
```

구독 취소가 성공하면 **OnKrakenFuturesUnSubscribed** 이벤트가 호출됩니다:

```
void OnkrakenFuturesUnSubscribed(TObject *Sender, string Feed, string ProductId)
{
  DoLog("#unsubscribed: " + Feed + " " + ProductId);
}
```

구독 / 구독 취소를 시도하는 중 오류가 발생하면 **OnKrakenFuturesError** 이벤트가 호출됩니다.

```
void OnKrakenFuturesError(TObject *Sender, string Error)
{
  DoLog("#error: " + Error);
}
```

거래 업데이트는 **OnKrakenData** 이벤트에서 알립니다.

```
{  "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

이 피드는 주문서에 대한 정보를 반환합니다.
SubscribeBook 메서드를 호출하여 Book을 구독하십시오. Symbol을 전달해야 합니다.

```
SubscribeBook(ARRAYOFCONST(("PI_XBTUSD")));
```

구독이 성공하면 **OnKrakenFuturesSubscribed** 이벤트가 호출됩니다:

```
void OnKrakenFuturesSubscribed(TObject *Sender, string Feed, string ProductId)
{
  DoLog("#subscribed: " + Feed + " " + ProductId);
}
```

UnSubscribeBook 메서드를 호출하여 구독 취소:

```
UnSubscribeBook(ARRAYOFCNST(("PI_XBTUSD")));
```

구독 취소가 성공하면 **OnKrakenFuturesUnSubscribed** 이벤트가 호출됩니다:

```
void OnKrakenFuturesUnSubscribed(TObject *Sender, string Feed, string ProductId)
{
    DoLog("#unsubscribed: " + Feed + " " + ProductId);
}
```

구독 / 구독 취소를 시도하는 중 오류가 발생하면 **OnKrakenFuturesError** 이벤트가 호출됩니다.

```
void OnKrakenFuturesError(TObject *Sender, string Error)
{
    DoLog("#error: " + Error);
}
```

Book 업데이트는 **OnKrakenData** 이벤트에서 알립니다.

```
{
    "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

ticker lite 피드는 상장된 상품에 대한 ticker 정보를 반환합니다.

SubscribeTickerLite 메서드를 호출하여 Spread 피드를 구독합니다.

```
SubscribeTickerLite(ARRAYOFCONST(("PI_XBTUSD")));
```

구독이 성공하면 **OnKrakenFuturesSubscribed** 이벤트가 호출됩니다:

```
void OnKrakenFuturesSubscribed(TObject *Sender, string Feed, string ProductId)
{
    DoLog("#subscribed: " + Feed + " " + ProductId);
}
```

UnSubscribeTickerLite 메서드를 호출하여 구독을 해제하십시오:

```
UnSubscribeTickerLite(ARRAYOFCONST(("PI_XBTUSD")));
```

구독 취소가 성공하면 **OnKrakenFuturesUnSubscribed** 이벤트가 호출됩니다:

```
void OnKrakenFuturesUnSubscribed(TObject *Sender, string Feed, string ProductId)
{
    DoLog("#unsubscribed: " + Feed + " " + ProductId);
}
```

구독 / 구독 취소를 시도하는 중 오류가 발생하면 **OnKrakenFuturesError** 이벤트가 호출됩니다.

```
void OnKrakenFuturesError(TObject *Sender, string Error)
{
    DoLog("#error: " + Error);
}
```

Spread 업데이트는 **OnKrakenData** 이벤트에서 알립니다.

```
{ "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

하트비트 피드는 정해진 간격으로 하트비트 메시지를 게시합니다.

```

SubscribeHeartBeat();
UnSubscribeHeartBeat();

```

Events

OnConnect: websocket 클라이언트가 서버에 연결될 때입니다.

OnKrakenFuturesConnect: websocket 연결이 성공한 후 그리고 서버가 시스템 상태를 보낼 때 호출됩니다.

OnKrakenFuturesSubscribed: 채널에 대한 구독이 성공한 후 호출됩니다.

OnKrakenFuturesUnSubscribed: 채널에서 성공적으로 구독 해제된 후 호출됩니다.

OnKrakenFuturesError: 구독/구독 취소 중에 오류가 있으면 호출됩니다.

OnKrakenData: 채널 구독에 업데이트가 있을 때마다 호출됩니다.

API Kraken Futures | WebSockets Private API

연결

URL: wss://futures.kraken.com/ws/v1

Authentication

WebSocket 프라이빗 피드에 대한 구독 및 구독 해제 요청에는 사용자 api_secret으로 서명된 챌린지 메시지가 필요합니다. 챌린지는 (api_key를 사용하여) WebSocket API Public 섹션에 표시된 대로 얻습니다.

인증된 요청은 원본 챌린지 메시지(original_challenge)와 서명된 챌린지(signed_challenge)를 모두 JSON 형식으로 포함해야 합니다.

Websockets Challenge를 얻으려면, API Key와 API Secret을 Kraken Options 구성 요소에 설정해야 하며, api 키는 계정에서 Kraken이 제공합니다

```
Kraken->ApiKey = "api key";
Kraken->ApiSecret = "api secret";
```

Methods

Open Orders Verbose

이 구독 피드는 사용자 미결제 주문에 대한 정보를 게시합니다. 이 피드는 장부 교차에 실패한 모든 post-only 주문에 대한 추가 정보를 더합니다.

```
SubscribeOpenOrdersVerbose();
```

나중에 **UnSubscribeOpenOrdersVerbose** 메서드를 호출하여 OpenOrdersVerbose 구독을 해제할 수 있습니다

```
UnSubscribeOpenOrdersVerbose();
```

서버의 응답 예제

```
{
  'feed': 'open_orders_verbose_snapshot',
  'account': '0f9c23b8-63e2-40e4-9592-6d5aa57c12',
  {
    'instrument': 'PI_XBTUSD',
    'time': 1567428848005,
    'last_update': ...
  }
}
```

미결제 포지션

이 구독 피드는 사용자 계정의 미결제 포지션을 게시합니다.

```
SubscribeOpenPositions();
```

나중에 **UnSubscribeOpenPositions** 메서드를 호출하여 OpenPositions에서 구독을 취소할 수 있습니다

```
UnSubscribeOpenPositions();
```

서버의 응답 예제

```
{
  "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
  }]
}
```

Account Log

이 구독 피드는 계정 정보를 게시합니다.

```
SubscribeAccountLog();
```

나중에 **UnSubscribeAccountLog** 메서드를 호출하여 AccountLog에서 구독을 취소할 수 있습니다

```
UnSubscribeAccountLog();
```

서버의 응답 예제

```
{
  '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

이 구독 피드는 fills 정보를 게시합니다.

```
SubscribeFills();
```

나중에 **UnSubscribeFills** 메서드를 호출하여 Fills를 구독 취소할 수 있습니다


```
UnSubscribeFills();
```

서버의 응답 예제

```
{
  "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"
    }
  ]
}
```

Open Orders

이 구독 피드는 사용자 미결제 주문에 대한 정보를 게시합니다.

```
SubscribeOpenOrders();
```

나중에 **UnSubscribeOpenOrders** 메서드를 호출하여 OpenOrders 구독을 취소할 수 있습니다

```
UnSubscribeOpenOrders();
```

서버의 응답 예제

```
{
  "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

이 구독 피드는 클라이언트 계정의 잔액 및 마진 정보를 반환합니다.

```
SubscribeAccountBalanceAndMargins();
```

나중에 **UnSubscribeAccountBalanceAndMargins** 메서드를 호출하여 AccountBalance 구독을 취소할 수 있습니다.

```
UnSubscribeAccountBalanceAndMargins();
```

서버의 응답 예제

```

{

"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

이 구독 피드는 클라이언트에 알림을 게시합니다.

```
SubscribeNotifications();
```

나중에 **UnSubscribeNotifications** 메서드를 호출하여 알림 구독을 취소할 수 있습니다.

```
UnSubscribeNotifications();
```

서버의 응답 예제

```

{

"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

연결

URL: <https://futures.kraken.com/derivatives/api/v3>

Kraken Futures Public API는 인증이 필요하지 않습니다.

구성

유일한 구성은 REST HTTP 요청에 대한 로그를 활성화할지 여부입니다. 모든 HTTP 요청/응답을 텍스트 파일 로그에 저장하려면 HTTPLogOptions를 활성화하십시오

Events

OnKrakenHTTPException: 이 이벤트는 REST Api에서 HTTP 요청을 수행하는 동안 예외가 있을 경우 호출됩니다.

Methods

GetFeeSchedules

이 엔드포인트는 모든 수수료 일정을 나열합니다. 인증이 필요하지 않습니다.

```
KrakenFutures->REST_API->GetFeeSchedules();
```

Order Book

이 엔드포인트는 현재 상장된 Futures 계약의 전체 비누적 주문 호가창을 반환합니다.

```
KrakenFutures->REST_API->GetOrderBook("PI_XBTUSD");
```

Tickers

이 엔드포인트는 현재 상장된 모든 Futures 계약과 인덱스에 대한 현재 시장 데이터를 반환합니다.

```
KrakenFutures->REST_API->GetTickers();
```

Instruments

이 엔드포인트는 현재 상장된 모든 Futures 계약 및 지수에 대한 사양을 반환합니다.

```
KrakenFutures->REST_API->GetInstruments();
```

History

이 엔드포인트는 지정된 lastTime 값에서 마지막 100개의 거래를 반환합니다. 값을 지정하지 않으면 마지막 100개의 거래를 반환합니다. 이 엔드포인트는 호출된 시점 또는 마지막 trading engine 릴리스 이후(둘 중 더 빠른 시점) 최대 7일 동안의 거래 기록만 반환합니다.

```
KrakenFutures->REST_API->GetHistory("PI_XBTUSD");
```

API Kraken Futures | REST Private API

연결

URL: <https://futures.kraken.com/derivatives/api/v3>

Authentication

REST Private API는 API Key와 API Secret을 필요로 하며, 이 값들은 계정에서 Kraken이 제공합니다.

```
Kraken->ApiKey = "api key";
Kraken->ApiSecret = "api secret";
```

Methods

EditOrderByOrderId

이 엔드포인트는 현재 상장된 Futures 계약에 대한 기존 주문을 편집할 수 있도록 합니다.

aOrderId: 편집하려는 주문의 ID입니다.

aSize: 주문과 연관된 크기

aLimitPrice: 주문과 연관된 지정가입니다.

aStopPrice: 스톱 주문과 연관된 스톱 가격. 이전 Order Type이 Stop인 경우 필수입니다.

```
KrakenFutures->REST_API->EditOrderByOrderId("Order_Id", 2, 1000);
```

EditOrderByCliOrderId

이 엔드포인트는 현재 상장된 Futures 계약에 대한 기존 주문을 편집할 수 있도록 합니다.

aCliOrderId: 사용자가 지정하는 주문 식별자. 전역적으로 고유해야 합니다.

aSize: 주문과 연관된 크기

aLimitPrice: 주문과 연관된 지정가입니다.

aStopPrice: stop 주문과 관련된 stop 가격입니다. Order Type이 Stop인 경우 필수입니다.

```
KrakenFutures->REST_API->EditOrderByCliOrderId("Cli_Order_Id", 2, 1000);
```

SendMarketOrder

이 엔드포인트를 사용하면 시장가 주문을 보낼 수 있습니다.

aSide: 주문의 방향입니다: buy 또는 sell.

aSymbol: futures의 symbol입니다

aSize: 주문과 연결된 크기입니다.

```
KrakenFutures->REST_API->SendMarketOrder(kosfBuy, "PI_XBTUSD", 1);
```

SendLimitOrder

이 엔드포인트를 통해 Limit Order를 보낼 수 있습니다.

aSide: 주문의 방향입니다: buy 또는 sell.
aSymbol: futures의 symbol입니다
aSize: 주문과 연결된 크기입니다.
aLimitPrice: 주문과 연관된 지정가입니다.

```
KrakenFutures->REST_API->SendLimitOrder(kosfBuy, "PI_XBTUSD", 1, 1000);
```

SendStopOrder

이 엔드포인트를 사용하면 Stop Order를 보낼 수 있습니다.

aSide: 주문의 방향입니다: buy 또는 sell.
aSymbol: futures의 symbol입니다
aSize: 주문과 연결된 크기입니다.
aStopPrice: stop order와 연결된 stop 가격입니다.
aLimitPrice: 주문과 연관된 지정가입니다.

```
KrakenFutures->REST_API->SendStopOrder(kosfBuy, "PI_XBTUSD", 1, 1000, 900);
```

SendTakeProfitOrder

이 엔드포인트를 통해 Take Profit Order를 보낼 수 있습니다.

aSide: 주문의 방향입니다: buy 또는 sell.
aSymbol: futures의 symbol입니다
aSize: 주문과 연결된 크기입니다.
aStopPrice: stop order와 연결된 stop 가격입니다.
aLimitPrice: 주문과 연관된 지정가입니다.

```
KrakenFutures->REST_API->SendTakeProfitOrder(kosfBuy, "PI_XBTUSD", 1, 1000, 900);
```

SendOrder

이 엔드포인트를 사용하면 현재 상장된 Futures 계약에 대해 limit, stop, take profit 또는 immediate-or-cancel 주문을 보낼 수 있습니다.

OrderType: 다음 중 하나를 선택하십시오: kotfLMT, kotfPOST, kotfMKT, kotfSTP, kotfTAKE_PROFIT, kotfIOC

Symbol: 선물 시장의 심볼

Side: 주문의 방향(buy 또는 sell)입니다.

Size: 주문에 연결된 크기입니다.

StopPrice: stop order와 연결된 stop 가격입니다.

LimitPrice: 주문과 연관된 지정가입니다.

TriggerSignal: Stop 또는 TakeProfit 주문을 넣는 경우, 트리거에 사용되는 신호로, kotsMark, kotsIndex, kotsLast 중 하나를 선택하십시오

CliOrderId: 사용자가 지정하는 주문 식별자입니다. 전역적으로 고유해야 합니다.

ReduceOnly: 주문이 기존 포지션을 줄이기만 하기를 원하면 true로 설정하십시오. 기존 포지션을 늘리는 주문은 거부됩니다. 기본값 false.

```
TsgHTTPKrakenFuturesOrder *oOrder = new TsgHTTPKrakenFuturesOrder(this);
try
{
    oOrder->Side = kosfBuy;
    oOrder->Symbol = "PI_XBTUSD";
    oOrder->OrderType = kotfMKT;
    oOrder->Size = 1;
    KrakenFutures->REST_API->SendOrder(oOrder);
}
```



```

}
__finally
{
    oOrder->Free;
}

```

CancelOrderByOrderId

이 엔드포인트는 Futures 계약의 미체결 주문을 취소할 수 있게 합니다.

aOrderId: 편집하려는 주문의 ID입니다.

```
KrakenFutures->REST_API->CancelOrderByOrderId("Order_Id");
```

CancelOrderByCliOrderId

이 엔드포인트는 Futures 계약의 미체결 주문을 취소할 수 있게 합니다.

aCliOrderId: 사용자가 지정하는 주문 식별자. 전역적으로 고유해야 합니다.

```
KrakenFutures->REST_API->CancelOrderByCliOrderId("Cli_Order_Id");
```

GetFills

이 엔드포인트는 모든 선물 계약에 대한 체결된 주문 정보를 반환합니다.

aLastFillDate: 제공되지 않으면 모든 futures 계약의 마지막 100개 fill을 반환합니다. 제공되면 lastFillTime 이전의 100개 항목을 반환합니다.

```
KrakenFutures->REST_API->GetFills("2020-07-22T13:45:00.000Z");
```

Transfer

이 엔드포인트를 사용하면 동일한 담보 통화를 가진 두 margin 계정 간에 또는 margin 계정과 cash 계정 간에 자금을 이체할 수 있습니다.

aFromAccount: 자금을 이체할 cash 또는 margin 계정의 이름입니다.

aToAccount: 자금을 이동할 cash 또는 margin 계정의 이름.

aUnit: 이체할 단위입니다.

aAmount: 이체할 금액입니다.

```
KrakenFutures->REST_API->Transfer("FI_XBTUSD", "cash", "xbt", 1.5);
```

GetOpenPositions

이 엔드포인트는 Futures 계약의 모든 미체결 포지션의 크기와 평균 진입 가격을 반환합니다. 여기에는 만기되었지만 아직 정산되지 않은 Futures 계약이 포함됩니다.

```
KrakenFutures->REST_API->GetOpenPositions();
```

GetNotifications

이 엔드포인트는 플랫폼의 알림을 제공합니다.

```
KrakenFutures->REST_API->GetNotifications();
```

GetAccounts

이 엔드포인트는 현금 계정 또는 마진 계정일 수 있는 모든 Kraken Futures 계정과 관련된 주요 정보를 반환합니다. 여기에는 디지털 자산 잔액, 상품 잔액, 마진 요구 사항, 마진 트리거 추정치, 그리고 사용 가능한 자금, 미결제 포지션의 PnL 및 포트폴리오 가치와 같은 보조 정보가 포함됩니다.

```
KrakenFutures->REST_API->GetAccounts();
```

CancelAllOrders

이 엔드포인트는 Futures 계약의 미결제 주문을 취소할 수 있게 합니다.

Symbol: 모든 미결제 주문을 취소할 선물 상품(선택 사항)

```
KrakenFutures->REST_API->CancelAllOrders();
```

CancelAllOrdersAfter

이 엔드포인트는 네트워크 오작동으로부터 클라이언트를 보호하기 위한 Dead Man's Switch 메커니즘을 제공합니다. 클라이언트는 초 단위 시간 초과가 있는 요청을 전송할 수 있으며, 이는 시간 초과가 만료되면 모든 클라이언트 주문을 취소하는 카운트다운 타이머를 트리거합니다.

aTimeout: 초 단위로 지정된 시간 초과입니다.

```
KrakenFutures->REST_API->CancelAllOrdersAfter(60);
```

GetOpenOrders

이 엔드포인트는 모든 Futures 계약에 대한 모든 미결제 주문 정보를 반환합니다.

```
KrakenFutures->REST_API->OpenOrders();
```

GetHistoricalOrders

이 엔드포인트는 계정에서 이루어진 과거 주문을 반환합니다.

aSince: 기준 DateTime

aBefore: 이전 DateTime

aSort: 오름차순 정렬은 "asc", 내림차순은 "desc"

aContinuationToken: 다음 사용 가능한 결과 집합을 반환하기 위해 호출에 사용할 수 있는 이전 응답에서 제공된 continuation token입니다

```
KrakenFutures->REST_API->GetHistoricalOrders(Now, Now - 5);
```

GetHistoricalTriggers

이 엔드포인트는 계정에서 수행된 과거 트리거를 반환합니다.

aSince: 기준 DateTime

aBefore: 이전 DateTime

aSort: 오름차순 정렬은 "asc", 내림차순은 "desc"

aContinuationToken: 다음 사용 가능한 결과 집합을 반환하기 위해 호출에 사용할 수 있는 이전 응답에서 제공된 continuation token입니다

```
KrakenFutures->REST_API->GetHistoricalTriggers(Now, Now - 5);
```

GetHistoricalExecutions

이 엔드포인트는 계정에서 수행된 과거 실행을 반환할 수 있습니다.

aSince: 기준 DateTime

aBefore: 이전 DateTime

aSort: 오름차순 정렬은 "asc", 내림차순은 "desc"

aContinuationToken: 다음 사용 가능한 결과 집합을 반환하기 위해 호출에 사용할 수 있는 이전 응답에서 제공된 continuation token입니다

```
KrakenFutures->REST_API->GetHistoricalExecutions(Now, Now - 5);
```

WithdrawalToSpotWallet

이 엔드포인트를 사용하면 Kraken Futures 지갑에서 Kraken Spot 지갑으로 디지털 자산을 출금하는 요청을 제출할 수 있습니다.

aCurrency: 출금할 디지털 자산입니다(예: xbt 또는 xrp).

aAmount: 출금할 통화의 양입니다.

```
KrakenFutures->REST_API->WithdrawalToSpotWallet("xbt", 1000);
```

GetFeeScheduleVolumes

이 엔드포인트는 30일간의 USD 거래량을 반환합니다.

```
KrakenFutures->REST_API->GetFeeScheduleVolumes();
```

GetAccountLogCSV

이 엔드포인트를 통해 클라이언트는 계정 로그의 csv 파일을 다운로드할 수 있습니다.

```
KrakenFutures->REST_API->GetAccountLogCSV();
```

API Pusher

Pusher

Pusher는 WebSocket 프로토콜을 기반으로 한 멋진 기능을 갖춘 쉽고 신뢰할 수 있는 플랫폼입니다: 유연한 pub/sub 메시징, 실시간 사용자 목록(presence), 인증 등.

Pusher WebSocket API는 7입니다.

데이터는 UTF8로 인코딩된 JSON을 포함하는 텍스트 데이터로 WebSocket을 통해 양방향으로 전송됩니다(바이너리 WebSocket 프레임은 지원되지 않습니다).

Ping 메시지를 호출하여 서버에 대한 연결을 테스트할 수 있습니다. 본질적으로 상대방으로부터 수신된 모든 메시지는 연결이 살아 있음을 의미하는 것으로 간주됩니다. 메시지가 없는 경우 어느 한쪽이 ping 메시지를 보내 상대방이 응답하는지 확인할 수 있으며, 상대방은 pong으로 응답해야 합니다.

연결하기 전에 다음 필드를 작성해야 합니다.

```
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)
```

중요

Pusher는 websocket 클라이언트가 이전 필드(key, cluster...)를 사용하여 URL에 연결할 것을 요구합니다. 이 필드들은 url을 빌드하는 데 사용되며, pusher 구성 요소에 클라이언트를 할당할 때 수행됩니다. 따라서 URL이 올바르게 빌드되도록 하려면 pusher 구성 필드를 채운 후 클라이언트를 설정하십시오. 아래에서 의사 코드를 확인하십시오:

```
// configure pusher fields
pusher.cluster = ...
pusher.key = ...
// set client
pusher.client = websocket client
// start connection
websocket client.Active = true;
```

성공적인 연결 후 **OnPusherConnect** 이벤트가 발생하고 다음 필드를 받습니다:

- Socket ID: 연결된 클라이언트의 고유 식별자.
- Timeout: 클라이언트가 ping 메시지를 시작해야 하는 서버 비활성 시간(초)입니다(이는 구성 요소에서 자동으로 처리됩니다).

오류 발생 시 **OnPusherError**가 발생하고 오류에 대한 정보가 제공됩니다. 오류는 유효하지 않은 인증, 유효하지 않은 명령 등에 대한 응답으로 Pusher에서 전송될 수 있습니다.

4000-4099

Pusher에 의해 연결이 닫히는 결과를 초래하는 오류를 나타내며, 동일한 매개변수를 사용하여 재연결을 시도해도 성공하지 못함을 나타냅니다.

- 4000: 애플리케이션이 SSL 연결만 수락하므로 wss://를 사용하여 다시 연결하십시오
- 4001: 애플리케이션이 존재하지 않습니다
- 4003: 애플리케이션 비활성화됨
- 4004: 애플리케이션이 연결 할당량을 초과했습니다
- 4005: 경로를 찾을 수 없음
- 4006: 잘못된 버전 문자열 형식
- 4007: 지원되지 않는 프로토콜 버전
- 4008: 프로토콜 버전이 제공되지 않음

4100-4199

Pusher가 연결을 닫는 결과를 초래하는 오류를 나타내며, 클라이언트가 1초 이상 후에 재연결할 수 있음을 나타냅니다.

- 4100: 용량 초과

4200-4299

Pusher가 연결을 닫게 되는 오류를 나타내며, 클라이언트가 즉시 다시 연결할 수 있음을 나타냅니다.

- 4200: 즉시 일반 재연결
- 4201: Pong 응답을 받지 못함: 클라이언트에 ping을 보냈으나 응답을 받지 못함 - ping 및 pong 메시지를 참조하십시오.
- 4202: 비활성으로 인해 종료됨: 클라이언트가 오랫동안(현재 24시간) 비활성 상태였으며 ping을 지원하지 않습니다. 최신 WebSocket draft로 업그레이드하거나 이 프로토콜의 버전 5 이상을 구현하십시오.

4300-4399

다른 모든 유형의 오류입니다.

- 4301: 속도 제한으로 인해 클라이언트 이벤트가 거부됨

Channels

채널은 Pusher의 기본 개념입니다. 각 애플리케이션에는 여러 채널이 있으며, 각 클라이언트는 어떤 채널을 구독할지 선택할 수 있습니다.

채널은 다음을 제공합니다:

- 데이터를 필터링하는 방법입니다. 예를 들어, 채팅 애플리케이션에서 'dogs'에 대해 논의하려는 사람들을 위한 채널이 있을 수 있습니다.
- 다양한 정보 스트림에 대한 액세스를 제어하는 방법입니다. 예를 들어, 프로젝트 관리 애플리케이션은 사람들이 'projectX'에 대한 업데이트를 받을 수 있도록 권한을 부여하려고 할 것입니다.

이벤트가 아니라 채널을 사용하여 데이터를 필터링하는 것이 강력히 권장됩니다. 이는 채널에 게시된 모든 이벤트가 이벤트 바 인딩에 관계없이 모든 구독자에게 전송되기 때문입니다.

채널은 명시적으로 생성할 필요가 없으며 클라이언트 요청 시 인스턴스화됩니다. 즉, 채널 생성이 쉽습니다. 클라이언트에게 구독하도록 지시하기만 하면 됩니다.

다음 유형의 채널이 지원됩니다:

- **공개 채널**은 이름을 아는 누구나 구독할 수 있습니다.
- **Private 채널**은 브로드캐스트하는 데이터에 대한 액세스를 서버가 제어할 수 있게 하는 메커니즘을 도입합니다
- **Presence 채널**은 private 채널의 확장입니다. 구독 시 사용자 정보를 등록할 수 있고, 채널의 다른 멤버에게 누가 온라인인지 알릴 수 있습니다
- **Cache 채널**은 마지막으로 트리거된 이벤트를 기억하고 새 구독자에게 첫 번째 이벤트로 전송합니다(public, private 및 presence 변형)
- **Private-Encrypted 채널**은 NaCl secretbox를 사용하여 종단간 암호화를 제공하므로 Pusher조차도 메시지 데이터를 읽을 수 없습니다

Public Channels

public 채널은 구독하기 위해 어떤 형태의 권한 부여도 필요하지 않으므로 공개적으로 액세스 가능한 데이터에 사용해야 합니다.

언제든지 채널을 구독하고 구독 취소할 수 있습니다. Pusher가 먼저 연결을 완료할 때까지 기다릴 필요가 없습니다.

예제: 채널 "my-channel"을 구독합니다.

```
CBuilder
APIPusher->Subscribe("my-channel");
```

구독에 성공하면 **OnPusherSubscribe** 이벤트가 발생하고, 오류가 있으면 **OnPusherError** 이벤트에서 메시지를 받습니다.

구독된 채널의 모든 메시지는 **OnPusherEvent** 이벤트에서 수신됩니다.

Publish 메서드가 호출되고 채널이 Public인 경우, 구성 요소는 WebSocket 프로토콜을 사용하는 대신 HTTP 프로토콜을 사용하여 TriggerEvent 메서드를 호출합니다(websocket 프로토콜을 사용한 publish는 허용되지 않음).

Private 채널

Indy 10.5.7 이상이 필요합니다

프라이빗 채널은 채널에 대한 접근을 어떤 식으로든 제한해야 할 때 사용해야 합니다. 사용자가 프라이빗 채널을 구독하려면 권한이 승인되어야 합니다.

예제: 채널 "my-private-channel"을 구독합니다.

```
CBuilder
APIPusher->Subscribe("my-private-channel", pscPrivateChannel);
```

구독에 성공하면 **OnPusherSubscribe** 이벤트가 발생하고, 오류가 있으면 **OnPusherError** 이벤트에서 메시지를 받습니다.

구독된 채널의 모든 메시지는 **OnPusherEvent** 이벤트에서 수신됩니다.

Presence 채널

Indy 10.5.7 이상이 필요합니다

Presence 채널은 Private 채널의 보안을 기반으로 하며 누가 해당 채널을 구독하고 있는지에 대한 인식이라는 추가 기능을 노출합니다. 이를 통해 채팅방 및 "온라인 상태" 유형 기능을 애플리케이션에 매우 쉽게 구축할 수 있습니다. 채팅방, 문서 공동 작업자, 동일한 웹 페이지를 보는 사람들, 게임의 경쟁자 등을 생각해 보십시오.

Presence 채널은 private 채널과 동일한 방식으로 클라이언트 API에서 구독되지만, 채널 이름 앞에 presence-를 붙여야 합니다. private 채널과 마찬가지로 현재 사용자가 채널에 접근할 권한이 있는지 확인하기 위해 구성 가능한 인증 URL로 HTTP 요청이 이루어집니다.

채널을 구독하고 구독 취소하는 사용자에 대한 정보는 presence 채널의 이벤트에 바인딩하여 접근할 수 있으며, 채널을 구독한 사용자의 현재 상태는 channel.members 속성을 통해 사용할 수 있습니다.

예제: "my-presence-channel" 채널을 구독합니다.

```
APIPusher->Subscribe("my-presence-channel", pscPresenceChannel,
    '{"user_id":"'John_Smith'", "user_info":{"name":"'John Smith'"}'}');
```

구독에 성공하면 **OnPusherSubscribe** 이벤트가 발생하고, 오류가 있으면 **OnPusherError** 이벤트에서 메시지를 받습니다.

구독된 채널의 모든 메시지는 **OnPusherEvent** 이벤트에서 수신됩니다.

Cache Channels

캐시 채널은 마지막으로 트리거된 이벤트를 기억하고 이를 새 구독자에게 첫 번째 이벤트로 보냅니다.

cache 채널에서 이벤트가 트리거되면 Pusher Channels는 이 이벤트를 캐시하고, 클라이언트가 cache 채널을 구독할 때 캐시된 값이 존재하면 해당 채널의 첫 번째 이벤트로 클라이언트에 전송됩니다. 이 동작은 개발자가 다른 곳에서 가져오는 추가 로직을 추가하지 않고도 초기 상태를 제공하는 데 도움이 됩니다.

다음 Cache 채널이 지원됩니다:

- Public Cache Channel
- Private Cache Channel
- Presence Cache Channel

예제: 공개 캐시 채널 "my-cache-channel"을 구독합니다.

```
APIPusher->Subscribe("my-cache-channel", pscCacheChannel);
```

구독에 성공하면 **OnPusherSubscribe** 이벤트가 발생하고, 오류가 있으면 **OnPusherError** 이벤트에서 메시지를 받습니다.

구독된 채널의 모든 메시지는 **OnPusherEvent** 이벤트에서 수신됩니다.

cache 채널을 구독할 때 캐시된 이벤트가 없으면 **OnPusherCacheMiss** 이벤트가 발생하여 채널 이름을 제공합니다. 이를 통해 애플리케이션은 캐시된 데이터가 없는 경우를 처리할 수 있습니다.

Private-Encrypted Channels

Private-Encrypted 채널은 메시지에 대한 종단간 암호화를 제공합니다. private 채널과 마찬가지로 인증이 필요하지만, 추가로 모든 데이터 페이로드가 NaCl secretbox를 사용하여 암호화되므로 권한이 부여된 구독자만 콘텐츠를 읽을 수 있습니다. Pusher 자체도 메시지를 복호화할 수 없습니다.

private-encrypted 채널을 사용하려면 인증 중에 **SharedSecret**을 제공해야 합니다. 공유 비밀은 메시지 데이터를 암호화하고 복호화하는 데 사용됩니다.

Example: private-encrypted 채널 "my-encrypted-channel"을 구독합니다.

```
APIPusher->Subscribe("my-encrypted-channel", pscPrivateEncryptedChannel);
```

암호화와 캐시 채널 동작을 결합한 private-encrypted-cache 변형도 사용할 수 있습니다:

```
APIPusher->Subscribe("my-encrypted-cache-channel", pscPrivateEncryptedCacheChannel);
```

private-encrypted 채널에서 **OnPusherAuthentication** 이벤트를 사용할 때, 응답 객체의 **SharedSecret** 속성을 설정하여 암호화 키를 제공할 수 있습니다:

```
void OnPusherAuthenticationEvent(TObject *Sender,
    TsgcWSPusherRequestAuthentication *AuthRequest,
    TsgcWSPusherResponseAuthentication *AuthResponse)
{
    AuthResponse->SharedSecret = "your-shared-secret-key";
}
```

Presence 이벤트

Presence 채널은 사용자가 채널에 가입하거나 떠날 때 애플리케이션에 알리는 추가 이벤트를 제공하며, 구독 수를 추적할 수 있게 합니다.

OnPusherMemberAdded

새 멤버가 presence 채널을 구독할 때 발생합니다. 가입한 멤버의 채널 이름, 사용자 ID 및 사용자 정보를 제공합니다.

```
void PUSHERPusherMemberAdded(TObject *Sender,
    String Channel, String UserId, String UserInfo)
{
    Log("Member joined: " + UserId + " on " + Channel);
}
```

OnPusherMemberRemoved

멤버가 presence 채널에서 구독을 취소할 때 발생합니다. 떠난 멤버의 채널 이름, 사용자 ID 및 사용자 정보를 제공합니다.

```
void PUSHERPusherMemberRemoved(TObject *Sender,
    String Channel, String UserId, String UserInfo)
{
    Log("Member left: " + UserId + " on " + Channel);
}
```

OnPusherSubscriptionCount

채널의 구독 수가 변경될 때 발생합니다. 채널 이름과 현재 구독자 수를 제공합니다. 이 이벤트는 Pusher 대시보드에서 활성화되어야 합니다.

```
void PUSHERPusherSubscriptionCount(TObject *Sender,
    String Channel, int SubscriptionCount)
{
    Log(Channel + " has " + IntToStr(SubscriptionCount) + " subscribers");
}
```

OnPusherCacheMiss

캐시된 이벤트가 없는 캐시 채널을 구독할 때 발생합니다. 채널 이름을 제공합니다. 이를 통해 애플리케이션이 캐시된 데이터가 없는 경우를 처리할 수 있습니다(예: 다른 소스에서 데이터를 가져옴).

```
void PUSHERPusherCacheMiss(TObject *Sender, String Channel)
{
    Log("Cache miss on: " + Channel);
}
```

메시지 게시

구독한 채널에서 메시지를 수신할 수 있을 뿐만 아니라, 다른 구독 사용자에게 메시지를 보낼 수도 있습니다.

채널의 모든 구독 사용자에게 메시지를 보내려면 **Publish** 메서드를 호출하십시오.

예시: "my-channel"의 구독된 모든 사용자에게 이벤트 전송

```
APIPusher->Publish("my-event", "my-channel");
```

클라이언트(연결)당 초당 10개 이하의 메시지를 게시하십시오. 이 속도 제한을 초과하여 트리거된 이벤트는 Pusher API에 의해 거부됩니다. 이것은 시스템 문제가 아니라 클라이언트 문제입니다. 한 채널에 있는 100개의 클라이언트가 이 속도로 메시지를 보내면 각자 초당 1,000개의 메시지를 처리해야 합니다! 일부 최신 브라우저는 이를 처리할 수 있을지 모르지만 대부분 좋은 생각은 아닙니다.

REST API

API는 `http://api-CLUSTER.pusher.com` 에서 호스팅되며, 여기서 CLUSTER는 자신의 앱 클러스터(예: eu)로 대체됩니다. HTTP 상태 코드는 요청의 성공 여부를 나타내는 데 사용됩니다. 다음 상태가 일반적입니다:

200 성공적인 요청. 본문에는 응답 데이터의 JSON 해시가 포함됩니다

400 오류: 자세한 내용은 응답 본문에 있습니다.

401 인증 오류: 응답 본문에 설명이 포함됩니다

403 Forbidden: 앱이 비활성화되었거나 메시지 할당량 초과

다음 REST API 함수가 구현되었습니다.

Function	설명
TriggerEvent	지정된 채널에서 새 이벤트를 트리거합니다. 선택적 SocketId(클라이언트를 제외하기 위함) 및 Info 매개변수를 지원합니다.
TriggerBatchEvents	단일 HTTP 요청으로 여러 이벤트를 트리거합니다. 이벤트 객체의 JSON 배열을 허용합니다.
GetChannels	모든 활성 채널의 목록을 제공합니다. 선택적 FilterByPrefix 및 Info 매개변수를 지원합니다.
GetChannel	특정 채널에 대한 정보를 제공합니다. 선택적 Info 매개변수를 지원합니다.
GetUsers	채널에 연결된 모든 사용자의 목록을 제공합니다.
TerminateUserConnections	사용자 ID로 식별되는 특정 사용자의 모든 연결을 종료합니다.

TriggerEvent

하나 이상의 채널에서 이벤트를 트리거합니다. 이벤트 이름, 채널 이름, 데이터 페이로드가 필요합니다.

Parameter	설명
aEventName	트리거할 이벤트의 이름.
aChannel	이벤트를 트리거할 채널 이름입니다.
aData	이벤트 데이터(JSON 문자열)입니다.
aSocketId (선택적)	이벤트 수신에서 제외할 소켓 ID입니다. 발신자가 자신의 메시지를 받지 않도록 하는 데 유용합니다.
aInfo (선택적)	응답에 포함할 속성의 심표로 구분된 목록입니다 (예: "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

단일 API 호출로 여러 이벤트를 트리거하며, 이는 각 이벤트에 대해 별도의 요청을 하는 것보다 효율적입니다. batch 매개변수는 이벤트 객체의 배열을 포함하는 JSON 문자열이어야 하며, 각 객체는 "channel", "name" 및 "data" 필드를 가집니다.


```
APIPusher->TriggerBatchEvents(
    [{"channel\\":\\"my-channel\\",\\"name\\":\\"my-event\\",\\"data\\":\\"hello\\"},
    {"channel\\":\\"my-channel-2\\",\\"name\\":\\"my-event\\",\\"data\\":\\"world\\"}]);
```

GetChannels

활성 채널 목록을 반환합니다. 결과를 필터링하고 추가 정보를 요청하기 위한 선택적 매개변수를 지원합니다.

Parameter	설명
aFilterByPrefix (선택적)	이름 접두사로 채널을 필터링합니다(예: presence 채널만 나열하려면 "presence-").
aInfo (선택적)	응답에 포함할 속성의 심표로 구분된 목록(예: "user_count").

```
// get all channels
APIPusher->GetChannels();

// get only presence channels with user count
APIPusher->GetChannels("presence-", "user_count");
```

GetChannel

특정 채널에 대한 정보를 반환합니다.

Parameter	설명
aChannel	정보를 가져올 채널 이름입니다.
aInfo (선택적)	포함할 속성의 심표로 구분된 목록입니다(예: "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

presence 채널에 연결된 사용자 목록을 반환합니다. 채널 이름에는 전체 접두사가 포함되어야 합니다(예: "presence-my-channel").

```
APIPusher->GetUsers("presence-my-channel");
```

TerminateUserConnections

주어진 사용자가 설정한 모든 연결을 종료합니다. 이는 특정 사용자가 모든 채널에서 연결을 끊도록 강제하는 데 사용할 수 있습니다. 사용자 ID는 사용자가 presence 채널을 구독할 때 사용한 "user_id"와 일치해야 합니다.

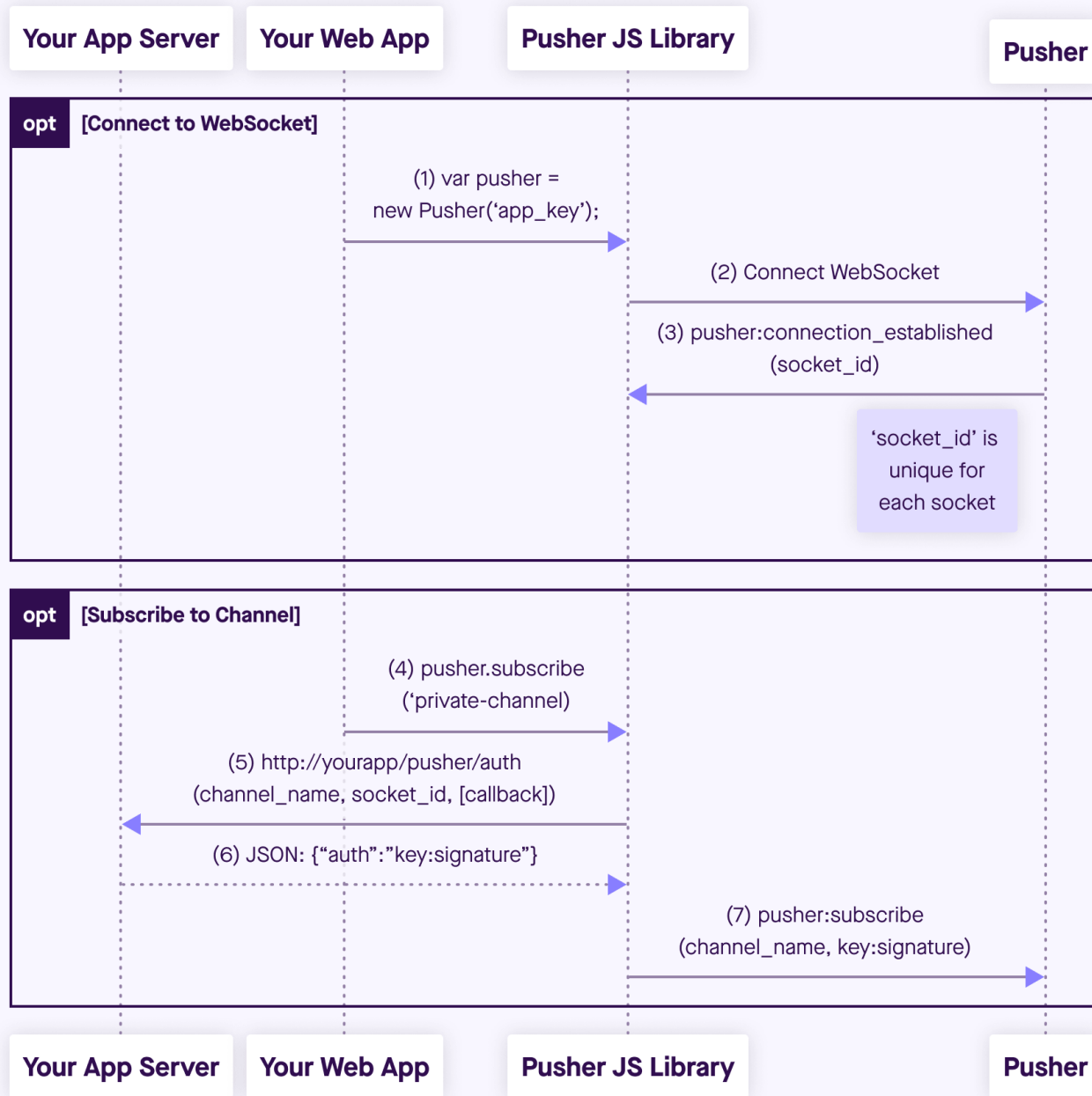
```
APIPusher->TerminateUserConnections("1234");
```

Custom Authentication

Pusher는 private 또는 presence 채널에 대한 구독만 허용합니다. 연결이 인증 토큰을 제공하면 액세스를 제한할 수 있습니다.

OnPusherAuthentication 이벤트를 사용하여 자체 인증 흐름을 구축할 수 있습니다. 이 이벤트는 구독 메시지가 Pusher가 제공한 비밀 키로 서명되기 전에 호출됩니다. 이 이벤트에는 SocketId, 채널 이름 등의 필드가 있는 요청 인증과 같은 2개의 매

개변수가 있으며, 이는 자체 인증 서버에서 요청을 인증하거나 인증하지 않는 데 사용할 수 있습니다. 아래에서 pusher 인증 흐름을 보여주는 스크린샷을 찾을 수 있습니다



클라이언트가 pusher 서버에 연결하면, pusher가 제공한 Key를 보내고 서버는 식별 id(socket_id)를 반환합니다. 클라이언트가 private(또는 presence) 채널을 구독하면, sgcWebSockets 클라이언트는 pusher가 제공한 Secret Key를 사용하여 구독 메시지에 포함되는 서명을 생성합니다. OnPusherAuthentication 이벤트를 사용하여 메시지 서명에 필요한 필드를 캡처하고, 자체 인증 방법을 구현하고, 성공하면 서명을 반환하면, 이 서명이 구독 메시지에 포함되어 서버로 전송됩니다.

예제:

```

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();
}
}

```

서명의 형식은 다음과 같습니다:

비공개 채널: key:HMAC256(SocketID, ChannelName)

Presence 채널: key: HMAC256(SocketID, ChannelName, Data)

TsgcWSPusherResponseAuthentication 객체는 다음 속성을 제공합니다:

속성	설명
Secret	HMAC 서명을 계산하는 데 사용되는 Pusher 비밀 키. 구성된 경우 Pusher.Secret로 미리 채워집니다.
서명	계산된 인증 서명입니다. 비워 두면 구성 요소가 Secret을 사용하여 자동으로 계산합니다.
SharedSecret	private-encrypted 채널에 대한 공유 시크릿 키입니다. pscPrivateEncryptedChannel 또는 pscPrivateEncryptedCacheChannel을 구독할 때 필요합니다. 메시지 데이터의 종단간 암호화에 사용됩니다.

API Bitmex

Bitmex

Bitmex는 암호화폐 거래소이자 파생상품 거래 플랫폼입니다.

다음 API가 지원됩니다:

1. **WebSocket streams:** 일부 메서드를 구독하여 데이터를 실시간으로 받을 수 있게 합니다. 이벤트는 서버에 의해 구독자에게 푸시됩니다. 프로토콜로 WebSocket을 사용합니다.
2. **REST API:** 클라이언트가 서버에 시장 및 계정 데이터를 요청할 수 있습니다. 인증을 위해 API Key와 Secret이 필요하며 프로토콜로 HTTPs를 사용합니다.

Properties

Bitmex API에는 2가지 유형의 메서드가 있습니다: 공개 및 비공개. 공개 메서드는 인증 없이 액세스할 수 있습니다(예: 티커 가격 가져오기). 일부는 비공개이며 사용자 데이터와 관련됩니다. 이러한 메서드는 Bitmex API 키 사용이 필요합니다.

- **ApiKey:** Bitmex 계정에서 새 api key를 요청하고, 그 값을 이 속성에 복사하면 됩니다.
- **ApiSecret:** API의 secret이며, 안전하게 보관하십시오.
- **TestNet:** 활성화되면 Bitmex Demo Account에 연결합니다(기본값 false).
- **HTTPLogOptions:** HTTP 요청 로그를 텍스트 파일에 저장합니다
 - **Enabled:** 활성화되면 WebSocket API의 모든 HTTP 요청을 저장합니다.
 - **FileName:** 로그가 저장될 파일 이름의 전체 경로

Most common uses

- **WebSockets API**
 - [WebSocket API에 연결하는 방법](#)
 - [WebSocket 채널 구독 방법](#)
- **REST API**
 - [Bitmex 주문을 넣는 방법](#)

WebSocket API

Subscribe / Unsubscribe

BitMEX는 실시간 데이터 구독을 허용합니다. 이 액세스는 연결되면 속도 제한이 없으며 프로그램에 가장 최신 데이터를 제공하는 가장 좋은 방법입니다. 일부 topic에서는 Symbol을 전달하여 심볼별로 이벤트를 필터링할 수 있습니다(예: trades, quotes...).

다음 구독 토픽은 인증 없이 사용할 수 있습니다:

- **btmAnnouncement:** 사이트 공지사항
- **btmChat:** Trollbox 채팅
- **btmConnected:** 연결된 사용자/bot의 통계
- **btmFunding:** 스왑 펀딩 비율 업데이트. 펀딩 간격마다 전송됨(일반적으로 8시간)
- **btmInstrument:** turnover 및 bid/ask를 포함한 상품 업데이트
- **btmInsurance:** 일일 Insurance Fund 업데이트
- **btmLiquidation:** book에 입력되는 Liquidation 주문
- **btmOrderBookL2_25:** level 2 호가창의 상위 25개 수준
- **btmOrderBookL2:** 전체 레벨 2 주문서
- **btmOrderBook10:** 전통적인 전체 book 푸시를 사용한 최상위 10개 수준
- **btmPublicNotifications:** 시스템 전체 알림(수명이 짧은 메시지에 사용됨)

- **btmQuote**: book의 최상위 수준
- **btmQuoteBin1m**: 1분 시세 bin
- **btmQuoteBin5m**: 5분 견적 bin
- **btmQuoteBin1h**: 1시간 호가 bin
- **btmQuoteBin1d**: 1일 quote bin
- **btmSettlement**: Settlements
- **btmTrade**: 실시간 거래
- **btmTradeBin1m**: 1분 거래 bin
- **btmTradeBin5m**: 5분 거래 bin
- **btmTradeBin1h**: 1시간 거래 bin
- **btmTradeBin1d**: 1일 거래 bin

다음 subject는 인증이 필요합니다:

- **btmAffiliate**: 총 추천 사용자 수 & 지급 % 등 제휴 상태
- **btmExecution**: 개별 체결, 주문당 여러 개일 수 있습니다.
- **btmOrder**: 주문에 대한 라이브 업데이트
- **btmMargin**: 현재 계정 잔액 및 margin 요구 사항에 대한 업데이트
- **btmPosition**: 포지션에 대한 업데이트
- **btmPrivateNotifications**: 개별 알림 - 현재 사용되지 않음
- **btmTransact**: 입금/출금 업데이트
- **btmWallet**: 총 입금 및 출금을 포함한 Bitcoin 주소 잔액 데이터

수신된 메시지의 예:

```
{
  "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"},
  ]
}
```

Authentication

user-locked 스트림을 구독하려면 먼저 인증해야 합니다. 잘못된 인증은 연결을 닫습니다.

BitMEX API를 사용하려면 API Key가 필요합니다.

영구 API Key는 메인 자격 증명을 손상시키지 않고도 IP 주소 범위로 잠그고 원할 때 폐기할 수 있습니다. 또한 갱신이 필요하지 않습니다.

API Key 인증을 사용하려면 계정에서 API Key를 생성해야 합니다.

인증된 토픽을 구독하기 전에 **Authenticate** 메서드를 호출하십시오.

CancelAllAfter (Dead Man's Switch)

CancelAllAfter 메서드는 Dead Man's Switch 기능을 구현합니다. 타임아웃 값(밀리초)으로 호출되면, 타임아웃 기간 내에 후속 CancelAllAfter 호출이 수신되지 않으면 서버에게 모든 미결제 주문을 취소하도록 지시합니다. 이는 네트워크 연결 끊김의 경우 주문이 취소되도록 보장하는 데 유용합니다.

REST API

Method	설명
GetExecutions	이는 주문 개시 및 취소, 주문 상태 변경을 포함한 모든 raw 트랜잭션을 반환합니다.
GetExecutionsTradeHistory	이는 더 집중된 트랜잭션을 반환합니다.
GetInstruments	이는 결제되었거나 상장 폐지된 것을 포함하여 모든 상품과 인덱스를 반환합니다. 개별 상품을 쿼리하거나 복잡한 필터를 사용하려면 이 엔드포인트를 사용하십시오.
GetOrders	미결제 주문만 가져오기
PlaceOrder	TsgcHTTPBitmexOrder 객체를 사용하여 raw 주문을 합니다.
PlaceMarketOrder	새 MARKET 주문을 배치합니다.
PlaceLimitOrder	새 LIMIT 주문을 실행합니다.
PlaceStopOrder	새 STOP 주문을 넣습니다.
PlaceStopLimitOrder	새 STOPLIMIT 주문을 넣습니다.
AmendOrder	기존 주문을 수정합니다.
CancelOrder	활성 주문을 취소합니다.
CancelAllOrders	모든 활성 주문을 취소합니다.
CancelAllOrdersAfter	일정 시간 후 모든 주문 취소.
ClosePosition	미결제 포지션을 청산합니다.
GetOrderBook	세로 형식으로 현재 OrderBook 가져오기
GetPosition	포지션을 가져옵니다.
SetPositionIsolate	포지션별로 isolated margin 또는 cross-margin을 활성화합니다.
SetPositionLeverage	포지션당 leverage를 선택합니다.
SetPositionRiskLimit	위험 한도를 업데이트합니다.
SetPositionTransferMargin	포지션에서 자기자본을 이체하거나 인출합니다.
GetQuotes	Get Quotes
GetTrades	Get Trades
GetFunding	펀딩 데이터를 가져옵니다.
GetInsurance	보험 기금 데이터를 가져옵니다.
GetTradeBucketed	구성 가능한 빈 크기로 버킷화된 거래 데이터(OHLCV)를 가져옵니다.
GetQuoteBucketed	구성 가능한 빈 크기로 버킷화된 호가 데이터를 가져옵니다.
GetSettlement	정산 데이터를 가져옵니다.
GetLiquidation	청산 주문을 가져옵니다.
GetInstrumentIndices	instrument 인덱스를 가져옵니다.
GetInstrumentCompositeIndex	약기에 대한 복합 인덱스 데이터를 가져옵니다.
GetStats	거래소 전체 통계를 가져옵니다.

GetStatsHistory	과거 거래소 통계를 가져옵니다.
GetStatsHistoryUSD	과거 USD 환율 통계를 가져옵니다.
GetUserMargin	계정 마진 데이터를 가져옵니다.
GetUserWallet	지갑 정보를 가져옵니다.
GetUserWalletHistory	지갑 거래 내역을 가져옵니다.
GetUserWalletSummary	지갑 요약을 가져옵니다.

Bitmex | Connect WebSocket API

Bitmex WebSocket API에 연결하려면, 새 Bitmex API 클라이언트를 생성하고 TsgcWebSocketClient에 연결하기만 하면 됩니다.

아래 예제를 참조하십시오:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bitmex oBitmex = new TsgcWSAPI_Bitmex();
oBitmex->Client = oClient;
oClient->Active = true;
```


Bitmex | Subscribe WebSocket Channel

Bitmex는 시장 데이터, 주문 등의 실시간 업데이트를 받기 위해 구독할 수 있는 다양한 채널을 제공합니다. 아래는 Trade Channel을 구독하는 방법의 샘플입니다:

```
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 | 주문 넣는 방법

Bitmex REST API는 public 및 private 엔드포인트를 제공합니다. Private 엔드포인트는 거래의 보안을 강화하기 위해 메시지에 서명할 것을 요구합니다.

먼저 Bitmex 계정에 로그인하고 새 API를 생성해야 합니다. 다음 값을 얻게 됩니다:

- ApiKey
- ApiSecret

이 필드는 Bitmex API 클라이언트 구성 요소의 Bitmex 속성에서 구성해야 합니다. 구성되면 Bitmex REST API에 비공개 요청을 시작할 수 있습니다.

Order Types

모든 주문에는 심볼이 필요합니다. 다른 모든 필드는 별도로 명시되지 않는 한 선택 사항입니다.

유효한 ordTypes는 다음과 같습니다:

- **Limit:** 기본 주문 유형입니다. orderQty와 price를 지정하십시오.
- **Market:** 전통적인 Market 주문입니다. Market 주문은 체결되거나 bankruptcy 가격에 도달할 때까지 실행되며, 그 시점에서 취소됩니다.
- **Stop:** Stop Market 주문입니다. orderQty와 stopPx를 지정하십시오. stopPx에 도달하면 주문이 호가창에 등록됩니다.
 - 매도 주문의 경우, 트리거 가격이 stopPx보다 낮으면 주문이 트리거됩니다. 매수 주문의 경우 더 높을 때 트리거됩니다.
 - 참고: Stop 주문은 트리거될 때까지 마진을 소비하지 않습니다. 주문이 완전히 트리거될 수 있도록 계정에 필요한 마진이 충분한지 확인하십시오.
 - Close Stops는 orderQty가 필요하지 않습니다. 아래의 Execution Instructions를 참조하십시오.
- **StopLimit:** Stop Market와 같지만 Market 주문 대신 Limit 주문을 진입시킵니다. orderQty, stopPx 및 price를 지정하십시오.
- **MarketIfTouched:** Stop과 유사하지만 트리거가 반대 방향으로 수행됩니다. Take Profit 주문에 유용합니다.
- **LimitIfTouched:** 위와 동일합니다. Take Profit Limit 주문에 사용하십시오.
- **Pegged:** Pegged 주문을 사용하면 사용자가 현재 시장 가격에 상대적인 limit price를 제출할 수 있습니다. pegPriceType과 pegOffsetValue를 지정하십시오.
 - Pegged 주문은 execlnst가 Fixed여야 합니다. 이는 주문이 수락될 때 limit price가 설정되고 참조 가격이 변경되어도 변경되지 않음을 의미합니다.
 - PrimaryPeg: 가격이 near touch 가격에 상대적으로 설정됩니다.
 - MarketPeg: 가격이 far touch 가격에 상대적으로 설정됩니다.
 - ordType 없이 제출된 pegPriceType은 Pegged 주문으로 처리됩니다.

Execution Instructions

다음 execlnst가 지원됩니다. 여러 개를 사용하는 경우 쉼표로 구분하십시오(예: LastPrice,Close).

- **ParticipateDoNotInitiate:** Post-Only order라고도 합니다. 이 주문이 배치 시 실행될 경우 대신 취소됩니다. 이는 주문이 전송 중일 때 far touch가 사용자 쪽으로 이동하는 것으로부터 보호하기 위한 것입니다. 제출 후 far touch가 멀어지는 것에 대해 투기하기 위한 것이 아닙니다. 우리는 그러한 행동을 남용으로 간주하고 모니터링합니다.
- **MarkPrice, LastPrice, IndexPrice:** stop 및 if-touched 주문에서 트리거 가격을 결정하는 데 사용됩니다. 하나만 사용하십시오. 기본적으로 MarkPrice가 사용됩니다. Pegged 주문에서 LastPeg의 값을 정의하는 데도 사용됩니다.
- **ReduceOnly:** ReduceOnly 주문은 포지션을 늘리지 않고 줄일 수만 있습니다. 포지션이 다른 주문에 의해 줄어드는 동안 호가창에 남아 있는 ReduceOnly limit 주문이 있으면, 그 주문 수량은 하향 수정되거나 취소됩니다. 여러 ReduceOnly 주문이 있으면 가장 덜 공격적인 것이 먼저 수정됩니다.
- **Close:** Close는 ReduceOnly를 의미합니다. Close 주문은 미체결 수량이 현재 포지션을 초과하면 동일한 방향과 심볼을 가진 다른 활성 지정가 주문을 취소합니다. 이는 stop에 유용합니다. 이러한 주문을 취소함으로써 Close Stop이 실행에 필요한 증거금을 갖도록 보장되며, 포지션의 전체 크기까지만 실행될 수 있습니다. orderQty가 지정되지 않으면 Close 주문은 현재 포지션 크기와 동일한 orderQty를 갖습니다.
 - orderQty가 없는 Close 주문에는 side가 필요하므로 BitMEX가 stopPx 위 또는 아래에서 트리거해야 하는지 알 수 있습니다.
- **LastWithinMark:** LastPrice를 사용하는 스탱 주문에서 다음의 경우에만 스탱 트리거를 허용하는 데 사용됩니다:

- Sell Stop Market / Stop Limit Order의 경우
 - Last Price $\leq$ Stop Price
 - Last Price $\geq$ Mark Price $\times (1 - 5\%)$
- Buy Stop Market / Stop Limit Order의 경우:
 - Last Price $\geq$ Stop Price
 - Last Price $\leq$ Mark Price $\times (1 + 5\%)$
- **Fixed:** Pegged 주문은 execlnst가 Fixed여야 합니다. 이는 주문이 수락되는 시점에 limit 가격이 설정되며 참조 가격이 변경되어도 변하지 않음을 의미합니다.

Pegged Orders

Pegged 주문은 사용자가 현재 시장 가격에 상대적인 limit 가격을 제출할 수 있게 합니다. limit 가격은 주문이 제출될 때 한 번 설정되며 참조 가격에 따라 변하지 않습니다. 이 주문 유형은 제출 후 far touch가 멀어지는 것에 대한 투기를 위한 것이 아닙니다. 우리는 그러한 행동을 남용으로 간주하고 모니터링합니다.

Pegged 주문은 ordType이 Pegged이고 execlnst가 Fixed입니다.

pegPriceType과 pegOffsetValue도 제출해야 합니다.

- PrimaryPeg - 가격이 니어 터치(near touch) 가격을 기준으로 설정됩니다.
- MarketPeg - 가격이 far touch 가격을 기준으로 설정됩니다

Trailing Stop Pegged Orders

Trailing Stop을 생성하려면 TrailingStopPeg의 pegPriceType을 사용하십시오.

가격은 제출 시 설정되며 기초 가격(last/mark/index)이 0.1% 이상 이동하면 초당 한 번 업데이트됩니다. 그러면 stopPx는 시장이 peg에서 멀어질 때 함께 이동하고 시장이 peg를 향해 이동할 때 고정됩니다.

주문의 stopPx를 설정하려면 pegOffsetValue를 사용하십시오. peg는 execlnst에 지정된 트리거 가격(기본값 MarkPrice)으로 설정됩니다. stop-sell 및 buy-if-touched 주문에는 음수 오프셋을 사용하십시오.

ordType: Stop, StopLimit, MarketIfTouched, LimitIfTouched가 필요합니다.

Trailing Stops

Trailing Stops를 생성하려면 'TrailingStopPeg'의 pegPriceType을 사용할 수 있습니다. peg된 stopPx는 시장이 peg에서 멀어질수록 이동하고 시장이 peg를 향해 이동하면 고정됩니다.

사용하려면 pegOffsetValue와 결합하여 주문의 stopPx를 설정하십시오. peg는 execlnst에 지정된 트리거 가격으로 설정됩니다(기본값 'MarkPrice'). stop-sell 및 buy-if-touched 주문에는 음수 오프셋을 사용하십시오.

ordType이 필요합니다: 'Stop', 'StopLimit', 'MarketIfTouched', 'LimitIfTouched'.

주문 추적

주문 ID를 직접 추적하려면 주문당 고유한 clOrdID를 설정하십시오. 이 clOrdID는 주문과 관련 실행(WebSocket 포함)에 속성으로 반환되며, 주문을 가져오거나 취소하는 데 사용할 수 있습니다. 최대 길이는 36자입니다.

예제:

```
// 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

Bitfinex은 세계에서 가장 크고 진보된 암호화폐 거래 플랫폼 중 하나입니다. 사용자는 Bitcoin, Ethereum, Ripple, EOS, Bitcoin Cash, Iota, NEO, Litecoin, Ethereum Classic 등을 교환할 수 있습니다...

Bitfinex WebSocket API 버전은 2.0입니다

Bitfinex의 WebSocket 채널을 통해 송수신되는 각 메시지는 JSON 형식으로 인코딩됩니다

심볼은 거래 쌍 또는 마진 통화일 수 있습니다:

- 거래 쌍 심볼은 쌍 앞에 "t"를 붙여서 형성됩니다(즉, tBTCUSD, tETHUSD).
- 마진 통화 심볼은 통화 앞에 "f"를 붙여서 형성됩니다(예: fUSD, fBTC, ...)

성공적인 연결 후 **OnBitfinexConnect** 이벤트가 발생하고 Bitfinex API 버전 번호를 매개변수로 받습니다.

서버에 대한 연결을 테스트하려면 **Ping** 메서드를 호출할 수 있습니다.

서버가 정보를 보내면 **OnBitfinexInfoMessage** 이벤트를 사용하여 처리할 수 있으며, 여기서 Code와 Message는 서버가 보낸 메시지에 대한 정보를 담은 매개변수입니다. 예제 코드:

```
20051 : WebSocket Server 중지/재시작(다시 연결하십시오)
20060 : 유지 관리 모드로 진입 중입니다. 모든 활동을 일시 중지하고 정보 메시지 20061을 받은 후 재개하십시오(최대 120초 정도 소요됩니다).
20061 : 유지 관리가 종료되었습니다. 정상적인 활동을 재개할 수 있습니다. 모든 채널을 다시 구독 해제/구독하는 것이 좋습니다.
```

오류가 발생하면 **OnBitfinexError**가 발생하고 오류에 대한 정보가 제공됩니다. 예제 오류 코드:

```
10000 : 알 수 없는 이벤트
10001 : 알 수 없는 페어
```

구성을 변경하려면 **Configuration** 메서드를 호출하고 다음 플래그 중 하나를 매개변수로 전달하십시오:

```
CS_DEC_S = 8; // Enable all decimal as strings.
CS_TIME_S = 32; // 모든 시간을 date 문자열로 활성화합니다.
CS_SEQ_ALL = 65536; // sequencing 활성화 BETA FEATURE
CHECKSUM = 131072; // 모든 호가창 반복에 대해 체크섬을 활성화합니다. 호가창의 각 측면에서 상위 25개 항목을 확인합니다. 체크섬은 부호 있는 정수입니다.
```

Subscribe Public Channels

public이고 서버에 대해 인증할 필요가 없는 채널이 있습니다. 모든 메시지는 **OnBitfinexUpdate** 이벤트로 발생합니다.

Subscribe Ticker

ticker는 시장 상태에 대한 고수준 개요입니다. 현재 최적의 bid와 ask, 그리고 마지막 거래 가격을 보여줍니다. 또한 일일 거래량과 지난 하루 동안 가격이 얼마나 움직였는지와 같은 정보도 포함합니다.

```
// 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

이 채널은 Bitfinex에서 거래가 발생할 때마다 거래 메시지를 보냅니다. 가격, 크기, 시간 등 거래의 모든 관련 세부 정보를 포함합니다.

```

// on trading pairs (ex. tBTCUSD)
[
    CHANNEL_ID,
    [
        [
            ID,
            MTS,
            AMOUNT,
            PRICE
        ],
        ...
    ]
]
// on funding currencies (ex. fUSD)
[
    CHANNEL_ID,
    [
        [
            ID,
            MTS,
            AMOUNT,
            RATE,
            PERIOD
        ],
        ...
    ]
]

```

SubscribeOrderBook

Order Books 채널을 사용하면 Bitfinex 호가창의 상태를 추적할 수 있습니다. 사용자 정의 가능한 정밀도로 가격 집계 기준으로 제공됩니다. 응답을 받은 후 호가창의 스냅샷을 받고, 호가창의 변경 사항에 따른 업데이트가 이어집니다.

```
// on trading pairs (ex. tBTCUSD)
[
  CHANNEL_ID,
  [
    [
      PRICE,
      COUNT,
      AMOUNT
    ],
    ...
  ]
]

// on funding currencies (ex. fUSD)
[
  CHANNEL_ID,
  [
    [
      RATE,
      PERIOD,
      COUNT,
      AMOUNT
    ],
    ...
  ]
]
```

SubscribeRawOrderBook

이것들은 가장 세분화된 호가창입니다.

```
// on trading pairs (ex. tBTCUSD)
[
  CHANNEL_ID,
  [
    [
      ORDER_ID,
      PRICE,
      AMOUNT
    ],
    ...
  ]
]

// on funding currencies (ex. fUSD)
[
  CHANNEL_ID,
  [
    [
      OFFER_ID,
      PERIOD,
      RATE,
      AMOUNT
    ],
    ...
  ]
]
```

SubscribeCandles

차트 캔들 정보에 액세스하는 방법을 제공합니다. 시간 프레임:

- 1m: 1분
- 5m : 5분
- 15m : 15분
- 30m : 30분
- 1h : 1시간
- 3h : 3시간
- 6h : 6시간
- 12h : 12시간

1D : 하루
 7D : 일주일
 14D : 2주
 1M : 1개월

```
[
  CHANNEL_ID,
  [
    [
      MTS,
      OPEN,
      CLOSE,
      HIGH,
      LOW,
      VOLUME
    ],
    ...
  ]
]
```

Subscribe Authenticated Channels

이 채널을 사용하면 계정 상태를 최신으로 유지할 수 있습니다. 포지션, 잔액, 주문 및 거래에 대한 업데이트를 받을 수 있습니다.

서버에 대해 인증하고 필요한 매개변수를 설정하려면 **Authenticate** 메서드를 사용하십시오.

인증되면 다음 항목의 업데이트를 수신하게 됩니다: 주문, 포지션, 거래, 펀딩 오퍼, 펀딩 크레딧, 펀딩 론, 지갑, 잔액 정보, 마진 정보, 펀딩 정보, 펀딩 거래...

서버에서 로그오프하려면 **UnAuthenticate** 메서드를 요청할 수 있습니다.

API Kucoin

Kucoin

Kucoin은 국제적인 다국어 암호화폐 거래소입니다. Kucoin 데이터에 접근하기 위한 일부 API를 제공합니다. 다음 API가 지원됩니다.

1. **WebSocket streams:** 일부 메서드를 구독하여 데이터를 실시간으로 받을 수 있게 합니다. 이벤트는 서버에 의해 구독자에게 푸시됩니다. 프로토콜로 WebSocket을 사용합니다.
2. **REST API:** 클라이언트는 서버에 시장 및 계정 데이터를 요청할 수 있습니다. 인증을 위해 API Key, Secret 및 Passphrase가 필요하며 프로토콜로 HTTPs를 사용합니다.

Properties

Kucoin API에는 공개와 비공개 2가지 유형의 메서드가 있습니다. 공개 메서드는 인증 없이 액세스할 수 있습니다(예: 티커 가격 가져오기). 사용자 데이터와 관련된 비공개 메서드는 Kucoin API 키 사용이 필요합니다.

- **ApiKey:** kucoin 계정에서 새 api key를 요청할 수 있으며, 값을 이 속성에 복사하기만 하면 됩니다.
- **ApiSecret:** API secret은 REST API에만 필요하며, websocket api는 일부 메서드에 대해 ApiKey만 필요합니다.
- **Passphrase:** Kucoin 서버에 연결하는 데 필요한 문자열입니다.
- **Sandbox:** 활성화되면 Kucoin Demo Account에 연결됩니다(기본적으로 false).
 - **HTTPLogOptions:** HTTP 요청 로그를 텍스트 파일에 저장합니다
 - **Enabled:** 활성화되면 WebSocket API의 모든 HTTP 요청을 저장합니다.
 - **FileName:** 로그가 저장될 파일 이름의 전체 경로
 - **REST:** REST API 요청 로그를 텍스트 파일에 저장합니다
 - **Enabled:** 활성화되면 REST API의 모든 HTTP 요청을 저장합니다.
 - **FileName:** 로그가 저장될 파일 이름의 전체 경로.

Most common uses

- **WebSockets API**
 - [WebSocket API 연결 방법](#)
 - [WebSocket Channel 구독 방법](#)
- **REST API**
 - [시장 데이터를 가져오는 방법](#)
 - [Private REST API 사용 방법](#)
 - [현물 거래 방법](#)
 - [Private Requests Time](#)

WebSocket 피드

특정 서버에서 채널 메시지를 구독하려면 클라이언트 측이 서버에 구독 메시지를 보내야 합니다. 구독이 성공하면, 응답이 true로 설정될 때 시스템이 ack 메시지를 보냅니다.

```
{
  "id": "1545910660739",
  "type": "ack"
}
```

토픽 메시지가 생성되는 동안 시스템은 해당 메시지를 클라이언트 측으로 보냅니다.

다음 Subscription / Unsubscription 메서드가 지원됩니다.

Public Channels

Method	매개변수	설명
SubscribeSymbolTicker	Symbol	이 topic을 구독하여 BBO 변경 사항의 푸시를 받으십시오. 1초 내에 변경 사항이 없으면 푸시되지 않습니다. 가장 최신 BBO와 함께 100ms마다 푸시됩니다. 마지막 데이터와 비교하여 변경 사항이 없으면 푸시되지 않습니다.
SubscribeAllSymbolsTicker		모든 시장 심볼 BBO 변경의 푸시를 받으려면 이 토픽을 구독하십시오.
SubscribeSymbolSnapshot	Symbol	단일 symbol에 대한 snapshot 데이터를 가져오려면 구독하십시오. snapshot 데이터는 2초 간격으로 푸시됩니다.
SubscribeMarketSnapshot	Market	전체 시장의 snapshot 데이터를 가져오려면 이 topic을 구독하십시오. snapshot 데이터는 2초 간격으로 푸시됩니다.
SubscribeLevel2MarketData	Symbol	Level2 주문서 데이터를 가져오려면 이 토픽을 구독하십시오. websocket 구독이 성공하면 시스템이 websocket으로 푸시한 증분 변경 데이터를 보냅니다.
SubscribeLevel2_5BestAskBid	Symbol	시스템은 100밀리초마다의 스냅샷 데이터인 최고 5개의 매도/매수 주문 데이터를 반환합니다(즉, 최고 5개의 매도/매수 주문 데이터가 100밀리초마다 실시간으로 반환됨).
SubscribeLevel2_50BestAskBid	Symbol	시스템은 100밀리초마다의 스냅샷 데이터인 최고 50개의 매도/매수 주문 데이터를 반환합니다(즉, 최고 50개의 매도/매수 주문 데이터가 100밀리초마다 실시간으로 반환됨).
SubscribeKlines	Symbol	K-Line 데이터를 얻으려면 이 토픽을 구독하십시오.
SubscribeMatchExecutionData	Symbol	이 토픽을 구독하여 Level 3의 일치 이벤트 데이터 흐름을 가져옵니다. 거래된 각 주문에 대해 시스템은 다음 형식으로 일치 메시지를 보냅니다.
SubscribeIndexPrice	Symbol	마진 거래에 대한 index price를 가져오려면 이 topic을 구독하십시오.
SubscribeMarkPrice	Symbol	margin 거래의 mark price를 얻으려면 이 토픽을 구독하십시오.
SubscribeOrderBookChanged	Symbol	margin 거래의 주문 장부 변경을 얻으려면 이 토픽을 구독하십시오.
SubscribeLevel1	Symbol	심볼에 대한 Level 1 최우선 매수/매도 데이터를 구독합니다.

ACK 매개변수가 true로 전송되면 성공적인 구독 / 구독 취소 후 클라이언트는 이에 대한 메시지를 수신합니다.

Private 채널

Kucoin 계정에서 얻은 유효한 ApiKey가 필요합니다. ApiKey, ApiSecret 및 Passphrase는 클라이언트 API 구성 요소의 Kucoin 속성에 설정해야 합니다.

다음 데이터는 변경이 있을 때마다 클라이언트에 푸시됩니다. 어떤 메서드도 구독할 필요가 없으며, 유효한 ApiKey를 설정하면 자동으로 수행됩니다.

Method	설명
SubscribeTradeOrders	이 토픽은 주문의 모든 변경 이벤트를 푸시합니다.
SubscribeAccountBalance	계정 잔액이 변경되면 이 메시지를 받게 됩니다. 메시지에는 변경 세부 정보가 포함됩니다.
SubscribePositionStatus	포지션 상태가 변경되면 시스템이 변경 이벤트를 푸시합니다.
SubscribeMarginTradeOrders	주문이 호가창에 들어가면 시스템이 이 메시지를 대출자에게 푸시합니다.
SubscribeStopOrder	stop order가 시스템에 수신되면 "open" 유형의 메시지를 받게 됩니다. 이는 이 주문이 시스템에 진입하여 트리거되기를 기다리고 있음을 의미합니다.

SubscribeTradeOrdersV2	향상된 주문 업데이트 알림을 위해 trade orders V2 채널을 구독합니다.
SubscribeCrossMarginPosition	교차 마진 포지션 업데이트를 구독합니다. 교차 마진 포지션이 변경되면 시스템이 변경 이벤트를 푸시합니다.
SubscribeIsolatedMarginPosition	isolated margin 포지션 업데이트를 구독합니다. isolated margin 포지션이 변경되면 시스템이 변경 이벤트를 푸시합니다.

REST API

모든 엔드포인트는 JSON 객체 또는 배열을 반환합니다.

Public API EndPoints

이러한 엔드포인트는 권한 부여 없이 접근할 수 있습니다.

General EndPoints

Method	매개변수	설명
GetServiceStatus		Rest API에 대한 연결성을 테스트하고 Service Status를 가져옵니다.
GetServerTime		Rest API에 대한 연결성을 테스트하고 현재 서버 시간을 가져옵니다.

Market Data EndPoints

Method	매개변수	설명
GetSymbolList	Market	이 엔드포인트를 통해 요청하여 거래 가능한 통화 쌍 목록을 가져옵니다. 거래 심볼의 마켓 정보를 가져오려는 경우
GetTicker	Symbol	이 엔드포인트를 통해 요청하여 Level 1 Market Data를 가져옵니다. 반환된 값에는 최적 매수 호가와 크기, 최적 매도 호가와 크기, 그리고 마지막 체결 가격과 마지막 체결 크기가 포함됩니다.
GetAllTickers		시장의 모든 거래 쌍에 대한 시장 티커를 요청합니다(24시간 거래량 포함).
Get24hrStats	Symbol	지난 24시간 동안 지정된 티커의 통계를 가져오려면 이 엔드포인트를 통해 요청하십시오.
GetMarketList		전체 거래 시장의 거래 통화를 가져오려면 이 엔드포인트를 통해 요청하십시오.
GetPartOrderBook20	Symbol	symbol에 대한 미체결 주문 목록을 가져오려면 이 엔드포인트를 통해 요청하십시오. Level-2 order book에는 모든 bid와 ask(가격별 집계)가 포함되며, 이 level은 각 활성 가격에 대해 하나의 크기만 반환합니다(해당 가격에 단일 주문만 있는 것처럼). 시스템은 order book에서 20개의 데이터(ask 및 bid 데이터)를 반환합니다.
GetPartOrderBook100	Symbol	이 엔드포인트를 통해 요청하여 심볼에 대한 미체결 주문 목록을 가져옵니다. Level-2 호가창에는 모든 매수 및 매도 호가(가격별로 집계됨)가 포함되며, 이 레벨은 각 활성 가격에 대해 하나의 크기만 반환합니다(해당 가격에 단일 주문만 있는 것처럼).

		럼). 시스템은 호가창에서 100개의 데이터(매도 및 매수 데이터)를 반환합니다.
GetFullOrderBook	Symbol	지정된 심볼의 order book을 얻으려면 이 엔드포인트를 통해 요청하십시오. Level 2 order book은 모든 bids와 asks를 포함합니다(가격별로 집계됨). 이 level은 각 가격에 대해 하나의 집계 크기만 반환합니다(마치 그 가격에 단일 주문 하나만 있는 것처럼). 이 API는 전체 depth의 데이터를 반환합니다.
GetKLines	Symbol	이 엔드포인트를 통해 요청하여 지정된 symbol의 kline을 가져옵니다. 데이터는 요청된 유형에 따라 그룹화된 버킷으로 반환됩니다.
GetCurrencies		통화 목록을 가져오려면 이 엔드포인트를 통해 요청하십시오.
GetCurrencyDetail	Currency	지정된 통화의 통화 세부 정보를 얻으려면 이 엔드포인트를 통해 요청하십시오.
GetFiatPrice		지정된 통화의 통화 세부 정보를 얻으려면 이 엔드포인트를 통해 요청하십시오.
GetPartOrderBook1	Symbol	이 엔드포인트를 통해 요청하여 심볼에 대한 Level 1 최적 매수/매도 호가를 가져옵니다.

Private API EndPoints

서버의 권한을 받으려면 APIKey와 APISecret이 필요합니다.

User EndPoints

Method	매개변수	설명
GetAllSubAccounts		이 인터페이스를 통해 모든 서브 사용자의 사용자 정보를 가져올 수 있습니다.
GetListAccounts		계정 목록을 가져옵니다.
GetAccount	AccountId	단일 계정에 대한 정보입니다. accountId를 알고 있을 때 이 엔드포인트를 사용하십시오.
GetAccountBalanceSubAccount	SubUserId	이 엔드포인트는 subUserId로 지정된 하위 사용자의 계정 정보를 반환합니다.
InnerTransfer		이 API 엔드포인트는 계정 간 자금을 내부적으로 이체하는 데 사용할 수 있습니다. 사용자는 메인 계정, 트레이딩 계정, 교차 마진 계정, 격리 마진 계정 간에 무료로 자금을 이체할 수 있습니다. 메인 계정, 교차 마진 계정, 트레이딩 계정에서 선물 계정으로의 자금 이체는 지원되지만, 선물 계정에서 다른 계정으로의 자금 이체는 지원되지 않습니다.
GetDepositAddresses	Currency	통화의 입금 주소를 가져옵니다.
CreateDepositAddress	Currency	통화에 대한 새 입금 주소를 생성합니다.
GetDepositList		입금 내역을 가져옵니다.
GetAccountLedgers		계정 원장 항목을 가져옵니다.
GetTradeFees	Symbols	지정된 심볼의 거래 수수료를 가져옵니다.

Withdraw EndPoints

Method	매개변수	설명
GetWithdrawalsList		Withdrawal 목록을 가져옵니다.
GetHistoricalWithdrawalsList		KuCoin V1 과거 출금 목록입니다.
GetWithdrawalsQuotas	Currency	Get Withdrawals Quotas
ApplyWithdraw	Currency, Address, Amount	Create a Withdraw
CancelWithdraw	WithdrawalId	PROCESSING 상태의 출금 요청만 취소할 수 있습니다.

Trade Endpoints

Method	매개변수	설명
PlaceOrder		두 가지 유형의 주문을 실행할 수 있습니다: limit 및 market. 주문은 계정에 충분한 자금이 있는 경우에만 실행할 수 있습니다. 주문이 실행되면 주문 기간 동안 계정 자금이 보류됩니다. 보류되는 자금의 양과 종류는 주문 유형 및 지정된 매개변수에 따라 다릅니다.
PlaceMarketOrder		시장가 주문을 넣습니다.
PlaceLimitOrder		Limit Order를 합니다.
PlaceMarginOrder		Margin Order를 발주합니다.
CancelOrder		주문 Id로 주문을 취소합니다.
CancelOrderByClientOid		Client Order Id로 주문을 취소합니다.
CancelAllOrders		모든 미체결 주문을 취소합니다.
ListOrders		이 엔드포인트를 통해 요청하여 현재 주문 목록을 가져옵니다. 항목은 페이지로 나뉘고 최신순으로 정렬되어 표시됩니다.
GetRecentOrders		이 엔드포인트를 통해 요청하여 지난 24시간 동안의 1000개 주문을 가져옵니다.
GetOrder		order ID로 단일 주문 정보를 가져오려면 이 엔드포인트를 통해 요청하십시오.
GetOrderByClientOid		Client order ID로 단일 주문 정보를 가져오려면 이 엔드포인트를 통해 요청하십시오.
ListFills		최근 체결 내역을 가져오려면 이 엔드포인트를 통해 요청하십시오.
GetRecentFills		최근 24시간 동안의 1000개의 fill 목록을 가져오려면 이 엔드포인트를 통해 요청하십시오.
PlaceStopOrder		Stop Order를 실행합니다.
PlaceStopMarketOrder		Stop Market Order를 합니다.
PlaceStopLimitOrder		Stop Limit Order를 배치합니다.
CancelStopOrder		Order Id로 미결제 Stop Order를 취소합니다
CancelStopOrderByClientOid		Client Order Id로 미결제 Stop Order를 취소합니다
CancelAllStopOrders		Cancel All Stop Orders
GetStopOrder		이 인터페이스를 통해 요청하여 주문 ID로 스탑 주문 정보를 가져옵니다.
GetStopOrderByClientOid		Client order ID를 통해 stop order 정보를 가져오려면 이 인터페이스를 통해 요청하십시오.
ListStopOrders		이 엔드포인트를 통해 요청하여 현재 트리거되지 않은 스톱 주문 목록을 가져옵니다. 항목은 페이지로 나뉘며 최신 항목이 먼저 표시되도록 정렬됩니다.
PlaceHFOOrder		고빈도 주문을 실행합니다.

CancelHFOrder		order ID로 고빈도 주문을 취소합니다.
CancelHFOrderByClientOid		클라이언트 주문 ID로 고빈도 주문을 취소합니다.
CancelAllHFOrders		모든 고빈도 주문을 취소합니다.
GetHFActiveOrders		활성 고빈도 주문을 가져옵니다.
GetHFDoneOrders		완료된 고빈도 주문을 가져옵니다.
GetHFOrder		order ID로 특정 고빈도 주문을 가져옵니다.

Events

Kucoin 메시지는 TsgcWebSocketClient 구성 요소에서 수신되며, 다음 이벤트를 사용할 수 있습니다:

OnConnect

Kucoin 서버에 성공적으로 연결된 후입니다.

OnDisconnect

Kucoin 서버에서 연결이 끊긴 후

OnMessage

서버가 클라이언트에 보낸 메시지는 이 이벤트에서 처리됩니다.

OnError

프로토콜에 오류가 있으면 이 이벤트가 호출됩니다.

OnException

처리되지 않은 예외가 있으면 이 이벤트가 호출됩니다.

또한 Kucoin API 구성 요소에는 **OnKucoinHTTPException**이라는 특정 이벤트가 있으며, HTTP 요청(REST API 또는 WebSocket Feeds) 호출 중 오류가 있을 때마다 발생합니다.

Kucoin | Connect WebSocket API

Kucoin WebSocket API에 연결하려면 새 Kucoin API 클라이언트를 만들고 TsgcWebSocketClient에 연결하기만 하면 됩니다.

아래 예제를 참조하십시오:

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_Kucoin *oKucoin = new TsgcWSAPI_Kucoin();
oKucoin->Client = oClient;
oClient->Active = true;
```

Kucoin | Subscribe WebSocket Channel

Kucoin은 시장 데이터, 주문 등의 실시간 업데이트를 받기 위해 구독할 수 있는 다양한 채널을 제공합니다... Ticker를 구독하는 방법의 샘플을 아래에서 확인하십시오:

```
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

Kucoin은 REST 엔드포인트를 통해 공개 Market Data를 제공합니다. 이러한 엔드포인트 중 하나를 호출하면 요청된 시장 데이터의 스냅샷을 얻습니다.

Market Data 엔드포인트는 인증이 필요하지 않으므로 모든 사용자가 자유롭게 사용할 수 있습니다.

예: ticker BTC-USDT의 스냅샷을 얻으려면 다음 호출을 수행하십시오

```
TsgcWSAPI_Kucoin *oKucoin = new TsgcWSAPI_Kucoin(this);
ShowMessage(oKucoin->REST_API->GetTicker("
BTC-USDT
"));
```


Kucoin | Private REST API

Kucoin REST API는 public 및 private 엔드포인트를 제공합니다. Private 엔드포인트는 트랜잭션의 보안을 높이기 위해 메시지에 서명할 것을 요구합니다.

먼저 Kucoin 계정에 로그인하여 새 API를 생성해야 하며, 다음 값을 얻게 됩니다:

- ApiKey
- ApiSecret
- Passphrase

이 필드들은 Kucoin API 클라이언트 구성 요소의 Kucoin 속성에서 구성해야 합니다.

구성이 완료되면 Kucoin Pro REST API에 비공개 요청을 시작할 수 있습니다

*Private 요청은 로컬 머신의 로컬 시간이 동기화되어 있어야 합니다. 그렇지 않으면 Kucoin 서버에서 요청이 거부됩니다. 이에 관한 다음 문서를 확인하십시오. [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

Kucoin은 REST API를 사용하여 현물 거래를 할 수 있게 합니다.

구성

먼저 Kucoin 계정에서 **API Key**를 생성하고 Spot 거래 권한을 추가해야 합니다.

이 작업이 완료되면 현물 거래를 시작할 수 있습니다.

먼저, **Kucoin Client 구성 요소**에서 **ApiKey**, **ApiSecret** 및 **Passphrase**를 설정하십시오. 이는 Kucoin 서버로 전송되는 요청에 서명하는 데 사용됩니다.

Place an Order

새 주문을 배치하려면 Kucoin Client 구성 요소의 **REST_API.PlaceOrder** 메서드를 호출하면 됩니다.

주문 유형(market, limit...)에 따라 API는 더 많거나 적은 필드를 요구합니다.

Parameters

Param	type	설명
clientOid	String	사용자가 자신의 주문을 식별하기 위해 생성한 고유 주문 id, 예: UUID.
side	String	buy 또는 sell
symbol	String	유효한 거래 심볼 코드. 예: ETH-BTC
type	String	[선택] limit 또는 market (기본값은 limit)
remark	String	[선택적] 주문에 대한 비고이며, 길이는 100 utf8 문자를 초과할 수 없습니다
stp	String	[선택 사항] self trade prevention , CN , CO , CB 또는 DC
tradeType	String	[선택 사항] 거래 유형: TRADE (Spot Trade) , MARGIN_TRADE (Margin Trade). 기본값은 TRADE 입니다. 참고: 시스템 성능을 개선하고 주문 배치 및 처리를 가속화하기 위해, KuCoin은 margin 주문 배치를 위한 새 인터페이스를 추가했습니다. 현재 인터페이스를 계속 사용하는 트레이더는 가능한 한 빨리 새 인터페이스로 이동하십시오. 현재 인터페이스는 2021년 5월 1일(UTC)부터 더 이상 margin 주문을 수락하지 않습니다. 그때 KuCoin은 공지를 통해 사용자에게 알릴 것이니, 주의를 기울이십시오.

LIMIT ORDER PARAMETERS

Param	type	설명
price	String	기준 통화당 가격
size	String	매수 또는 매도할 기축 통화의 수량입니다
timeInForce	String	[선택적] GTC , GTT , IOC , 또는 FOK (기본값은 GTC), Time In Force 를 읽으십시오.
cancelAfter	long	[Optional] n 초 후 취소, timeInForce 가 GTT 여야 합니다
postOnly	boolean	[선택적] Post only 플래그이며, timeInForce 가 IOC 또는 FOK 일 때 유효하지 않습니다
hidden	boolean	[선택 사항] 주문이 order book에 표시되지 않습니다
iceberg	boolean	[선택] 주문의 일부만 오더북에 표시됩니다

Param	type	설명
visibleSize	String	[선택적] iceberg 주문의 최대 가시 크기

MARKET ORDER PARAMETERS

Param	type	설명
size	String	[선택적] 기준 통화로 표시된 원하는 금액
funds	String	[선택적] 사용할 quote currency의 원하는 양입니다

주문을 보낼 때 두 가지 가능성이 있습니다:

1. **성공:** PlaceOrder 함수는 Kucoin 서버가 보낸 메시지를 반환합니다.
2. **Error:** 예외는 OnKucointHTTPException 이벤트에서 반환됩니다.

시장가 주문 1 BTC-USDT 실행

```
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

));
```

40000에 1 BTC-USDT Limit Order 배치

```
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 | Private Requests Time

Kucoin에 비공개 요청을 할 때, 요청의 보안을 높이기 위해 메시지에 서명합니다. 메시지는 로컬 시간을 취하여 서명된 메시지 내부로 전송합니다. 로컬 시간이 Kucoin 서버와 5초보다 큰 차이가 있으면 요청이 거부됩니다. 따라서 로컬 시간이 동기화되어 있는지 확인하는 것이 중요하며, OS의 시간 동기화 메서드를 사용하여 이를 수행할 수 있습니다.

GetServerTime 메서드를 호출하여 Kucoin 서버 시간을 확인할 수 있으며, 이는 Kucoin 서버의 시간을 반환합니다.

API Kucoin Futures

Kucoin Futures

Kucoin은 국제적인 다국어 암호화폐 거래소입니다. Kucoin 데이터에 접근하기 위한 일부 API를 제공합니다. 다음 API가 지원됩니다.

1. **WebSocket streams:** 일부 메서드를 구독하여 데이터를 실시간으로 받을 수 있게 합니다. 이벤트는 서버에 의해 구독자에게 푸시됩니다. 프로토콜로 WebSocket을 사용합니다.
2. **REST API:** 클라이언트는 서버에 시장 및 계정 데이터를 요청할 수 있습니다. 인증을 위해 API Key, Secret 및 Passphrase가 필요하며 프로토콜로 HTTPs를 사용합니다.

Properties

Kucoin API에는 2가지 유형의 메서드가 있습니다: public 및 private. Public 메서드는 인증 없이 액세스할 수 있습니다(예: ticker 가격 가져오기). Private 및 사용자 데이터와 관련된 메서드는 Kucoin API 키 사용이 필요합니다.

- **ApiKey:** kucoin 계정에서 새 api key를 요청할 수 있으며, 값을 이 속성에 복사하기만 하면 됩니다.
- **ApiSecret:** API secret은 REST API에만 필요하며, websocket api는 일부 메서드에 대해 ApiKey만 필요합니다.
- **Passphrase:** Kucoin 서버에 연결하는 데 필요한 문자열입니다.
- **Sandbox:** 활성화되면 Kucoin Demo Account에 연결됩니다(기본적으로 false).
 - **HTTPLogOptions:** HTTP 요청 로그를 텍스트 파일에 저장합니다
 - **Enabled:** 활성화되면 WebSocket API의 모든 HTTP 요청을 저장합니다.
 - **FileName:** 로그가 저장될 파일 이름의 전체 경로
 - **REST:** REST API 요청 로그를 텍스트 파일에 저장합니다
 - **Enabled:** 활성화되면 REST API의 모든 HTTP 요청을 저장합니다.
 - **FileName:** 로그가 저장될 파일 이름의 전체 경로.

Most common uses

- **WebSockets API**
 - [WebSocket API에 연결하는 방법](#)
 - [WebSocket Channel을 구독하는 방법](#)
- **REST API**
 - [시장 데이터를 가져오는 방법](#)
 - [Private REST API 사용 방법](#)
 - [선물 거래 방법](#)
 - [Private Requests Time](#)

WebSocket 피드

특정 서버에서 채널 메시지를 구독하려면 클라이언트 측이 서버에 구독 메시지를 보내야 합니다. 구독이 성공하면, 응답이 true로 설정될 때 시스템이 ack 메시지를 보냅니다.

```
{
  "id": "1545910660739",
  "type": "ack"
}
```

토픽 메시지가 생성되는 동안 시스템은 해당 메시지를 클라이언트 측으로 보냅니다.

다음 Subscription / Unsubscription 메서드가 지원됩니다.

Public Channels

Method	매개변수	설명
SubscribeSymbolTickerV2	Symbol	BBO 변경의 실시간 푸시를 받으려면 이 토픽을 구독하십시오. 구독 후호가창에 변경이 있으면, 시스템이 실시간 ticker 심볼 정보를 푸시합니다. 적시 정보를 위해 새 토픽을 사용하는 것이 권장됩니다.
SubscribeSymbolTicker	Symbol	BBO 변경의 실시간 푸시를 받으려면 이 토픽을 구독하십시오. ticker 채널은 매칭이 발생할 때마다 실시간 가격 업데이트를 제공합니다. 여러 주문이 동시에 매칭되면 마지막 매칭 이벤트만 푸시됩니다.
SubscribeLevel2MarketData	Symbol	Level 2 order book 데이터를 가져오려면 이 topic을 구독하십시오.
SubscribeExecutionData	Symbol	실행된 각 주문에 대해, 시스템은 다음 형식으로 매칭 메시지를 보냅니다.
SubscribeLevel2_5BestAskBid	Symbol	최대 100밀리초마다 반환됩니다.
SubscribeLevel2_50BestAskBid	Symbol	최대 100밀리초마다 반환됩니다.
SubscribeContractMarketData	Symbol	계약의 시장 데이터를 얻으려면 이 토픽을 구독하십시오.
SubscribeSystemAnnouncements	Symbol	시스템 공지를 받으려면 이 토픽을 구독하십시오.
SubscribeTransactionStatistics	Symbol	거래 통계는 5초마다 사용자에게 푸시됩니다.
SubscribeKlines	Symbol	계약 klines(캔들스틱) 데이터를 구독합니다.
SubscribeFundingFeeSettlement	Symbol	펀딩 수수료 정산 알림을 구독합니다.

ACK 매개변수가 true로 전송되면 성공적인 구독 / 구독 취소 후 클라이언트는 이에 대한 메시지를 수신합니다.

Private 채널

Kucoin 계정에서 얻은 유효한 ApiKey가 필요합니다. ApiKey, ApiSecret 및 Passphrase는 클라이언트 API 구성 요소의 Kucoin 속성에 설정해야 합니다.

다음 데이터는 변경이 있을 때마다 클라이언트에 푸시됩니다. 어떤 메서드도 구독할 필요가 없으며, 유효한 ApiKey를 설정하면 자동으로 수행됩니다.

Method	설명
SubscribeTradeOrders	이 토픽은 주문의 모든 변경 이벤트를 푸시합니다.
SubscribeAccountBalance	계정 잔액이 변경되면 이 메시지를 받게 됩니다. 메시지에는 변경 세부 정보가 포함됩니다.
SubscribePositionChange	포지션 상태가 변경되면 시스템이 변경 이벤트를 푸시합니다.
SubscribeStopOrder	stop order가 시스템에 수신되면 "open" 유형의 메시지를 받게 됩니다. 이는 이 주문이 시스템에 진입하여 트리거되기를 기다리고 있음을 의미합니다.
SubscribeMarginMode	margin 모드 변경을 구독합니다. margin 모드가 업데이트되면 시스템이 변경 이벤트를 푸시합니다.
SubscribeCrossMarginLeverage	교차 마진 leverage 변경을 구독합니다. 교차 마진 leverage가 업데이트되면 시스템이 변경 이벤트를 푸시합니다.

REST API

모든 엔드포인트는 JSON 객체 또는 배열을 반환합니다.

Public API EndPoints

이러한 엔드포인트는 권한 부여 없이 접근할 수 있습니다.

General EndPoints

Method	매개변수	설명
GetServiceStatus		Rest API에 대한 연결성을 테스트하고 Service Status를 가져옵니다.
GetServerTime		Rest API에 대한 연결성을 테스트하고 현재 서버 시간을 가져옵니다.

Market Data EndPoints

Method	매개변수	설명
GetOpenContractList		모든 현재 열린 계약의 정보를 가져오기 위해 요청을 제출합니다.
GetOrderInfoContract		지정된 계약의 정보를 가져오기 위한 요청을 제출합니다.
GetTicker	Symbol	실시간 ticker에는 마지막 거래 가격, 마지막 거래 크기, 트랜잭션 ID, 유동성 taker의 방향, 최우선 매수 가격 및 크기, 최우선 매도 가격 및 크기뿐만 아니라 주문의 트랜잭션 시간이 포함됩니다. 이러한 메시지는 Websocket을 통해서도 얻을 수 있습니다. Sequence Number는 Websocket이 푸시한 메시지가 연속적인지 판단하는 데 사용됩니다.
GetPartOrderBook20	Symbol	심볼에 대한 집계된 미결 주문의 스냅샷을 가져옵니다.
GetPartOrderBook100	Symbol	심볼에 대한 집계된 미결 주문의 스냅샷을 가져옵니다.
GetFullOrderBook	Symbol	심볼에 대한 집계된 미결 주문의 스냅샷을 가져옵니다.
GetLevel2PullingMessages	Symbol	Websocket으로 푸시된 메시지가 연속적이지 않은 경우, 다음 요청을 제출하고 데이터를 다시 가져와 시퀀스가 누락되지 않도록 할 수 있습니다. 요청에서 start 매개변수는 마지막으로 받은 메시지의 시퀀스 번호에 1을 더한 값이고, end 매개변수는 현재 받은 메시지의 시퀀스 번호에서 1을 뺀 값입니다. 메시지를 다시 가져와 로컬 거래소 주문 호가창에 적용한 후 Websocket 증분 피드를 통해 호가창을 계속 업데이트할 수 있습니다. end와 start 매개변수의 차이가 500을 초과하면 이 요청 사용을 중지하고 Level 2 주문 호가창을 재구성하는 것이 좋습니다.
GetTradeHistory	Symbol	심볼에 대한 마지막 100건의 거래를 나열합니다.
GetInterestRateList	Symbol	이자율 목록을 확인합니다.
GetIndexList	Symbol	인덱스 목록 확인
GetCurrentMarkPrice	Symbol	현재 mark price를 확인합니다.
GetPremiumIndex	Symbol	premium index를 얻기 위한 요청을 제출합니다.
GetCurrentFundingRate	Symbol	현재 마크 가격을 확인하기 위한 요청을 제출합니다.
GetKLine	Symbol	Contract의 K Line 데이터 가져오기

Private API EndPoints

서버의 권한을 받으려면 APIKey와 APISecret이 필요합니다.

User EndPoints

Method	매개변수	설명
GetAccountOverview		Get Account Overview
GetTransactionHistory		미결제 포지션이 있는 경우, 반환되는 첫 페이지의 상태는 Pending이며, 현재 8시간 정산 기간의 실현 손익을 나타냅니다. 페이지를 넘기려면 현재 페이지의 최소 오프셋 번호를 offset 필드에 지정하십시오.

Trade Endpoints

Method	매개변수	설명
PlaceOrder		두 가지 유형의 주문을 넣을 수 있습니다: limit 및 market. 계정에 충분한 자금이 있는 경우에만 주문할 수 있습니다. 주문이 들어가면 자금이 주문 기간 동안 보류됩니다. 보류되는 자금의 양은 주문 유형 및 지정된 매개변수에 따라 다릅니다.
PlaceMarketOrder		시장가 주문을 넣습니다.
PlaceLimitOrder		Limit Order를 합니다.
CancelOrder		주문 Id로 주문을 취소합니다.
LimitOrderMassCancellation		모든 미결제 주문(stop 주문 제외)을 취소합니다. 응답은 취소된 주문의 orderID 목록입니다.
StopOrderMassCancellation		트리거되지 않은 모든 stop order를 취소합니다. 응답은 취소된 stop order의 orderID 목록입니다. 트리거된 stop order를 취소하려면 'Limit Order Mass Cancellation'을 사용하십시오.
GetOrderList		현재 주문을 나열합니다.
GetUntriggeredStopOrderList		트리거되지 않은 stop order 목록을 가져옵니다.
GetListOrdersCompleted24hr		최근 24시간 동안의 최근 1000개 주문 목록을 가져옵니다. 최근 거래된 주문 기록을 낮은 지연 시간으로 가져와야 하는 경우 이 엔드포인트를 쿼리할 수 있습니다.
GetOrder		주문 id로 단일 주문을 가져옵니다(stop 주문 포함).
GetOrderByClientOid		client order id로 단일 주문을 가져옵니다(stop order 포함).
GetFills		최근 fill 목록을 가져옵니다.
GetRecentFills		지난 24시간 내 최근 1000개의 체결 목록을 가져옵니다. 최근 거래된 주문 기록을 낮은 지연으로 가져와야 하는 경우 이 엔드포인트를 쿼리할 수 있습니다.
ActiveOrderValueCalculation		이 엔드포인트를 쿼리하여 모든 활성 주문의 총 개수와 가치를 가져올 수 있습니다.
GetPositionDetails		지정된 포지션의 포지션 세부 정보를 가져옵니다.
GetPositionList		지정된 포지션의 포지션 세부 정보를 가져옵니다.
AutoDepositMargin		Auto-Deposit Margin 활성화/비활성화
AddMarginManually		마진 수동 추가
ObtainFuturesRiskLimitLevel		이 인터페이스는 특정 계약의 위험 한도 레벨에 대한 정보를 얻는 데 사용할 수 있습니다

AdjustRiskLimitLevel		이 인터페이스는 리스크 한도 레벨 조정을 위한 것입니다. 레벨을 조정하면 미결제 주문이 취소되며, 응답은 조정 요청 제출의 성공 여부만 나타낼 수 있습니다.
GetFundingHistory		funding 기록을 가져오기 위한 요청을 제출합니다.
GetMaxOpenSize		계약에 대한 최대 미결제 포지션 크기를 가져옵니다.
SwitchMarginMode		cross margin과 isolated margin 모드 간에 전환합니다.
GetMarginMode		계약의 현재 마진 모드를 가져옵니다.

Events

Kucoin 메시지는 TsgcWebSocketClient 구성 요소에서 수신되며, 다음 이벤트를 사용할 수 있습니다:

OnConnect

Kucoin 서버에 성공적으로 연결된 후입니다.

OnDisconnect

Kucoin 서버에서 연결이 끊긴 후

OnMessage

서버가 클라이언트에 보낸 메시지는 이 이벤트에서 처리됩니다.

OnError

프로토콜에 오류가 있으면 이 이벤트가 호출됩니다.

OnException

처리되지 않은 예외가 있으면 이 이벤트가 호출됩니다.

또한 Kucoin API 구성 요소에는 **OnKucoinHTTPException**이라는 특정 이벤트가 있으며, HTTP 요청(REST API 또는 WebSocket Feeds) 호출 중 오류가 있을 때마다 발생합니다.

Kucoin | Futures Connect WebSocket API

Kucoin WebSocket API에 연결하려면 새 Kucoin API 클라이언트를 만들고 TsgcWebSocketClient에 연결하기만 하면 됩니다.

아래 예제를 참조하십시오:

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();  
TsgcWSAPI_Kucoin_Futures  
*oKucoin = new  
TsgcWSAPI_Kucoin_Futures  
();  
oKucoin->Client = oClient;  
oClient->Active = true;
```

Kucoin | Futures Subscribe WebSocket Channel

Kucoin은 시장 데이터, 주문 등의 실시간 업데이트를 받기 위해 구독할 수 있는 다양한 채널을 제공합니다... Ticker를 구독하는 방법의 샘플을 아래에서 확인하십시오:

```
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

Kucoin은 REST 엔드포인트를 통해 공개 Market Data를 제공합니다. 이러한 엔드포인트 중 하나를 호출하면 요청된 시장 데이터의 스냅샷을 얻습니다.

Market Data 엔드포인트는 인증이 필요하지 않으므로 모든 사용자가 자유롭게 사용할 수 있습니다.

예: ticker BTC-USDT의 스냅샷을 얻으려면 다음 호출을 수행하십시오

```
TsgcWSAPI_Kucoin_Futures
*oKucoin = new
TsgcWSAPI_Kucoin_Futures
(this);
ShowMessage(oKucoin->REST_API->GetTicker("
XBTUSDM
"));
```

Kucoin | Futures Private REST API

Kucoin REST API는 public 및 private 엔드포인트를 제공합니다. Private 엔드포인트는 트랜잭션의 보안을 높이기 위해 메시지에 서명할 것을 요구합니다.

먼저 Kucoin 계정에 로그인하여 새 API를 생성해야 하며, 다음 값을 얻게 됩니다:

- ApiKey
- ApiSecret
- Passphrase

이 필드들은 Kucoin API 클라이언트 구성 요소의 Kucoin 속성에서 구성해야 합니다.

구성이 완료되면 Kucoin Pro REST API에 비공개 요청을 시작할 수 있습니다

*비공개 요청은 로컬 컴퓨터의 시간이 동기화되어 있어야 하며, 그렇지 않으면 요청이 Kucoin 서버에 의해 거부됩니다. 이에 대해서는 다음 문서 [Kucoin Private Requests Time](#)을 확인하십시오.

```
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 | 선물 거래

Kucoin은 REST API를 사용하여 Futures를 거래할 수 있게 합니다.

구성

먼저 Kucoin 계정에서 **API Key**를 생성하고 Futures 거래 권한을 추가해야 합니다.

이 작업이 완료되면 선물 거래를 시작할 수 있습니다.

먼저, **Kucoin Client 구성 요소**에서 **ApiKey**, **ApiSecret** 및 **Passphrase**를 설정하십시오. 이는 Kucoin 서버로 전송되는 요청에 서명하는 데 사용됩니다.

Place an Order

새 주문을 배치하려면 Kucoin Client 구성 요소의 **REST_API.PlaceOrder** 메서드를 호출하면 됩니다.

주문 유형(market, limit...)에 따라 API는 더 많거나 적은 필드를 요구합니다.

Parameters

Param	type	설명
clientOid	String	주문을 식별하기 위해 사용자가 생성한 고유 주문 id입니다(예: UUID). 숫자, 문자, 밑줄(_) 및 구분자(-)만 허용됩니다
side	String	buy 또는 sell
symbol	String	유효한 계약 코드. 예: XBTUSD
type	String	[선택 사항] limit 또는 market 중 하나
leverage	String	주문의 레버리지
remark	String	[선택 사항] 주문에 대한 비고, 길이는 100 utf8 문자를 초과할 수 없습니다
stop	String	[선택 사항] down 또는 up 중 하나입니다. stopPrice 와 stopPriceType 이 정의되어야 합니다.
stopPriceType	String	[선택 사항] TP , IP 또는 MP 중 하나입니다. stop 이 지정된 경우 정의해야 합니다.
stopPrice	String	[선택 사항] stop 이 지정된 경우 정의해야 합니다.
reduceOnly	boolean	[선택 사항] 포지션 크기만 줄이는 마크입니다. 기본적으로 false로 설정됩니다. reduceOnly가 true일 때 포지션 크기를 설정해야 합니다.
closeOrder	boolean	[선택 사항] 포지션을 청산하는 표시입니다. 기본적으로 false로 설정됩니다. closeOrder가 true이면 모든 포지션을 청산합니다.
forceHold	boolean	[선택 사항] 포지션 크기를 줄이는 주문이더라도 주문에 대해 자금을 강제로 보류하는 표시입니다. 이는 주문이 order book에 유지되고 포지션 크기가 변경될 때 취소되지 않도록 도움을 줍니다. 기본적으로 false로 설정됩니다.

LIMIT ORDER PARAMETERS

Param	type	설명
price	String	Limit price
size	Integer	주문 크기입니다. 양수여야 합니다
timeInForce	String	[선택 사항] GTC , IOC (기본값은 GTC), Time In Force 를 읽으십시오
postOnly	boolean	[선택적] Post only 플래그, timeInForce 가 IOC 일 때 유효하지 않습니다. postOnly를 선택하면 hidden 또는 iceberg를 선택할 수 없습니다.
hidden	boolean	[선택적] 호가창에 표시되지 않는 주문입니다. hidden을 선택하면 postOnly를 선택할 수 없습니다.
iceberg	boolean	[선택 사항] 주문의 보이는 부분만 호가창에 표시됩니다. iceberg를 선택하면 postOnly를 선택할 수 없습니다.
visibleSize	Integer	[선택적] iceberg 주문의 최대 표시 크기

MARKET ORDER PARAMETERS

Param	type	설명
size	Integer	[선택적] 매수 또는 매도할 계약 수량

주문을 보낼 때 두 가지 가능성이 있습니다:

1. **성공:** PlaceOrder 함수는 Kucoin 서버가 보낸 메시지를 반환합니다.
2. **Error:** 예외는 OnKucointHTTPException 이벤트에서 반환됩니다.

Place Market Order 1 XBTUSDM

```

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, "XBTUSDM", 1
));

```

Place Limit Order 1 XBTUSDM 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

Kucoin에 비공개 요청을 할 때, 요청의 보안을 높이기 위해 메시지에 서명합니다. 메시지는 로컬 시간을 취하여 서명된 메시지 내부로 전송합니다. 로컬 시간이 Kucoin 서버와 5초보다 큰 차이가 있으면 요청이 거부됩니다. 따라서 로컬 시간이 동기화되어 있는지 확인하는 것이 중요하며, OS의 시간 동기화 메서드를 사용하여 이를 수행할 수 있습니다.

GetServerTime 메서드를 호출하여 Kucoin 서버 시간을 확인할 수 있으며, 이는 Kucoin 서버의 시간을 반환합니다.

API 3Commas

3Commas

지원되는 API

- **WebSockets API**: 공개 websocket 서버에 연결하고 실시간 시장 데이터 업데이트를 제공합니다.
- **REST API**: REST API에는 계정 및 주문 관리뿐만 아니라 공개 시장 데이터를 위한 엔드포인트가 있습니다.

WebSockets API

websocket 피드는 Trades 및 Deals에 대한 실시간 시장 데이터 업데이트를 제공합니다

다음 **Public** 채널을 구독할 수 있습니다:

메서드	Arguments	설명
SubscribeSmartTrades		
SubscribeDeals		

이 채널들은 3Commas 서버에 대한 **인증**이 필요합니다. 따라서 먼저 3Commas 계정에서 API 키를 요청한 다음 구성 요소의 ThreeComas 속성에 값을 설정하십시오:

- ApiKey
- ApiSecret

구독이 성공하면 **OnThreeCommasConfirmSubscription** 이벤트가 호출됩니다. 그렇지 않으면 **OnThreeCommasRejectSubscription** 이벤트가 호출되며, **aRawMessage** 매개변수를 사용하여 거부 사유를 얻을 수 있습니다.

REST API

Test Connectivity

메서드	Arguments	설명
GetPing		
GetServerTime		서버 시간을 반환합니다

Account

메서드	Arguments	설명
GetAccounts		사용자 연결된 거래소 목록

GetMarketList		지원되는 시장 목록
GetMarketPairs	aMarketCode: 마켓의 코드	모든 시장 쌍
GetCurrencyRatesWithLeverageData	aMarketCode: 시장의 코드 aPair: pair 이름	레버리지 데이터가 포함된 통화 환 율 및 제한
GetCurrencyRates	aMarketCode: 시장의 코드 aPair: pair 이름	통화 율 및 한도
GetBalances	aAccountId: 계정의 id	지정된 거래소에 대한 로 드 밸런 싱
GetAccountTableData	aAccountId: 계정의 id	지정된 거래소의 모든 사 용자 잔 액에 대 한 정보
GetAccountLeverage	aAccountId: 계정의 id aPair: pair 이름	계정 레 버리지에 대한 정 보
GetAccountInfo	aAccountId: 계정의 id	Single Account Info

Smart Trades

메서드	Arguments	설명
GetSmartTradeHistory		거래 내역 가져오기
PlaceMarketOrder	aAccountId: 계정의 id aOrderSide: 매수 또는 매도 aPair: 페어 이름 aQuantity: 수량	Market Order를 배치합니다
PlaceLimitOrder	aAccountId: 계정의 id aOrderSide: 매수 또는 매도 aPair: 쌍 이름 aQuantity: 수량	지정가 주문을 넣습니다

	aPrice: limit 가격	
GetSmartTrade	ald: 거래의 id입니다.	거래의 Id로 Smart Trade를 가져옵니다
CancelSmartTrade	ald: 거래의 id입니다.	Trade의 Id로 Smart Trade 취소
CloseByMarketSmartTrade	ald: 거래의 id입니다.	
EditSmartTrade	ald: 거래의 id입니다.	기존 Smart Trade 편집
ForceStartSmartTrade	ald: 거래의 id입니다.	Smart Trade를 강제로 시작
AddFundsSmartTrade	ald: 거래의 id입니다.	Smart Trade에 자금 추가
GetSmartTradeTrades	ald: 거래의 id입니다.	Smart Trade의 거래를 가져옵니다.

DCA Bot

메서드	Arguments	설명
CreateDCABot		새 DCA Bot 생성
GetDCABot	ald: 봇의 id	Id로 DCA Bot 가져오기
GetDCABots		모든 DCA Bot 가져오기
EnableDCABot	ald: 봇의 id	Enable a DCA Bot
DisableDCABot	ald: 봇의 id	Disable a DCA Bot
DeleteDCABot	ald: 봇의 id	DCA Bot 삭제
CancelDCABot	ald: 봇의 id	DCA Bot 취소
GetDCABotStats		Get DCA Bot statistics
GetAvailableStrategyList		사용 가능한 전략 목록을 가져옵니다.
GetBlacklistPairs		blacklist 쌍 가져오기
AddBlacklistPairs		블랙리스트 쌍 추가

거래(Deals)

메서드	Arguments	설명
GetDeals		Get all deals
GetDeal	aid: deal의 id	Id로 거래 가져오기
UpdateDeal	aid: deal의 id	거래 업데이트
CancelDeal	aid: deal의 id	거래 취소
CloseAtMarketDeal	aid: deal의 id	시장가로 거래 종료

Grid Bot

메서드	Arguments	설명
CreateGridBot		새 Grid Bot 생성
GetGridBot	aid: 봇의 id	Id로 Grid Bot 가져오기
GetGridBots		모든 Grid Bot 가져오기
EnableGridBot	aid: 봇의 id	Grid Bot 활성화
DisableGridBot	aid: 봇의 id	Grid Bot 비활성화
DeleteGridBot	aid: 봇의 id	Grid Bot 삭제

Events

OnConnect

새 WebSocket 연결이 열릴 때

OnDisconnect

WebSocket 연결이 닫힐 때

OnThreeCommasConnect

클라이언트가 3Commas 서버로부터 Welcome 메시지를 수신하면, 연결이 준비되었음을 의미합니다.

OnThreeCommasConfirmSubscription

클라이언트가 보낸 이전 구독을 확인합니다.

OnThreeCommasRejectSubscription

3Commas 채널 구독을 시도하는 중 오류가 발생했습니다

OnThreeCommasMessage

여기서 클라이언트는 구독한 채널과 관련하여 서버가 보낸 데이터를 수신합니다

OnThreeCommasPing

서버가 클라이언트에 보내는 Ping입니다.

OnThreeCommasHTTPException

HTTP REST 메서드를 호출하는 동안 오류가 있으면 이 이벤트가 호출됩니다.

API OKX

OKX

지원되는 API

- **WebSockets API:** 웹소켓 서버에 연결하여 실시간 시장 데이터 업데이트, 계정 변경 사항을 제공하고 거래 주문을 넣습니다.

Properties

WebSocket 채널은 두 가지 범주로 나뉩니다: public 및 private 채널.

- **Public 채널:** tickers 채널, K-Line 채널, limit price 채널, order book 채널, mark price 채널 등을 포함하며 로그인 필요하지 않습니다.
- **Private channels:** account 채널, order 채널, position 채널 등을 포함하며 로그인이 필요합니다.

OKX 속성에서 다음 속성을 구성할 수 있습니다.

- **ApiKey:** OKX 계정에서 새 api key를 요청하여 그 값을 이 속성에 복사하기만 하면 됩니다.
- **ApiSecret:** api의 비밀 값입니다.
- **Passphrase:** 새 api 키를 생성할 때 정의된 사용자 지정 문자열입니다.
- **IsDemo:** 활성화되면 OKX Demo 계정에 연결됩니다(기본적으로 비활성화).
- **IsPrivate:** 활성화되면 private 채널에 연결할 수 있습니다(기본적으로 비활성화됨).
- **IsBusiness:** 활성화되면 Business WebSocket 엔드포인트(wss://ws.okx.com:8443/ws/v5/business)에 연결됩니다. candle 채널, algo order 채널 및 기타 고급 채널에 이를 사용하십시오(기본적으로 비활성화됨).

연결

클라이언트가 OKX 서버에 성공적으로 연결되면 **OnOKXConnect** 이벤트가 발생합니다. 연결을 시도하는 동안 오류가 발생하면 **OnOKXError** 이벤트가 오류 세부 정보와 함께 발생합니다.

OnOKXConnect 이벤트가 발생한 후, OKX servers로 메시지를 보내고 받기 시작할 수 있습니다.

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_OKX *oOKX = new TsgcWSAPI_OKX(NULL);
oOKX->Client = oClient;
oOKX->OKX->ApiKey = "alsdj23kandfnasbdfdkjhdsf";
oOKX->OKX->ApiSecret = "aldskjfk3jkadknfajndsifj23j";
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);
}
```

Public Channels

websocket 피드는 주문 및 거래에 대한 실시간 시장 데이터 업데이트를 제공합니다. websocket 피드에는 ticker, trades 등과 같은 일부 public 채널이 있습니다...

다음 **Public 채널**을 구독할 수 있습니다:

메서드	설명
SubscribeInstruments	전체 상품 목록이 구독 후 처음으로 푸시됩니다. 이후 상품 상태에 변경 사항이 있는 경우(예: FUTURES 인도, OPTION 행사, 신규 계약/거래 쌍 상장, 거래 정지 등) 상품이 푸시됩니다.
SubscribeTicker	instrument의 최근 거래 가격, bid 가격, ask 가격 및 24시간 거래량을 검색합니다. 데이터는 100ms마다 푸시됩니다.
SubscribeOpenInterest	미결제 약정을 검색합니다. 데이터는 3초마다 푸시됩니다.
SubscribeCandlestick	상품의 캔들스틱 데이터를 검색합니다. 푸시 빈도는 가장 빠른 간격인 500ms로 데이터를 푸시합니다.
SubscribeTrades	최근 거래 데이터를 검색합니다. 거래가 있을 때마다 데이터가 푸시됩니다.
SubscribeEstimatedPrices	FUTURES 계약 및 OPTION의 예상 인도/행사 가격을 검색합니다. 예상 인도/행사 가격만 인도/행사 1시간 전에 푸시되며, 가격 변경이 있는 경우 푸시됩니다.
SubscribeMarkPrice	mark price를 검색합니다. 데이터는 mark price가 변경될 때 200 ms마다 푸시되고, mark price가 변경되지 않을 때 10초마다 푸시됩니다.
SubscribeMarkPriceCandlestick	mark price의 캔들스틱 데이터를 검색합니다. 데이터는 500 ms마다 푸시됩니다.
SubscribePriceLimit	instrument의 최대 매수 가격과 최소 매도 가격을 검색합니다. 데이터는 limit에 변경이 있을 때 5초마다 푸시되며, limit에 변경이 없을 때는 푸시되지 않습니다.
SubscribeOrderBook	order book 데이터를 검색합니다. 400개 깊이 레벨에는 books, 5개 깊이 레벨에는 book5, 틱 단위 1개 깊이 레벨에는 bbo-tbt, 틱 단위 50개 깊이 레벨에는 books50-l2-tbt, 틱 단위 400개 깊이 레벨에는 books-l2-tbt를 사용하십시오. • books: 초기 전체 스냅샷에서 400개의 깊이 레벨이 푸시됩니다. 주문서에 변경이 있을 때 100 ms마다 증분 데이터가 푸시됩니다. • books5: 매번 5개의 깊이 레벨이 푸시됩니다. 주문서에 변경이 있을 때 100 ms마다 데이터가 푸시됩니다. • bbo-tbt: 매번 1개의 깊이 레벨이 푸시됩니다. 주문서에 변경이 있을 때 10 ms마다 데이터가 푸시됩니다. • books-l2-tbt: 초기 전체 스냅샷에서 400개의 깊이 레벨이 푸시됩니다. 주문서에 변경이 있을 때 10 ms마다 증분 데이터가 푸시됩니다. • books50-l2-tbt: 초기 전체 스냅샷에서 50개의 깊이 레벨이 푸시됩니다. 주문서에 변경이 있을 때 10 ms마다 증분 데이터가 푸시됩니다. asks 또는 bids가 빈 배열이면 이는 50 깊이가 아닌 400 깊이에 변경이 있음을 의미합니다. 주문서 데이터를 로컬에서 유지하는 경우 빈 asks와 bids는 무시하십시오.
SubscribeOptionSummary	모든 OPTION 계약의 상세 가격 정보를 검색합니다. 데이터는 한 번에 푸시됩니다.
SubscribeFundingRate	펀딩 비율을 검색합니다. 데이터는 30초에서 90초 내에 푸시됩니다.
SubscribeIndexCandlestick	인덱스의 캔들스틱 데이터를 검색합니다. 데이터는 500ms마다 푸시됩니다.
SubscribeIndexTicker	index ticker 데이터를 검색합니다
SubscribeStatus	시스템 유지 보수 상태를 가져오고 시스템 유지 보수 상태가 변경될 때 푸시합니다. 첫 구독: "최신 변경 데이터 푸시"; 상태 변경이 있을 때마다 변경된 내용을 푸시
SubscribePublicStructureBlockTrades	블록 거래가 있을 때마다 데이터가 푸시됩니다.
SubscribeBlockTickers	지난 24시간 동안의 최신 블록 거래량을 검색합니다. 데이터는 거래 실행 이벤트에 의해 트리거될 때 푸시됩니다. 또한 구독 단위에 따라 5분 간격으로도 푸시됩니다.
SubscribeAllTrades	모든 거래 데이터를 검색합니다. 거래가 있을 때마다 데이터가 푸시됩니다.
SubscribeLiquidationOrders	청산 주문을 검색합니다. 청산 주문이 있을 때 데이터가 푸시됩니다.

SubscribeADLWarning	ADL 경고 데이터를 검색합니다. 자동 디레버리징 위험이 증가하면 데이터가 푸시됩니다.
SubscribeEconomicCalendar	경제 캘린더 이벤트를 검색합니다. 경제 이벤트에 업데이트가 있을 때 데이터가 푸시됩니다.
SubscribePublicBlockTrades	public block trades를 검색합니다. block trade가 있을 때마다 데이터가 푸시됩니다.
SubscribeOptionTrades	옵션 거래 데이터를 검색합니다. 옵션 거래가 있을 때마다 데이터가 푸시됩니다.
SubscribeCallAuctionDetails	call auction 세부 정보를 검색합니다. call auction 정보에 업데이트가 있으면 데이터가 푸시됩니다.

```

TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_OKX *oOKX = new TsgcWSAPI_OKX(NULL);
oOKX->Client = oClient;
oOKX->OKX->ApiKey = "alsdkj23kandfnasbdfdkjhdsdf";
oOKX->OKX->ApiSecret = "aldskjfk3jkadknfajndsrfj23j";
oOKX->OKX->Passphrase = "secret_passphrase";
oClient->Active = true;
void OnOKXConnect(TObject *Sender, string aMessage, string aCode, string aRawMessage)
{
    oOKX->SubscribeInstruments(okxitFutures);
}

```

Private 채널

account 채널, order 채널, position 채널 등을 포함하며 로그인에 필요합니다.

다음 **Private** 채널을 구독할 수 있습니다:

메서드	설명
SubscribeAccount	계정 정보를 검색합니다. 주문 등록, 주문 취소, 거래 실행 등의 이벤트에 의해 트리거될 때 데이터가 푸시됩니다. 또한 구독 세분성에 따라 정기적인 간격으로 푸시됩니다.
SubscribePositions	포지션 정보를 검색합니다. 구독 단위에 따라 초기 스냅샷이 푸시됩니다. 주문 넣기/취소와 같은 이벤트에 의해 트리거될 때 데이터가 푸시되며, 구독 단위에 따라 정기적인 간격으로도 푸시됩니다.
SubscribeBalanceAndPosition	계정 잔액과 포지션 정보를 검색합니다. 체결된 주문, 펀딩 이체와 같은 이벤트에 의해 트리거될 때 데이터가 푸시됩니다.
SubscribeOrders	주문 정보를 검색합니다. 처음 구독할 때는 데이터가 푸시되지 않습니다. 주문 배치/취소와 같은 이벤트에 의해 트리거될 때만 데이터가 푸시됩니다.
SubscribeOrdersAlgo	algo 주문(trigger 주문, oco 주문, conditional 주문 포함)을 검색합니다. 데이터는 처음 구독할 때 푸시되지 않습니다. 데이터는 주문 배치/취소와 같은 이벤트에 의해 트리거될 때만 푸시됩니다.
SubscribeAdvanceAlgo	고급 algo 주문(Iceberg 주문, TWAP 주문, Trailing 주문 포함)을 검색합니다. 처음 구독 시 데이터가 푸시됩니다. 주문 넣기/취소와 같은 이벤트로 트리거될 때 데이터가 푸시됩니다.
SubscribePositionRisk	이 푸시 채널은 위험 경고로만 사용되며, 전략적 거래를 위한 위험 판단으로 권장되지 않습니다. 시장이 격렬하게 움직이지 않는 경우, 이 메시지가 푸시되는 동시에 포지션이 청산되었을 가능성이 있습니다.
SubscribeAccountGreeks	계정 greeks 정보를 검색합니다. 데이터는 계정의 포지션 증가/감소 또는 현금 잔액과 같은 이벤트에 의해 트리거될 때 푸시되며, 구독 세분성에 따라 정기적인 간격으로도 푸시됩니다.
SubscribeRfq	Rfq를 검색합니다.

SubscribeQuotes	Quotes를 검색합니다.
SubscribePrivateStructureBlockTrades	구조화된 블록 거래를 검색합니다.
SubscribeSpotGridAlgoOrders	spot grid algo 주문을 검색합니다. 처음 구독할 때 데이터가 푸시됩니다. 주문 배치/취소와 같은 이벤트에 의해 트리거될 때 데이터가 푸시됩니다.
SubscribeContactGridAlgoOrders	계약 grid algo 주문을 검색합니다. 데이터는 처음 구독할 때 푸시됩니다. 데이터는 주문 배치/취소와 같은 이벤트에 의해 트리거될 때 푸시됩니다.
SubscribeGridPositions	grid 포지션을 검색합니다. 데이터는 처음 구독할 때 푸시됩니다. 데이터는 주문 배치/취소와 같은 이벤트에 의해 트리거될 때 푸시됩니다.
SubscribeGridSubOrders	grid 하위 주문을 검색합니다. 데이터는 처음 구독할 때 푸시됩니다. 데이터는 주문 배치와 같은 이벤트에 의해 트리거될 때 푸시됩니다.
SubscribeFills	체결된 주문 데이터를 검색합니다. 주문이 체결되면 데이터가 푸시됩니다.
SubscribeDepositInfo	입금 정보를 검색합니다. 입금 상태 업데이트가 있을 때 데이터가 푸시됩니다.
SubscribeWithdrawalInfo	출금 정보를 검색합니다. 데이터는 출금 상태 업데이트가 있을 때 푸시됩니다.

```

TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_OKX *oOKX = new TsgcWSAPI_OKX(NULL);
oOKX->Client = oClient;
oOKX->OKX->ApiKey = "alsdkj23kandfnasbdfdkjhdsdf";
oOKX->OKX->ApiSecret = "aldskjfk3jkadknfajndsxfj23j";
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

WebSocket Trade는 **Authentication**을 요구합니다.

충분한 자금이 있는 경우에만 주문할 수 있습니다. 아래에서 요청 매개변수가 있는 표를 확인하십시오:

매개변수	Type	필수	설명
id	String	Yes	메시지의 고유 식별자입니다 클라이언트가 제공합니다. 해당 요청을 식별하기 위해 응답 메시지에서 반환됩니다. 대소문자를 구분하는 영숫자, 모두 숫자 또는 모두 문자로 최대 32자까지의 조합입니다.
> instId	String	Yes	Instrument ID, 예: BTC-USD-190927-5000-C
> tdMode	String	Yes	Trade mode Margin mode isolated cross Non-Margin mode cash
> ccy	String	No	마진 통화 Single-currency margin 의 cross MARGIN 주문에만 적용됩니다.
> clOrdId	String	No	클라이언트가 제공한 주문 ID 대소문자를 구분하는 영숫자, 모두 숫자 또는 모두 문자의 조합으로 최대 32자.
> tag	String	No	주문 태그 대소문자를 구분하는 영숫자, 모두 숫자 또는 모두 문자의 조합으로 최대 16자.
> side	String	Yes	Order side, buy sell

매개변수	Type	필수	설명
> posSide	String	선택적	Position side 기본값은 <code>net</code> 모드에서 <code>net</code> 입니다 <code>long/short</code> 모드에서는 필수이며, <code>long</code> 또는 <code>short</code> 만 될 수 있습니다. <code>FUTURES/SWAP</code> 에만 적용됩니다.
> ordType	String	Yes	주문 유형 <code>market</code> : market order <code>limit</code> : limit order <code>post_only</code> : Post-only order <code>fok</code> : Fill-or-kill order <code>ioc</code> : Immediate-or-cancel order <code>optimal_limit_ioc</code> : immediate-or-cancel order가 있는 Market order
> sz	String	Yes	매수 또는 매도할 수량입니다.
> px	String	선택적	Price <code>limit</code> , <code>post_only</code> , <code>fok</code> , <code>ioc</code> 주문에만 적용됩니다.
> reduceOnly	Boolean	No	포지션을 축소만 할지 여부, <code>true</code> <code>false</code> , 기본값은 <code>false</code> 입니다. <code>MARGIN</code> 주문 및 <code>net</code> 모드의 <code>FUTURES/SWAP</code> 주문에만 적용됩니다 <code>Single-currency margin</code> 및 <code>Multi-currency margin</code> 에만 적용됩니다
> tgtCcy	String	No	수량 유형 <code>base_ccy</code> : Base currency, <code>quote_ccy</code> : Quote currency Market 주문으로 거래되는 <code>SPOT</code> 에만 적용됩니다 기본값은 매수의 경우 <code>quote_ccy</code> , 매도의 경우 <code>base_ccy</code> 입니다
> banAmend	Boolean	No	현물 주문 수정을 금지할지 여부입니다(true 또는 false, 기본값은 false). banAmend가 true일 때 잔액이 충분하지 않으면 주문 발주에 실패합니다. SPOT 시장가 주문에만 적용됩니다.

주문 발주 예제

충분한 자금이 있는 경우에만 주문할 수 있습니다.

```
// Place Martket Order
TsgcWSAPI_OKX1->PlaceMarketOrder(okxosBuy, "ETH-BTC", 1);
// Place Limit Order
TsgcWSAPI_OKX1->PlaceLimitOrder(okxosBuy, "ETH-BTC", 1, 0.25);
```

주문 취소 예제

미완료 주문을 취소합니다

```
TsgcWSAPI_OKX1->CancelOrder("
ETH-BTC
", "457589362405027840");
```

Amend Order

미완료 주문을 수정합니다.

```
TsgcWSAPI_OKX1->AmendOrder("ETH-BTC", "457589362405027840", "", 2);
```

Batch Trade Operations

WebSocket Trade API는 또한 여러 주문을 한 번에 발주, 취소 및 수정하는 배치 작업을 지원합니다. 이러한 작업에는 **Authentication**이 필요합니다.

메서드	설명
BatchPlaceOrders	단일 요청으로 여러 주문을 실행합니다. 한 번에 최대 20개의 주문을 실행할 수 있습니다.
BatchCancelOrders	단일 요청으로 여러 주문을 취소합니다. 한 번에 최대 20개의 주문을 취소할 수 있습니다.
BatchAmendOrders	단일 요청으로 여러 개의 미완료 주문을 수정합니다. 한 번에 최대 20개의 주문을 수정할 수 있습니다.
MassCancelOrders	특정 상품 유형에 대한 모든 대기 중인 주문을 일괄 취소합니다.

API XTB

XTB

지원되는 API

- **WebSockets API**: websocket 서버에 연결하여 실시간 시장 데이터 업데이트, 계정 변경 및 거래 주문 발주를 제공합니다.

Properties

WebSocket 프로토콜은 2가지 유형의 요청을 허용합니다: **Streaming commands**(실시간 업데이트 수신) 및 **Retrieve Trading Data**(일부 정보를 검색하기 위해 서버에 요청 전송).

XTB 속성에서 다음 속성을 구성할 수 있습니다.

- **User**: 연결을 식별하는 사용자 이름.
- **Password**: 사용자의 secret 값입니다.
- **Demo**: 활성화하면 XTB Demo 계정에 연결합니다(기본적으로 비활성화됨).

연결

클라이언트가 XTB 서버에 성공적으로 연결되면 **OnXTBConnect** 이벤트가 발생합니다. 연결을 시도하는 동안 오류가 발생하면 **OnXTBError** 이벤트가 오류 세부 정보와 함께 발생합니다.

OnXTBConnect 이벤트가 발생한 후에는 XTB 서버에서 메시지를 송신하고 수신하기 시작할 수 있습니다.

```
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);
}
```

연결 명령

메서드	설명
Login	작업을 수행하려면 클라이언트 애플리케이션이 로그인 프로세스를 수행해야 합니다. 적절한 로그인 프로세스 전에는 어떤 기능도 사용할 수 없습니다. login 메서드는 클라이언트가 websocket 서버에 연결되고 User/Password 값이 설정된 후 자동으로 호출됩니다.
Logout	

Streaming Commands

다음 채널을 구독할 수 있습니다:

메서드	설명
SubscribeBalance	시스템에서 사용 가능해지는 즉시 실시간으로 실제 계정 지표 값을 가져올 수 있게 합니다.
SubscribeCandles	API 차트 캔들을 구독하고 구독 취소합니다. 모든 캔들의 간격은 1분입니다. 새 캔들이 매분 도착합니다.
SubscribeKeepAlive	'keep alive' 메시지를 구독하고 구독 취소합니다. 새 'keep alive' 메시지는 API에서 3초마다 전송됩니다.
SubscribeNews	뉴스를 구독하고 구독을 취소합니다.
SubscribeProfits	profits를 구독하고 구독 취소합니다.
SubscribeTickPrices	시세에 대한 구독을 설정하고 시스템에서 사용 가능해지는 즉시 관련 정보를 실시간으로 얻을 수 있도록 합니다. getTickPrices 명령은 동일한 심볼에 대해 여러 번 호출할 수 있지만, 주어진 심볼에 대해서는 하나의 구독만 생성됩니다. 여러 레코드가 사용 가능한 경우 수신되는 순서는 보장되지 않으니 주의하십시오.
SubscribeTrades	사용자 거래 상태 데이터에 대한 구독을 설정하고, 시스템에서 사용 가능해지는 즉시 관련 정보를 실시간으로 얻을 수 있게 합니다. 여러 레코드가 사용 가능한 경우, 수신되는 순서는 보장되지 않습니다.
SubscribeTradeStatus	전송된 거래 요청에 대한 상태를 시스템에서 사용 가능해지는 즉시 실시간으로 가져올 수 있습니다. 여러 레코드가 사용 가능할 때 수신되는 순서는 보장되지 않음에 유의하십시오
SubscribePing	이 함수를 정기적으로 호출하는 것만으로 시스템의 모든 구성 요소의 내부 상태를 새로 고치기에 충분합니다. 스트리밍 연결은 세션에서 클라이언트가 명령을 보내지 않으면 단방향 네트워크 트래픽만 생성합니다. 다른 명령을 실행하지 않는 모든 애플리케이션은 최소 10분마다 한 번씩 이 명령을 호출하는 것이 권장됩니다.

거래 데이터 검색

다음 요청을 전송할 수 있습니다:

메서드	설명
GetAllSymbols	사용자가 사용할 수 있는 모든 심볼의 배열을 반환합니다.
GetCalendar	시장 이벤트가 있는 캘린더를 반환합니다.
GetChartLastRequest	이 함수는 일반적으로 현재 candle 데이터를 검색하는 데 선호되는 방법인 스트리밍 등가물 getCandles로 대체될 수 있습니다. start date부터 현재 시간까지의 차트 정보를 반환합니다. 선택한 CHART_LAST_INFO_RECORD의 기간이 1분보다 큰 경우, API가 반환하는 마지막 candle은 기간이 끝날 때까지 변경될 수 있습니다 (candle은 매분 자동으로 업데이트됩니다).
GetChartRangeRequest	이 함수는 일반적으로 현재 캔들 데이터를 검색하는 권장 방법인 스트리밍 대응 함수 getCandles로 대체될 수 있음에 유의하십시오. 지정된 시작 및 종료 날짜 사이의 데이터가 포함된 차트 정보를 반환합니다.
GetCommissionDef	수수료와 환율 계산을 반환합니다. 값은 예상 값으로 계산되므로 완벽하게 정확하지 않을 수 있습니다.
GetCurrentUserData	계정 통화 및 계정 레버리지에 대한 정보를 반환합니다.

GetIbsHistory	지정된 시간 범위의 IBs 데이터를 반환합니다.
GetMarginLevel	이 함수는 일반적으로 스트리밍 등가물인 getBalance로 대체될 수 있으며, 이것이 계정 지표를 검색하는 선호되는 방법입니다. 다양한 계정 지표를 반환합니다
GetMarginTrade	주어진 약기와 거래량에 대한 예상 증거금을 반환합니다. 값은 예상 증거금 값으로 계산되므로 완벽하게 정확하지 않을 수 있습니다.
GetNews	이 함수는 일반적으로 뉴스 데이터를 검색하는 선호되는 방법인 스트리밍 등가물 getNews로 대체할 수 있다는 점을 유의하십시오. 지정된 기간 내에 전송된 거래 서버의 뉴스를 반환합니다.
GetProfitCalculation	주어진 거래 데이터에 대한 예상 수익을 계산합니다. calculator와 같은 앱에만 사용해야 합니다. 열린 거래의 수익은 서버 계산의 더 높은 정밀도로 인해 서버에서 가져와야 합니다
GetServerTime	거래 서버의 현재 시간을 반환합니다.
GetStepRules	DMA에 대한 step rule 목록을 반환합니다.
GetSymbol	사용자가 사용 가능한 심볼에 대한 정보를 반환합니다
GetTickPrices	이 함수는 일반적으로 tick 데이터를 검색하는 데 선호되는 방법인 스트리밍 등가물 getTickPrices로 대체될 수 있습니다. 주어진 symbol에 대한 현재 시세 배열을 반환하며, 주어진 타임스탬프에서 변경된 시세만 반환됩니다. 출력에서 얻은 새 타임스탬프는 이 명령의 다음 호출의 인수로 사용됩니다.
GetTradeRecords	orders 인수에 나열된 거래 배열을 반환합니다
GetTrades	이 함수는 일반적으로 거래 데이터를 검색하는 선호되는 방법인 스트리밍 등가물 getTrades로 대체될 수 있다는 점에 유의하십시오. 사용자의 거래 배열을 반환합니다.
GetTradesHistory	이 함수는 일반적으로 거래 데이터를 검색하는 데 선호되는 방식인 스트리밍 등가 함수 getTrades로 대체될 수 있다는 점에 유의하십시오. 지정된 기간 내에 청산된 사용자 거래 배열을 반환합니다.
GetTradingHours	호가와 거래 시간을 반환합니다.
GetVersion	현재 API 버전을 반환합니다.
Ping	이 함수를 정기적으로 호출하는 것만으로도 시스템의 모든 구성 요소의 내부 상태를 새로 고치기에 충분합니다. 다른 명령을 실행하지 않는 모든 애플리케이션은 이 명령을 최소 10분마다 한 번씩 호출하는 것이 권장됩니다. 이 함수의 스트리밍 대응물은 ping과 getKeepAlive의 조합입니다
TradeTransaction	거래 트랜잭션을 시작합니다. tradeTransaction은 주요 트랜잭션 정보를 서버에 보냅니다.
TradeTransactionStatus	이 함수는 일반적으로 트랜잭션 상태 데이터를 검색하는 데 선호되는 방법인 스트리밍 등가물 getTradeStatus로 대체될 수 있습니다. 현재 트랜잭션 상태를 반환합니다. 트랜잭션 처리 중 언제든지 클라이언트는 서버 측에서 트랜잭션 상태를 확인할 수 있습니다. 이를 위해 클라이언트는 tradeTransaction 호출에서 가져온 고유 주문을 제공해야 합니다.

API Bybit

Bybit

지원되는 API

- **WebSocket API:** websocket 서버에 연결하고 실시간 시장 데이터 업데이트, 계정 변경 등을 제공합니다.
- **REST API:** HTTP 요청을 보내 시장 데이터를 가져오고, 주문하고, 계정 데이터를 가져옵니다...

현재 API 지원 버전은 V5입니다. V5 API는 Bybit의 제품 라인에 통일성과 효율성을 가져오며, Spot, Derivatives 및 Options를 하나의 사양 집합으로 통합합니다.

OpenAPI Version	계정 유형	Linear			Inverse		Spot	Options
		USDT Perpetual	USDC Perpetual	USDC Futures	Perpetual	Futures		
V5	Unified trading account	✓	✓	✓	참고 사항 참조		✓	✓
	Classic account	✓			✓	✓	✓	
V3	Unified trading account	✓	✓					✓
	Classic account	✓			✓	✓	✓	

*참고: Unified 계정은 inverse 거래를 지원합니다. 그러나 사용되는 마진은 unified 지갑이 아닌 inverse derivatives 지갑에서 나옵니다.

Properties

Bybit 속성에서 다음 속성을 구성할 수 있습니다.

- **ApiKey:** Bybit 계정에서 새 api 키를 요청하고 값을 이 속성에 복사하기만 하면 됩니다. ApiKey가 설정되면 클라이언트는 웹소켓 비공개 서버에 연결됩니다. 비어 있으면 WebSocket 공개 서버에 연결됩니다.
- **ApiSecret:** api의 비밀 값입니다.
- **SignatureExpires:** 서명이 만료되는 초 수입니다(기본값 10초).
- **TestNet:** 활성화하면 Bybit TestNet 데모 계정에 연결합니다(기본적으로 비활성화됨).

연결

클라이언트가 Bybit 서버에 성공적으로 연결되면 **OnConnect** 이벤트가 발생합니다. **OnConnect** 이벤트가 발생한 후, Bybit 서버로/로부터 메시지를 전송하고 수신하기 시작할 수 있습니다. private websocket 채널에 연결하는 경우, 채널을 구독하기 전에 **OnBybitAuthentication** 이벤트가 발생할 때까지 기다리고 success 매개변수가 true인지 확인해야 합니다.

클라이언트는 여러 API를 지원하므로 BybitClient 속성을 사용하여 사용할 API를 설정하십시오:

- **bybSpot**
- **bybInverse**
- **bybLinear**
- **bybPerpetual**

WebSocket Spot Private API에 연결하는 예제는 아래에서 확인하십시오.

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bybit *oBybit = new TsgcWSAPI_Bybit(NULL);
oBybit->Client = oClient;
oBybit->Bybit->ApiKey = "alsdj23kandfnasbdfdkjhdsf";
oBybit->Bybit->ApiSecret = "aldskjfk3jkadknfajnds23j";
oBybit->BybitClient = bybSpot;
oClient->Active = true;
void OnConnect(TsgcWSConnection *Connection)
{
    DoLog("#Bybit Connected");
}
```

Spot WebSocket 서버에 성공적으로 연결한 후 WebSocket 채널 구독을 시작할 수 있습니다. **SPOT** 속성에 액세스한 다음 사용 가능한 subscribe/unsubscribe 메서드를 호출하기만 하면 됩니다.

Events

bybit 클라이언트는 연결 흐름을 제어하고 WebSocket 서버에서 전송된 데이터를 가져오기 위해 다음 이벤트를 구현합니다:

- **OnConnect:** websocket 클라이언트가 WebSocket Server에 연결할 때 발생합니다.
- **OnDisconnect:** websocket 클라이언트가 WebSocket Server에서 연결을 끊을 때 발생합니다.
- **OnBybitAuthentication:** 클라이언트가 Private WebSocket 서버에 대해 인증할 때 발생합니다.
- **OnBybitSubscribe:** 클라이언트가 websocket 채널을 구독할 때.
- **OnBybitUnSubscribe:** 클라이언트가 websocket 채널에서 구독을 해제할 때.
- **OnBybitData:** 클라이언트가 서버로부터 데이터를 수신할 때.
- **OnBybitError:** bybit websocket 연결 중 오류가 있을 때.
- **OnBybitHTTPException:** REST 요청 중에 오류가 있을 때입니다.

WebSocket API

websocket 피드는 주문 및 거래에 대한 실시간 시장 데이터 업데이트를 제공합니다. websocket 피드에는 ticker, trades 등과 같은 일부 public 채널이 있습니다...

다음 채널을 구독할 수 있습니다:

메서드	공개 또는 비 공개	설명
SubscribeOrderBook	Public	orderbook 스트림을 구독합니다. 다양한 depth를 지원합니다.
SubscribeTrade	Public	최근 거래 스트림을 구독합니다.
SubscribeTicker	Public	ticker 스트림을 구독합니다.
SubscribeKLine	Public	klines 스트림을 구독합니다.
SubscribeLiquidation	Public	liquidation 스트림을 구독합니다
SubscribeLT_KLine	Public	leveraged token kline 스트림을 구독합니다.
SubscribeLT_Ticker	Public	leveraged token 티커 스트림을 구독합니다.
SubscribeLT_Nav	Public	leveraged token 티커 스트림을 구독합니다.
SubscribePosition	Private	leveraged token nav 스트림을 구독합니다.
SubscribeExecution	Private	구독
SubscribeOrder	Private	구독

SubscribeWallet	Private	구독
SubscribeGreek	Private	구독
SubscribeDcp	Private	구독
SubscribeInsurance	Public	보험 기금 스트림을 구독합니다.
SubscribeOrderPriceLimit	Public	주문 가격 제한 스트림을 구독합니다.
SubscribeADLAlert	Public	자동 디레버리지 알림 스트림을 구독합니다.
SubscribeFastExecution	Private	fast execution 스트림을 구독합니다.

성공적인 인증 후 private websocket 채널을 구독하는 예제는 아래에서 확인하십시오.

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bybit *oBybit = new TsgcWSAPI_Bybit(NULL);
oBybit->Client = oClient;
oBybit->Bybit->ApiKey = "alsdk23kandfnasbdfdkjhdsdf";
oBybit->Bybit->ApiSecret = "aldskjfk3jkadknfajndsjsfj23j";
oBybit->BybitClient = bybSpot;
oClient->Active = true;
void OnBybitAuthentication(TObject *Sender, bool aSuccess, const string aError, const string aRawMessage)
{
    if (aSuccess == true)
    {
        oClient->SubscribeOrderBook("BTCUSDT");
        oClient->SubscribeTrade("BTCUSDT");
    }
}
```

REST API

REST API에는 마켓, 비공개 계정 및 지갑으로부터 데이터를 요청하기 위한 Public 및 Private 메서드 목록이 있습니다. 사용 가능한 메서드 목록은 아래를 참조하십시오.

메서드	공개 / 비공개
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

미결제 주문을 가져오는 예제는 아래를 참조하십시오.

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bybit *oBybit = new TsgcWSAPI_Bybit(NULL);
oBybit->Client = oClient;
oBybit->Bybit->ApiKey = "a1sdjk23kandfnasbdfdkjhdsf";
oBybit->Bybit->ApiSecret = "a1dskjfk3jkadknfajndsxfj23j";
oBybit->BybitClient = bybSpot;
oBybit->REST_API->GetAccountInfo();
```

API Forex

Forex.com (StoneX / CityIndex)

Forex.com은 StoneX가 운영하는 소매 거래 장소입니다. 구성 요소는 Forex.com, StoneX 및 CityIndex(세 곳 모두 동일한 플랫폼을 공유함)가 사용하는 Trading REST API와 Lightstreamer 기반 스트리밍 서비스를 래핑합니다. 다음 API가 지원됩니다:

1. **Streaming API:** WebSocket을 통한 Lightstreamer TLCP 2.5입니다. 가격 틱, 주문 업데이트, 포지션 업데이트, 계정 마진 및 시세가 성공적인 REST LogOn 후 실시간으로 푸시됩니다.
2. **REST API:** 세션 로그인(UserName + Password + AppKey), 거래 및 stop-limit 주문 관리, 계정 쿼리 및 시장 데이터. HTTPS를 사용합니다.

구성 요소

- **TsgcWSAPI_Forex:** 주 클라이언트입니다. **Credentials**를 보유하고, **REST_API**, 내부 **Lightstreamer** 스트리밍 클라이언트 및 **OnForex*** 이벤트를 노출합니다.
- **TsgcHTTP_API_Forex:** REST 구성 요소로, 메인 클라이언트의 **REST_API** 속성을 통해 액세스할 수 있습니다.
- **TsgcWSPClient_Lightstreamer:** 스트리밍 구독에 사용되는 내부 TLCP 2.5 서버프로토콜 클라이언트입니다 (PRICES / ORDERS / QUOTES / CLIENTACCOUNTMARGIN / TRADEMARGIN).

Published Properties

- **Credentials:** REST API에 로그인하는 데 필요한 필드 그룹입니다.
 - **UserName:** Forex.com / StoneX / CityIndex 계정 사용자 이름입니다.
 - **Password:** 계정 비밀번호입니다.
 - **AppKey:** 거래소에서 발급한 애플리케이션 키입니다.
 - **AppVersion:** 선택적 클라이언트 버전 문자열입니다.
 - **AppComments:** 로그인 시 로깅되는 자유 형식 텍스트입니다.
 - **StreamingHost:** Lightstreamer 호스트(기본값은 공개 스트리밍 호스트)입니다.
- **AutoReconnect:** 활성화되면 기본 WebSocket 클라이언트 WatchDog로 전달되어 네트워크 끊김 후 스트리밍 세션이 자동으로 재연결됩니다(기본값 true).
- **REST_API:** TsgcHTTP_API_Forex 구성 요소에 대한 참조입니다.
- **Lightstreamer:** 내부 TsgcWSPClient_Lightstreamer 서버프로토콜 클라이언트에 대한 참조입니다(AdapterSet은 STREAMINGALL로 고정됨).
- **SessionToken:** 읽기 전용입니다. REST LogOn 호출이 반환한 세션 토큰입니다. Lightstreamer의 LS_password로 사용됩니다.
- **ClientAccountId / TradingAccountId:** 읽기 전용. 성공적인 LogOn 후에 채워집니다.

게시된 이벤트

OnForexConnect

스트리밍 세션이 생성되고 구독을 시작할 수 있게 되면 발생합니다.

OnForexDisconnect

스트리밍 세션이 종료될 때(서버 닫기, 네트워크 끊김 또는 명시적 Disconnect) 발생합니다.

OnForexException

REST 오류(LogOn 실패 포함) 및 스트리밍 세션 결함 시 발생합니다.

OnForexPriceTick

모든 PRICES 업데이트에 대해 발생합니다: Bid, Offer, Price, Change, Direction, AuditId, TickDate.

OnForexOrderUpdate

모든 ORDERS 업데이트(신규, 체결, 취소, 거부)에 대해 발생합니다.

OnForexPositionUpdate

활성 TradingAccount의 모든 미결제 포지션 업데이트에 대해 발생합니다.

OnForexAccountMargin

모든 CLIENTACCOUNTMARGIN / TRADEMARGIN 업데이트(equity, margin, free margin, currency)에 대해 발생합니다.

OnForexQuote

모든 QUOTES 업데이트(request-for-quote 워크플로)에 대해 발생합니다.

OnForexSubscribe

서버가 Lightstreamer 구독을 확인 할 때 마 다 발 생 합 니 다(SUBOK / SUBCMD). 매 개 변 수: *aAdapter*(PRICES, ORDERS, QUOTES, CLIENTACCOUNTMARGIN 또는 TRADEMARGIN), *aGroup*(항목 그룹, 예: *ID.<ClientAccountId>*), *alters*, *aFields*.

OnForexUnsubscribe

구독이 종료될 때 발생합니다. **UnwatchMarket**을 통해, 스트리밍 연결 끊김 시, 또는 서버가 UNSUB 프레임을 보낼 때입니다. 매개변수: *aAdapter*, *aGroup*.

Most common uses

- REST API
 - [REST API에 연결하는 방법](#)
 - [Trade 및 Stop-Limit 주문을 실행하는 방법](#)
 - [계정, 포지션 및 주문 내역을 조회하는 방법](#)
 - [시장 데이터를 가져오는 방법](#)
- WebSockets API
 - [WebSocket API에 연결하는 방법](#)
 - [Streaming Adapters를 구독하는 방법](#)

Sample

클라이언트를 생성하고, 자격 증명을 설정하고, 연결하고, 시장 가격 스트림을 구독하고, 연결을 끊습니다.

```
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

Forex.com / StoneX / CityIndex Trading API에 대한 모든 호출에는 활성 세션이 필요합니다. 세션 상태는 **LogOn**으로 얻고, **LogOff**로 종료하며, **Ping**으로 활성 상태를 유지합니다.

자격 증명

메인 클라이언트 구성 요소에 자격 증명을 설정하십시오. 첫 호출 시 REST 세션에 복사됩니다.

- **UserName:** Forex.com / StoneX / CityIndex 계정 사용자 이름입니다.
- **Password:** 계정 비밀번호.
- **AppKey:** 거래소가 발급한 애플리케이션 키.
- **AppVersion / AppComments:** 로그인 시 기록되는 선택적 문자열입니다.

Session 메서드

Method	매개변수	설명
LogOn	UserName, Password, AppKey	REST 세션을 엽니다. 전체 JSON 응답을 반환하고 메인 클라이언트에 SessionToken, ClientAccountId 및 TradingAccountId를 저장합니다. 이후의 모든 REST 및 스트리밍 호출은 해당 토큰으로 인증합니다.
LogOff		서버에서 세션을 무효화하고 토큰을 지웁니다.
Ping		세션을 유지합니다. 서버 세션은 일정 기간 비활성 상태가 지나면 만료됩니다. 다른 REST 호출이 이루어지지 않을 때는 주기적으로 Ping을 보내십시오.
GetServiceStatus		현재 서비스 상태를 보고합니다(인증 필요 없음).

Token Flow

LogOn이 반환한 토큰은 모든 후속 REST 요청에 *UserName* HTTP 헤더로 첨부됩니다. 동일한 토큰은 Lightstreamer 세션이 생성될 때 스트리밍 클라이언트에 의해 *LS_password* 필드로 재사용됩니다. **LogOn**을 두 번 호출할 필요는 없습니다. 메인 클라이언트에서 **Connect**를 호출하면 먼저 REST **LogOn**을 수행한 다음에야 스트리밍 소켓을 엽니다.

Sample

REST 세션을 열고 Ping을 호출합니다

```
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

REST API는 다섯 개의 order-management 엔드포인트를 노출합니다. 각 엔드포인트는 호출자가 필요한 제어 수준을 선택할 수 있도록 세 가지 오버로드 형식으로 제공됩니다.

Endpoints

Method	설명
NewTradeOrder	CFD 시장에 market / at-best 거래 주문을 배치합니다.
UpdateTradeOrder	기존 미결제 거래에 연결된 stop / limit 보호를 연결하거나 변경합니다.
NewStopLimitOrder	보류 중인 stop 또는 limit 주문(아직 시장에 없음)을 배치합니다.
UpdateStopLimitOrder	기존 보류 중인 stop 또는 limit order를 수정합니다.
CancelOrder	보류 중인 스탑 또는 지정가 주문을 취소합니다.

Overload Tiers

Order 메서드는 두 가지 오버로드로 제공됩니다:

1. **스칼라 매개변수 오버로드**(NewTradeOrder 및 CancelOrder만 해당) — 가장 많이 사용되는 필드(NewTradeOrder의 경우 MarketId, TradingAccountId, Direction, Quantity, BidPrice / OfferPrice, AuditId; CancelOrder의 경우 OrderId, TradingAccountId, MarketId)를 받습니다. 헬퍼가 내부적으로 요청을 빌드합니다. 단순한, 보호 없는 주문에 유용합니다.
2. **타입이 지정된 order-object 오버로드** — TsgcHTTPForexTradeOrder, TsgcHTTPForexStopLimitOrder 또는 TsgcHTTPForexCancelOrder를 받습니다. 이 객체들은 IfDone(연결된 stop / limit 브래킷), Close 배열(close-by-order-id 의미), PositionMethodId, Reference 및 부분 체결 플래그를 포함한 모든 문서화된 필드를 노출합니다. 단순한 market 주문을 넘어서는 모든 것에 권장되는 경로입니다. 객체 빌드 규칙은 Kucoin 같은 다른 거래소와 공유됩니다.

TsgcHTTPForexTradeOrder 필드

- **MarketId**: ListCfdMarkets 또는 FullSearchWithTags가 반환한 숫자 market id.
- **TradingAccountId**: 활성 거래 계정입니다. LogOn 후 메인 클라이언트에서 읽으십시오.
- **Direction**: *buy* 또는 *sell*.
- **Quantity**: 시장의 기본 단위로 표현된 거래 규모.
- **BidPrice / OfferPrice**: 주문이 제출될 때 클라이언트가 관찰한 가격입니다(Forex.com은 이를 사용하여 오래된 시세를 감지합니다).
- **AuditId**: 최신 가격 틱에서의 가격 audit id.
- **PositionMethodId**: netting / hedging 동작(1 = Netting, 2 = Hedging).
- **IfDone**: 부모가 체결될 때 활성화되는 첨부된 주문(스톱, 지정가, 보장 스톱, 추적 스톱) 목록입니다.
- **Close**: 이 주문이 체결될 때 닫을 주문 id의 배열(부분 종료).
- **Reference**: 호출자가 제공한 클라이언트 참조로, 주문 업데이트에서 다시 에코됩니다.

Sample

take-profit 및 stop-loss가 첨부된 시장가로 1000 EUR/USD 매수

```
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

이러한 엔드포인트는 계정 상태와 주문 기록을 읽습니다. 활성 세션이 필요합니다([Connect REST API](#) 참조).

Endpoints

Method	매개변수	설명
GetClientAndTradingAccount		클라이언트 계정과 인증된 사용자에게 연결된 거래 계정 목록을 반환합니다.
ListOpenPositions	TradingAccountId	거래 계정의 현재 미결제 포지션을 모두 반환합니다.
ListActiveStopLimitOrders	TradingAccountId	모든 보류 중인 stop 및 limit 주문을 반환합니다.
GetOrder	OrderId	fill 기록 및 연결된 IfDone 주문을 포함하여 id로 단일 주문을 반환합니다.
ListTradeHistory	TradingAccountId, MaxResults	계정의 거래 주문 기록을 가장 최근 것부터 반환합니다.
ListStopLimitOrderHistory	TradingAccountId, MaxResults	계정의 stop-limit 주문 내역을 최신순으로 반환합니다.
SimulateTrade	JSON / scalar / order-object	제출하지 않고 서버 validator를 통해 trade order를 실행합니다. 사전 비행 margin 및 price-tolerance 검사에 유용합니다. NewTradeOrder와 동일한 3가지 오버로드 계층을 수락합니다.

Retired Endpoints

공식 Forex.com / StoneX / CityIndex 참조에는 라이브 서버가 더 이상 제공하지 않는 몇 가지 엔드포인트가 여전히 문서화되어 있습니다. 이들은 의도적으로 구성 요소에 의해 노출되지 않습니다.

- **GetSystemLookup** — 참조는 일반 조회 엔드포인트를 설명하지만 서버는 알려진 모든 조회 키에 대해 HTTP 404로 응답합니다. 대신 **GetClientAndTradingAccount** 및 **ListCfdMarkets**를 사용하십시오.
- **/news/headlines/*** — news-feed 계열은 여전히 참조에 나열되어 있지만 라이브 서버에서 HTTP 500을 반환합니다. Trading API를 통해 사용할 수 있는 대체 항목이 없습니다. 뉴스는 웹 프론트엔드를 통해서만 제공됩니다.

Sample

미체결 포지션과 최근 50개의 거래를 나열합니다

```
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

정적 시장 메타데이터 및 과거 가격입니다. 이 엔드포인트는 거래 엔드포인트 및 스트리밍 어댑터에 공급되는 **MarketId** 값을 검색하는 데 사용됩니다.

Endpoints

Method	매개변수	설명
GetMarketInformation	MarketId	전체 시장 설명자(심볼, 소수 자릿수, pip 크기, 마진 계수, 거래 시간)를 반환합니다.
ListCfdMarkets	SearchByMarketName, ClientAccountId, MaxResults	이름 접두사와 일치하는 CFD 시장을 나열합니다.
FullSearchWithTags	Query, Tags, MaxResults	선택적 태그 필터(FX, Indices, Shares, Commodities)를 사용하여 모든 시장에서 자유 텍스트 검색을 수행합니다.
GetPriceBars	MarketId, Interval, Span, PriceBars	과거 OHLC bars. Interval은 MINUTE, HOUR, DAY, WEEK 중 하나입니다. Span은 bar당 interval 수입니다.
GetPriceTicks	MarketId, PriceTicks	시장의 가장 최근 tick snapshot(bid / ask / mid).

Sample

EUR/USD를 조회하고 마지막 100개의 1분 바를 읽습니다

```
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 / CityIndex는 Lightstreamer를 통해 시장 데이터와 계정 업데이트를 스트리밍합니다. 구성 요소는 WebSocket을 통해 네이티브 Lightstreamer TLCP 2.5 wire 프로토콜로 통신합니다. Lightstreamer 네이티브 라이브러리가 필요하지 않습니다.

Transport

- **WebSocket 서버프로토콜:** *TLCP-2.5.0.lightstreamer.com. Sec-WebSocket-Protocol* 요청 헤더에 광고됩니다.
- **Adapter set:** *STREAMINGALL*. 내부 **Lightstreamer** 서버프로토콜 클라이언트에 구성됩니다. 모든 Forex.com 데이터 어댑터(PRICES, ORDERS, QUOTES, CLIENTACCOUNTMARGIN, TRADEMARGIN)는 이 어댑터 set 아래에 있습니다.
- **Authentication:** REST **LogOn** 호출에서 얻은 세션 토큰은 Lightstreamer 세션을 생성할 때 *LS_password*로 전송되며, 계정 사용자 이름으로 설정된 *LS_user*와 함께 전송됩니다.

Reconnection

AutoReconnect가 활성화되면(기본값) 기본 WebSocket 클라이언트 WatchDog가 네트워크 오류 후 소켓을 다시 엽니다. Lightstreamer 프로토콜 자체는 서버가 클라이언트를 다른 URL로 이동하거나 세션을 다시 바인딩하기를 원할 때 *LOOP* 프레임 알립니다. 구성 요소는 이를 투명하게 처리하며 전체 세션 중단 시에만 **OnForexDisconnect** / **OnForexConnect**를 발생시킵니다.

REST 세션이 만료되면(오랫동안 Ping이 없음) 스트리밍 서버가 세션을 닫습니다. 구성 요소는 저장된 자격 증명을 사용하여 **LogOn**을 다시 호출하고, **SessionToken**을 새로 고치고, 새 토큰으로 Lightstreamer 세션을 다시 시작합니다.

Sample

스트리밍 세션 열기

```
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 | 스트리밍 어댑터 구독

스트리밍 데이터는 *STREAMINGALL* 어댑터 세트 아래의 다섯 개의 Lightstreamer 데이터 어댑터를 통해 전달됩니다. 각 어댑터는 자체 항목 명명 규칙과 구독 모드를 가집니다 — 스냅샷 스타일 상태(현재 가격, 현재 계정 마진)의 경우 **MERGE**, 이벤트 스트림 채널(주문 업데이트, 포지션 업데이트, quote 이벤트)의 경우 **RAW**입니다.

어댑터

DataAdapter	Item	Mode	Event	Fields
PRICES	ID.{MarketId}	MERGE	OnForexPriceTick	MarketId, TickDate, Bid, Offer, Price, 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
CLIENTACCOUNTMARGIN	ID. {ClientAccountId}	MERGE	OnForexAccountMargin	ClientAccountId, Cash, Margin, 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

참고:

- PRICES와 CLIENTACCOUNTMARGIN은 통합된 *ID.{numericId}* 항목 형식을 사용하고 항목당 단일 변경 가능 레코드를 스트리밍합니다(MERGE).
- ORDERS, QUOTES, TRADEMARGIN은 legacy 어댑터별 접두사(*ORDERS.{id}*, *QUOTES.{id}*, *TRADEMARGIN.{id}*)를 사용하며 이벤트의 fire-hose(RAW)를 스트리밍합니다. 한 항목은 로그인한 **ClientAccountId**의 모든 order / quote / trade를 전달합니다. 레코드는 **OrderId** / **QuoteId**로 스스로를 식별합니다.
- TRADEMARGIN 레코드는 레거시 CityIndex SDK의 공식 *TradeMarginDTO*와 일치합니다. **DirectionId**는 숫자 코드로 도착합니다(1 = 매수, 2 = 매도). 라이브러리는 또한 **Direction**을 문자열 "buy" / "sell"로 노출합니다. **OpenPrice**는 거래 진입 가격(서버 필드 *PriceTakenAt*)이고, **CurrentPrice**는 최신 평가 가격(*PriceCalculatedAt*)이며, **OpenTradeEquity**는 계정 통화로 표시된 변동 손익(*OTEConverted*)입니다.
- ORDERS **Direction** 필드는 서버가 게시한 숫자 코드입니다(1 / 2). **Status**는 문자열입니다(예: "Accepted", "Open", "Cancelled"). **OrderType**은 문자열입니다("Market" / "Stop" / "Limit").
- TRADEMARGIN 피드는 거래별 **TradingAccountId**, **MarketName** 또는 거래별 타임스탬프를 게시하지 않습니다. 하나의 항목이 로그인한 **ClientAccountId**에 대한 모든 포지션을 전달합니다. 이러한 필드는

REST_API.ListOpenPositions를 호출하십시오. ORDERS 피드는 **TriggerPrice**나 **Applicability**를 게시하지 않습니다. 이러한 항목이 필요할 때는 **REST_API.GetOrder**를 호출하십시오.

헬퍼 메서드

기본 클라이언트는 Lightstreamer SUB / UNSUB 배관을 래핑합니다. 일반적인 것은 다음과 같습니다:

- **WatchMarket(aMarketId)**: 하나의 market에 대해 PRICES를 구독합니다. 업데이트는 **OnForexPriceTick**에서 도착합니다.
- **UnwatchMarket(aMarketId)**: subscription을 제거합니다.
- **WatchAccount**: 로그인한 계정에 대해 ORDERS, CLIENTACCOUNTMARGIN 및 TRADEMARGIN을 구독합니다. 업데이트는 일치하는 이벤트로 도착합니다.
- **WatchQuotes**: request-for-quote 워크플로를 위해 QUOTES adapter를 구독합니다.

구독 수명 주기 이벤트

메인 클라이언트는 기본 TLCP SUBOK / UNSUB 프레임의 미러링하는 두 가지 수명 주기 이벤트를 발생시킵니다:

- **OnForexSubscribe(Sender, aAdapter, aGroup, aItems, aFields)**: 서버가 모든 Lightstreamer 구독을 확인할 때 발생합니다. *aAdapter*는 데이터 adapter 이름(PRICES, ORDERS, QUOTES, CLIENTACCOUNTMARGIN, TRADEMARGIN)이고, *aGroup*은 item 그룹(예: *ID.<ClientAccountId>* 또는 *ORDERS.<ClientAccountId>*)이며, *aItems*와 *aFields*는 서버가 SUBOK / SUBCMD 프레임에서 보고한 개수입니다.
- **OnForexUnsubscribe(Sender, aAdapter, aGroup)**: **UnwatchMarket**을 통해, 스트리밍 연결 해제 시, 또는 서버가 UNSUB 프레임의 보낼 때 등 구독이 종료될 때 발생합니다.

Sample

EUR/USD 가격 틱 구독

```
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

Blockchain WebSocket API를 사용하면 개발자가 새 트랜잭션 및 블록에 대한 실시간 알림을 받을 수 있습니다.

WebSocket이 열리면 채널을 구독할 수 있습니다:

- **SubscribeTransactions:** 모든 새 bitcoin 트랜잭션에 대한 알림을 구독합니다.
- **UnsubscribeTransactions:** 모든 새 bitcoin 트랜잭션에 대한 알림 구독을 취소합니다.
- **SubscribeAddress:** 특정 bitcoin 주소에 대한 새 거래를 수신합니다.
- **UnSubscribeAddress:** 특정 bitcoin 주소에 대한 새 트랜잭션 수신을 중지합니다.

트랜잭션은 **OnNewTransaction** 이벤트로 수신됩니다:

```
{
  "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:** 새 블록이 발견되면 알림을 받습니다. 참고: 체인이 분할되면 특정 블록 높이에 대해 둘 이상의 알림을 받을 수 있습니다.
- **UnSubscribeBlocks:** 새 블록이 발견될 때 알림 수신을 중지합니다. 참고: 체인이 분할되면 특정 블록 높이에 대해 둘 이상의 알림을 받게 됩니다.

블록은 **OnNewBlock** 이벤트로 수신됩니다:

```
{
  "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": "94e51495e0e8a0c3b78dac1220b2f35ceda8799b0a20cfa68601ed28126fcc2",  
    "version": 1,  
    "time": 1331301261,  
    "bits": 436942092,  
    "nonce": 758889471  
  }  
}
```

API Cex

Cex

WebSocket API를 사용하면 추가 요청을 보내지 않고도 실시간 알림을 받을 수 있어, 거래소에서 데이터를 더 빠르게 얻을 수 있는 방법입니다.

Cex 구성 요소에는 Cex라는 속성이 있으며, 여기에 Cex에서 제공한 API 키를 입력하여 계정 데이터에 액세스할 수 있습니다.

Message encoding

- 모든 메시지는 JSON 형식으로 인코딩됩니다.
- 가격은 클라이언트 측에서 JSON 구문 분석 시 반올림 오류를 피하기 위해 문자열로 표시됩니다.
- WebSocket 프레임의 압축은 서버에서 지원되지 않습니다.
- 시간은 초 단위의 정수 UNIX 타임스탬프로 표시됩니다.

Authentication

CEX.IO WebSocket 데이터에 접근하려면 권한이 부여되어야 합니다.

- CEX.IO 계정에 로그인합니다.
- <https://cex.io/trade/profile#/api> 페이지로 이동하십시오.
- 필요한 권한 유형을 선택하십시오.
- "Generate Key" 버튼을 클릭하고 시크릿 키를 저장하십시오. 활성화 후에는 액세스할 수 없게 됩니다.
- 키를 활성화합니다.

Connectivity

- 연결된 Socket이 15초 동안 비활성 상태이면, CEX.IO 서버가 PING 메시지를 보냅니다.
- 서버만 PING 요청의 Initiator가 될 수 있습니다.
- 서버는 인증된 사용자에게만 ping을 보냅니다.
- 사용자는 PONG 메시지로 응답해야 합니다. 그렇지 않으면 WebSocket의 연결이 끊깁니다. 이는 라이브러리에서 자동으로 처리됩니다.
- 인증된 사용자의 경우, 서버로부터 15초 이내에 알림이나 ping이 없으면, 연결 및 인증을 보장하기 위해 'ticker' 또는 'get-balance'와 같은 요청을 보내고 응답을 받는 것이 더 안전합니다.

Public Channels

이 채널은 인증이 필요하지 않습니다. 서버의 응답은 OnCexMessage 이벤트를 통해 수신됩니다.

- **SubscribeTickers:** 모든 페어에서 이루어진 거래의 가격만 포함하는 티커 피드입니다(사용 중단됨)

```
{
  "e": "tick",
  "data": {
    "symbol1": "BTC",
    "symbol2": "USD",
    "price": "428.0123"
  }
}
```

- **SubscribeChart:** Open, High, Low, Close, Volume 숫자가 있는 OHLCV 차트 피드(더 이상 사용되지 않음)

```
{
  'e': 'ohlcv24',
  'pair': 'BTC:USD',
  'data': [
    '418.2936',
    '420.277',
    '412.09',
    '416.9778',
    '201451078368'
  ]
}
```

- **Subscribe Pair: Market Depth** 피드(더 이상 사용되지 않음)

```
{
  '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: Order Book** 피드(사용 중단됨)

```
{
  '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 채널

이러한 채널에 액세스하려면 먼저 Authenticate 메서드를 호출하십시오. 서버로부터의 응답은 OnCexMessage 이벤트를 통해 수신됩니다.

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

CEX.IO 시스템에서 주문은 거래 엔진 또는 아카이브 데이터베이스에 존재할 수 있습니다. 주문이 완료/취소되었지만 아직 아카이브 데이터베이스로 이동되지 않은 시간 기간(~2초 이상)이 있을 수 있습니다. 즉, archived-orders/open-orders 호출을 사용하여 볼 수 없습니다. 이 호출을 통해 어떤 경우에도 주문 정보를 얻을 수 있습니다. 응답은 주문 위치에 따라 다른 형식을 가질 수 있습니다.

```
{
  "e": "get-order",
  "data": {
    "user": "XXX",
    "type": "buy",
    "symbol1": "BTC",
    "symbol2": "USD",
    "amount": "0.02000000",
    "remains": "0.02000000",
    "price": "50.75",
    "time": 1450214742160,
    "tradingFeeStrategy": "fixedFee",
    "tradingFeeBuy": "5",
    "tradingFeeSell": "5",
    "tradingFeeUserVolumeAmount": "nil",
    "a:USD:c": "1.08",
    "a:USD:s": "1.08",
    "a:USD:d": "0.00",
    "status": "a",
    "orderId": "5582060"
  },
  "oid": "1450214742135_10_get-order",
  "ok": "ok"
}
```

CancelOrderRequest

```
{
  "e": "cancel-order",
  "data": {
    "order_id": "2477098",
    "time": 1443468122895
  },
  "oid": "1435927928274_12_cancel-order",
  "ok": "ok"
}
```

GetArchivedOrders

```
{
  "e": "archived-orders",
  "data": [
    {
      "type": "buy",
      "symbol1": "BTC",
      "symbol2": "USD",
      "amount": 0,
      "amount2": 5000,
      "remains": 0,
      "time": "2015-04-17T10:46:27.971Z",
      "tradingFeeBuy": "2",
      "tradingFeeSell": "2",
      "ta:USD": "49.00",
      "fa:USD": "0.98",
      "orderId": "2340298",
      "status": "d",
      "a:BTC:cds": "0.18151851",
      "a:USD:cds": "50.00",
      "f:USD:cds": "0.98"
    }
  ]
}
```

```

},
{
  "type": "buy",
  "symbol1": "BTC",
  "symbol2": "USD",
  "amount": 0,
  "amount2": 10000,
  "remains": 0,
  "time": "2015-04-08T15:46:04.651Z",
  "tradingFeeBuy": "2.99",
  "tradingFeeSell": "2.99",
  "ta:USD": "97.08",
  "fa:USD": "2.91",
  "orderId": "2265315",
  "status": "d",
  "a:BTC:cds": "0.39869578",
  "a:USD:cds": "100.00",
  "f:USD:cds": "2.91"
}
],
"oid": "1435927928274    15_archived-orders",
"ok": "ok"
}

```

OpenPosition

```

{
  "e": "open-position",
  "oid": "1435927928274_7_open-position",
  "data": {
    'amount': '1',
    'symbol': 'BTC',
    "pair": [
      "BTC",
      "USD"
    ],
    'leverage': '2',
    'ptype': 'long',
    'anySlippage': 'true',
    'eoprice': '650.3232',
    'stopLossPrice': '600.3232'
  }
}

```

GetPosition

```

{
  "e": "get_position",
  "ok": "ok",
  "data": {
    "user": "ud100036721",
    "pair": "BTC:USD",
    "amount": "1.00000000",
    "symbol": "BTC",
    "msymbol": "USD",
    "omamount": "1528.77",
    "lsymbol": "USD",
    "lamount": "3057.53",
    "slamount": "3380.11",
    "leverage": "3",
    "stopLossPrice": "3380.1031",
    "dfl": "3380.10310000",
    "flPrice": "3057.53333333",
    "otime": 1513002370342,
    "psymbol": "BTC",
    "ptype": "long",
    "ofee": "10",
    "pfee": "10",
    "cfee": "10",
    "tfeeAmount": "152.88",
    "rinterval": "14400000",
    "okind": "Manual",
    "a:BTC:c": "1.00000000",
    "a:BTC:s": "1.00000000",
    "oorder": "89101551",
    "pamount": "1.00000000",
    "lremains": "3057.53",

```

```

    "slremains": "3380.11",
    "oprice": "4586.3000",
    "status": "a",
    "id": "125531",
    "a:USD:cds": "4739.18"
  }
}

```

GetOpenPositions

```

{
  'e': 'open_positions',
  "oid": "1435927928256_7_open-positions",
  'ok': 'ok',
  'data': [
    {
      'user': 'ud100036721',
      'id': '104102',
      'otime': 1475602208467,
      'symbol': 'BTC',
      'amount': '1.00000000',
      'leverage': '2',
      'ptype': 'long',
      'psymbol': 'BTC',
      'msymbol': 'USD',
      'lsymbol': 'USD',
      'pair': 'BTC:USD',
      'oprice': '607.5000',
      'stopLossPrice': '520.3232',
      'ofee': '1',
      'pfee': '3',
      'cfee': '4',
      'tfeeAmount': '3.04',
      'pamount': '1.00000000',
      'omamount': '303.75',
      'lamount': '303.75',
      'oorder': '34106774',
      'rinterval': '14400000',
      'dfl': '520.32320000',
      'slamount': '520.33',
      'slremains': '520.33',
      'lremains': '303.75',
      'flPrice': '303.75000000',
      'a:BTC:c': '1.00000000',
      'a:BTC:s': '1.00000000',
      'a:USD:cds': '610.54',
    },
    ...
  ]
}

```

ClosePosition

```

{
  'e': 'close_position',
  "oid": "1435927928364_7_close-position",
  'ok': 'ok',
  'data': {
    'id': 104034,
    'ctime': 1475484981063,
    'ptype': 'long',
    'msymbol': 'USD',
    'pair': {
      'symbol1': 'BTC',
      'symbol2': 'USD'
    },
    'price': '607.1700',
    'profit': '-12.48',
  }
}

```

API Cex Plus

Cex Plus

지원되는 API

- [WebSockets API](#): 공개 websocket 서버에 연결하고 실시간 시장 데이터 업데이트를 제공합니다.

WebSockets API

WebSocket은 TCP 기반의 전이중 통신 프로토콜입니다. 전이중이란 양측이 동일한 통신 채널을 사용하여 서로에게 비동기적으로 메시지를 보낼 수 있음을 의미합니다. 이 섹션은 Exchange Plus와 클라이언트가 서로 보내야 하는 메시지를 설명합니다. 모든 메시지는 유효한 JSON 객체여야 합니다.

WebSocket API는 주로 REST API를 사용하여 사용할 수 없거나 쉽지 않은 정보를 얻거나 작업을 수행하는 데 사용됩니다. 그러나 일부 요청이나 작업은 REST API와 WebSocket API 모두에서 가능합니다. Exchange Plus는 클라이언트가 이전에 보낸 요청에 대한 응답으로, 또는 (클라이언트의 사전 요청 없이) 일부 이벤트에 대한 알림으로 클라이언트에 메시지를 보냅니다.

Public API Calls

공개 API 속도 제한은 DDoS 공격으로부터 시스템을 보호하고 모든 클라이언트가 Exchange Plus API 엔드포인트에 동일한 수준의 안정적인 액세스를 가질 수 있도록 암시적으로 적용됩니다. 공개 요청은 공개 API 요청이 이루어지는 IP 주소로 제한됩니다. 요청 제한은 각 공개 API 호출과 관련된 비용에서 결정됩니다. 기본적으로 각 공개 요청의 비용은 1포인트이지만 일부 특정 요청의 경우 이 비용이 더 높을 수 있습니다. 각 메서드의 사양에서 최신 요청 속도 제한 비용 정보를 확인하십시오.

Exchange Plus는 Public API 호출을 분당 최대 100 포인트로 제한하며, 각 Public API 호출에는 비용이 있다는 점을 고려합니다(아래 참조). 요청 속도 제한에 도달하면 Exchange Plus는 오류로 응답하고, Client에 disconnected 이벤트를 보내며, 그 후 WS 연결을 닫습니다. Exchange Plus는 다음 달력 분(minute)부터 Client에 대한 서비스를 계속합니다. 다음 예제에서 요청 카운터는 11:02:00.000에 재설정됩니다.

메서드	설명
GetTicker	이 메서드는 현재 가격, 24시간 가격 및 거래량 변동, 마지막 거래 이벤트 등 특정 자산에 대한 Ticker의 현재 정보를 얻도록 설계되었습니다.
GetOrderBook	이 메서드를 사용하면 클라이언트가 특정 trading pair에 대한 현재 order book 스냅샷을 수신할 수 있습니다.
GetCandles	Candles 메서드를 사용하여 클라이언트는 다양한 해상도와 데이터 유형의 과거 OHLCV 캔들을 받을 수 있습니다. 클라이언트는 추가 시간 프레임과 제한 필터를 표시하여 응답을 클라이언트의 요구 사항에 더 정확하게 맞출 수 있습니다.
GetTradeHistory	이 메서드를 통해 클라이언트는 요청된 거래 쌍에서 발생한 거래에 대한 과거 데이터를 얻을 수 있습니다. 클라이언트는 timeframe 기간, tradeIds 범위, side 등의 추가 필터 매개변수로 Trade History 요청을 보완하여 요청 매개변수와 일치하는 거래를 받을 수 있습니다.
GetServerTime	이 메서드는 Exchange Plus 서버의 현재 시간을 가져오는 데 사용됩니다. 서버 시간과 동기화해야 하는 애플리케이션에 유용할 수 있습니다.
GetPairsInfo	Pair Info 메서드는 클라이언트가 지원되는 모든 거래 쌍의 매개변수를 받을 수 있게 합니다.
GetCurrenciesInfo	Currencies Info 메서드는 클라이언트가 Exchange Plus에 구성된 모든 통화의 매개변수와, Exchange Plus와 CEX.IO Wallet 간의 입출금 가용성을 수신할 수 있도록 합니다.
GetProcessingInfo	이 요청을 통해 클라이언트는 각 지원되는 암호화폐에 대해 외부 지갑에서 입금하고 외부 지갑으로 출금하기 위한 사용 가능한 옵션에 대한 상세 정보를 받을 수 있으며, 여기에는 암호화폐 이름과 입금\출금에 사용 가능한 블록체인이 포함됩니다. 또한 각 지원되는 블록체인에 대해 해당 블록체인의 암호화

	폐 유형, 현재 입금\출금 가능 여부, 입금\출금 최소 금액, 외부 출금 수수료가 표시됩니다. 처리 정보는 클라이언트가 입금 주소를 받고 특정 블록체인을 통해 외부 출금을 시작하기 위해 원하는 블록체인을 선택하는 데 더 유연하게 해주므로, 클라이언트가 CEX.IO 생태계로 또는 생태계에서 자신의 암호화 자산을 전송하는 더 편리한 방법을 사용할 수 있습니다.
SubscribeOrderBook	WebSocket을 통해 구독하면 클라이언트는 요청한 거래 쌍에 대한 호가창 피드를 구독할 수 있습니다. Order Book Subscribe 요청에 대한 응답으로 클라이언트는 표시된 seqId 번호와 함께 요청한 쌍의 현재(초기) 호가창 스냅샷을 받습니다. 호가창에 대한 후속 업데이트를 추적하려면 클라이언트는 WebSocket을 통해 "order_book_increment" 메시지를 구독해야 하며, 이 메시지에는 거래 쌍 이름, seqId 번호, 매수 및 매도 가격 레벨 델타가 포함됩니다.
UnSubscribeOrderBook	호가창 채널에서 구독을 취소합니다.
SubscribeTrade	Trade Subscribe 메서드를 사용하면 클라이언트는 WebSocket을 통해 요청된 거래 페어에서 발생하는 거래 이벤트의 라이브 피드를 구독할 수 있습니다. Trade Subscribe 요청에 대한 응답으로 클라이언트는 거래 구독의 고유 식별자를 받게 되며, 이는 클라이언트가 더 이상 거래 구독이 필요하지 않을 때 구독 취소에 추가로 사용해야 합니다. 클라이언트는 초기 및 주기적 Trade History 스냅샷과 요청된 거래 페어의 라이브 거래 이벤트를 수신하기 위해 WebSocket을 통해 "tradeHistorySnapshot" 및 "tradeUpdate" 메시지를 구독해야 합니다.
UnSubscribeTrade	trade 채널에서 구독을 취소합니다.

예제: BTC-USD 쌍의 최신 ticker 가져오기

```
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;
}
```

Private API Calls

Exchange Plus는 API 키를 사용하여 Private API에 대한 액세스를 허용합니다.

클라이언트는 API Keys Management Profile 섹션의 Exchange Plus Web Terminal을 통해 api key를 생성, 구성 및 관리하고, permission 수준을 설정하고, API key에 대한 허용된 IP 등을 설정할 수 있습니다.

API Keys 제한: 기본적으로 클라이언트는 최대 5개의 API Key를 가질 수 있습니다.

API 키를 사용하는 동안 특정 기능에 대한 액세스를 제한하려면 각 API 키에 대해 특정 권한 집합을 정의해야 합니다. 정의된 권한 집합은 필요한 경우 나중에 추가로 편집할 수 있습니다.

API Key에 대해 다음 권한 수준을 사용할 수 있습니다:

- **Read:** 계정 관련 데이터 조회, 보고서 수신, 시장 데이터 구독 등에 대한 permission 수준입니다.
- **Trade:** 계정을 대신하여 주문을 넣고 취소할 수 있는 권한 수준입니다.

- **Funds Internal:** CEX.IO Exchange Plus Portfolio의 계정 간(하위 계정 간 또는 메인 계정과 하위 계정 간) 자금 이체를 허용하는 권한 수준입니다.
- **Funds Wallet:** CEX.IO Exchange Plus Portfolio 계정(메인 계정 및 하위 계정)에서 CEX.IO Wallet으로, 그리고 그 반대로 자금을 이체할 수 있는 권한 수준입니다.

메서드	설명
GetCurrentFee	이 메서드는 Client의 최신 30일 거래량과 요일을 고려하여 특정 시점의 현재 수수료를 나타냅니다(수수료는 예를 들어 주말에 다를 수 있음).
GetFeeStrategy	Fee Strategy는 클라이언트의 거래량, 요일, 페어, 페어 그룹 등을 고려하여 클라이언트에게 적용될 수 있는 모든 수수료 옵션을 반환합니다. 이 메서드는 클라이언트에게 적용될 수 있는 모든 가능한 거래 수수료 값을 포함하는 일반 수수료 전략에 대한 정보를 제공합니다. 클라이언트의 현재 30일 거래량을 기반으로 현재 거래 수수료를 받으려면 클라이언트는 [Current Fee] 메서드를 사용해야 합니다. 현재 30일 거래량을 받으려면 클라이언트는 [Volume] 메서드를 사용해야 합니다.
GetVolume	이 요청은 클라이언트가 지난 30일 동안의 거래량을 USD 등가로 수신할 수 있도록 합니다.
CreateAccount	이 요청을 통해 클라이언트는 새 하위 계정을 생성할 수 있습니다. 기본적으로 클라이언트는 메인 계정을 포함하여 최대 5개의 하위 계정을 가질 수 있습니다.
GetAccountStatus	Account Status V3 메서드를 사용하면 클라이언트는 각 하위 계정 및 통화에 대해 현재 잔액과 변환된 통화(기본값 "USD")로의 표시 등가물, 미결제(할성) 주문에 잠긴 금액을 확인할 수 있습니다. 클라이언트에게 거래 수수료 잔액을 사용할 수 있는 경우, 응답에는 프로모션 이름, 통화 이름, 총 잔액 및 Trading Fee Balance의 이 프로모션 만료 날짜와 같은 일반 거래 수수료 잔액 데이터도 포함됩니다. 하위 계정의 미결제(할성) 주문에 잠긴 잔액 금액만큼 줄어든 현재 하위 계정 잔액으로 사용 가능한 거래 잔액을 추적하는 것은 클라이언트의 책임입니다.
GetOrders	이 요청을 통해 클라이언트는 자신의 주문에 대한 정보를 찾을 수 있습니다.
NewOrder	클라이언트는 Do My New Order Request를 사용하여 WebSocket API를 통해 새 주문을 할 수 있습니다. 이 요청에 대한 응답과 함께, 요청이 성공하면 Exchange Plus는 Account Event 및 Execution Report 메시지를 클라이언트에 보냅니다. Response 메시지는 응답을 보내는 시점에 시스템에서 사용할 수 있는 주문의 최신 상태를 나타냅니다. 클라이언트가 Do My New Order Request에 대한 Response 메시지를 받지 못한 경우, 클라이언트는 clientOrderId 매개변수와 함께 Get My Orders Request를 사용하여 주문의 현재 상태를 쿼리할 수 있습니다. 새 주문에 대한 요청을 보낼 때, 클라이언트 측의 특정 새 주문 요청에 해당하는 clientOrderId 매개변수를 사용하는 것이 강력히 권장됩니다. Exchange Plus는 동일한 clientOrderId로 주문을 여러 번 하는 것을 방지합니다. 동일한 clientOrderId 및 기타 동일한 주문 매개변수를 가진 둘 이상의 새 주문이 식별되면, Exchange Plus는 첫 번째 주문만 처리하고 동일한 매개변수를 가진 두 번째 및 이후의 새 주문 요청에 대한 응답으로 해당 주문의 상태를 클라이언트에 반환합니다. 동일한 clientOrderId이지만 다른 주문 매개변수를 가진 둘 이상의 새 주문이 식별되면, Exchange Plus는 첫 번째 주문만 처리하고 동일한 clientOrderId이지만 다른 주문 매개변수를 가진 두 번째 및 이후의 새 주문 요청을 거부합니다.
NewMarketOrder	새 시장가 주문을 넣습니다.
NewLimitOrder	새 limit 주문을 합니다.
CancelOrder	클라이언트는 주문을 취소할 수 있습니다. 이 요청에 대한 응답과 함께, 이 요청이 성공하면 Exchange Plus는 클라이언트에 Account Event 및 Execution Report 메시지를 보냅니다. 또한 주문 취소 요청이 거부되면, Exchange Plus는 Order Cancellation Rejection 메시지를 보냅니다.
CancelAllOrders	클라이언트는 WebSocket API를 통해 모든 미체결 주문을 취소할 수 있습니다. 이 요청에 대한 응답과 함께 Exchange Plus는 모든 미체결 주문에 대한 취소 프로세스를 시작하고 해당 Account Event 및 Execution Report 메시지를 클라이언트에 보냅니다.
GetTransactionHistory	이 요청을 통해 클라이언트는 자신의 금융 거래(입금, 출금, 내부 이체, 수수료 또는 거래)를 확인할 수 있습니다.
GetFundingHistory	이 요청을 통해 클라이언트는 자신의 입금 및 출금 거래를 찾을 수 있습니다.
InternalTransfer	클라이언트는 자신의 하위 계정 간 또는 메인 계정과 하위 계정 간에 자금 이체를 요청할 수 있습니다. Exchange Plus는 계정 간 자금 이체에 대해 클라이언트에게 수수료를 부과하지 않습니다.

	이 요청에 대한 응답과 함께, 요청이 성공하면 Exchange Plus는 클라이언트에게 Account Event 메시지를 보냅니다.
GetDepositAddress	이 메서드는 클라이언트가 암호화폐를 입금하기 위한 crypto 주소를 받는 데 사용할 수 있습니다. 입금 주소는 메인 및 하위 계정에 대해 생성할 수 있습니다. 입금 주소 생성에 사용할 수 있는 블록체인 목록은 Get Processing Info 요청을 통해 클라이언트가 받을 수 있습니다.
FundsDepositFromWallet	클라이언트는 CEX.IO Wallet에서 Exchange Plus 계정으로 자금을 입금할 수 있습니다. 시스템은 동일한 clientTxId를 가진 여러 입금 요청의 처리를 방지합니다. 동일한 clientTxId를 가진 여러 입금 요청이 수신되면 시스템은 첫 번째 입금 요청만 처리하고 동일한 clientTxId를 가진 두 번째 및 후속 입금 요청을 거부합니다.
FundsWithdrawalToWallet	클라이언트는 Exchange Plus 계정에서 CEX.IO Wallet으로 자금을 인출할 수 있습니다. 시스템은 동일한 clientTxId로 여러 인출 요청이 발생하는 것을 방지합니다. 동일한 clientTxId로 여러 인출 요청이 수신되면, 시스템은 첫 번째 인출 요청만 처리하고 동일한 clientTxId를 가진 두 번째 및 후속 인출 요청을 거부합니다.
GetWalletBalance	CEX.IO Wallet 잔액 정보를 검색합니다.
SubscribeAccountEvents	실시간 계정 이벤트 알림(잔액 변경, 주문 체결)을 구독합니다.
UnSubscribeAccountEvents	계정 이벤트 알림 구독을 해제합니다.

예제: 주문 가져오기.

```
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

참고: API v10부터 Discord API 도메인이 discordapp.com에서 discord.com으로 변경되었습니다. 구성 요소는 API 버전 10과 새 도메인을 사용하도록 업데이트되었습니다.

Gateway는 안전한 WebSockets를 통한 Discord의 실시간 통신 형태입니다. 클라이언트는 연결된 gateway를 통해 이벤트와 데이터를 수신하고 REST API를 통해 데이터를 보냅니다.

권한 부여

먼저 새 Bot을 생성하고 API를 통해 인증하는 데 사용할 Bot Token을 복사해야 합니다. 그런 다음 이 토큰을 API Component에 설정하십시오.

```
TsgcWSAPI_Discord1->DiscordOptions->BotOptions->Token = "...bot token here...";
```

Intents

상태 저장 애플리케이션을 유지하는 것은 특히 대규모로 처리해야 하는 데이터 양 측면에서 어려울 수 있습니다. Gateway Intents는 그러한 계산 부담을 줄이는 데 도움이 되는 시스템입니다.

gateway에 식별할 때, intents 매개변수를 지정하여 사전 정의된 "intents", 즉 Discord가 정의한 이벤트 그룹을 조건부로 구독할 수 있습니다. 특정 intent를 지정하지 않으면, 해당 그룹으로 묶인 gateway 이벤트를 받지 못합니다. 유효한 intent는 다음과 같습니다(0 값은 모든 이벤트가 수신됨을 의미):

```
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

하트비트는 구성 요소가 자동으로 처리하므로 신경 쓸 필요가 없습니다. 클라이언트가 서버에 연결되면 서버가 하트비트 간격이 있는 HELLO 응답을 보내고, 구성 요소가 응답을 읽고 x초마다 ping을 보내도록 하트비트를 자동으로 조정합니다. 때때로 서버가 클라이언트에 ping을 보낼 수 있으며, 이것도 클라이언트가 자동으로 처리합니다.

Connection Ready

연결이 준비되면, 즉 성공적인 로그인과 서버의 권한 부여 후에 **OnDiscordReady** 이벤트가 발생하며, 그 후에 서버로부터 업데이트를 수신하기 시작할 수 있습니다.

Connection Resume

연결이 예기치 않게 닫히면 클라이언트가 다시 연결하려고 할 때 **OnDiscordBeforeReconnect** 이벤트를 호출합니다. 구성 요소는 성공적인 재개에 필요한 모든 데이터를 자동으로 저장하지만, 필요한 경우 매개변수를 변경할 수 있습니다. 다시 연결하지 않고 새로운 깨끗한 세션을 시작하려면 Reconnect를 False로 설정하기만 하면 됩니다.

세션이 재개되면 **OnDiscordResumed** 이벤트가 발생합니다. 새 세션인 경우 **OnDiscordReady**가 발생합니다.

Dispatch Events

이벤트는 **OnDiscordDispatch**를 통해 디스패치되므로 여기에서 서버가 클라이언트에 보낸 이벤트를 읽을 수 있습니다.

```
void OnDiscordDispatch(TObject *Sender, const string aEvent, const string RawData)
{
    DoLog("#discord dispatch: " + aEvent + " " + RawData);
}
```

aEvent 매개변수는 이벤트 이름을 포함합니다.

RawData는 전체 JSON 메시지를 포함합니다.

HTTP Requests

길드, 사용자에게 대한 정보를 요청하거나 데이터를 업데이트하려면... gateway websocket 메시지를 사용하는 대신 Discord는 HTTP 요청 사용을 요구합니다. HTTP 요청을 만들기 위해 사용 가능한 모든 메서드를 아래에서 확인하십시오:

```
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;
```

예제: 현재 사용자 정보 가져오기

```
result = GET_Request("/users/@me");
```

서버의 샘플 응답:

```
{
  "id": "637423922035480852",
  "username": "test",
  "avatar": null,
  "discriminator": "5125",
  "bot": true,
  "email": null,
  "verified": true,
  "locale": "en-US",
  "mfa_enabled": false,
  "flags": 0
}
```

API | OpenAI

OpenAI Realtime API는 음성 대 음성 대화 경험 및 실시간 전사를 포함한 저지연 멀티모달 상호 작용을 가능하게 합니다.

TsgcWSAPI_OpenAI 구성 요소는 RealTime OpenAI API를 구현합니다.

구성

Conversation 또는 **Transcription**을 선택하려면 **method** 속성 을 사용하십시오. 현재 Transcription 모드만 지원됩니다.

InputAudio 속성을 사용하면 다음 데이터를 사용자 정의할 수 있습니다:

- **Language:** 예: 영어(value = 'en').
- **Model:** 사용할 모델, 예: whisper-1
- **Prompt:** 모델에 지침을 제공하는 선택적 프롬프트입니다.

OpenAI

OpenAI API는 인증을 위해 API 키를 사용합니다. 요청에서 사용할 API 키를 검색하려면 [API Keys](#) 페이지를 방문하십시오. API 키는 비밀이라는 점을 기억하십시오! 다른 사람과 공유하거나 클라이언트 측 코드(브라우저, 앱)에 노출하지 마십시오. 프로덕션 요청은 API 키를 환경 변수 또는 키 관리 서비스에서 안전하게 로드할 수 있는 자체 백엔드 서버를 통해 라우팅해야 합니다.

이 **API Key**는 구성 요소의 **OpenAIOptions.ApiKey** 속성에 구성되어야 합니다. 선택적으로, 여러 조직에 속한 사용자의 경우 계정이 조직에 속해 있으면 **OpenAIOptions.Organization** 속성에 Organization을 설정할 수 있습니다.

API Key가 구성되면, OpenAI API와 상호작용하기 위해 사용 가능한 함수 목록을 아래에서 확인하십시오.

Azure

클라이언트는 Microsoft Azure OpenAI Services를 지원하므로 Azure 계정을 사용하여 Azure OpenAI API와 상호 작용할 수도 있습니다. 클라이언트를 Azure에서 작동하도록 구성하려면 다음 단계를 따르십시오:

1. 속성 **OpenAIOptions.Provider** = oapvAzure를 구성하십시오
2. ResourceName과 DeploymentId의 값을 설정하십시오(이 값은 Azure Account에서 찾을 수 있습니다).
 1. **OpenAIOptions.AzureOptions.ResourceName** = <your resource name>.
 2. **OpenAIOptions.AzureOptions.DeploymentId** = <your deployment id>.
3. Azure 계정의 API Key를 설정하십시오
 1. **OpenAIOptions.ApiKey** = <azure api key>.

전사 예제

아래에서 openAI API를 사용한 실시간 전사 예제를 찾을 수 있습니다

```
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

MEXC는 안전한 WebSocket 연결을 통해 스트리밍 시장 및 계정 데이터를 노출하는 글로벌 암호화폐 거래소입니다.

WebSocket Spot API는 [Market Streams](#) 및 [User Data Streams](#) 문서에 게시된 사양을 따릅니다. Market 채널은 Protocol Buffers 프레임으로 전달되는 반면 비공개 스트림은 JSON을 사용합니다. **TsgcWSAPI_MEXC** 구성 요소는 연결 수명 주기, 인증 및 listen key 유지 관리를 관리하므로 애플리케이션은 푸시된 데이터 처리에 집중할 수 있습니다.

WebSockets를 통한 스트리밍 외에도 톨킷은 완전한 Spot HTTP API도 노출합니다. **TsgcHTTP_API_MEXC_Spot** 구성 요소(유닛 **sgcHTTP_API_MEXC**)는 모든 공식 REST 엔드포인트를 래핑하므로 애플리케이션이 시장 데이터를 쿼리하거나 서명된 요청으로 거래할 수 있습니다.

구성 요소 구성

기본 WebSocket 엔드포인트는 **wss://wbs-api.mexc.com/ws**입니다. **MEXCUserDataStreams.UserStream**이 활성화되고 유효한 API 키가 제공되면, 구성 요소는 자동으로 listen key를 요청하고 연결 URL에 추가합니다.

TsgcWebSocketClient와 **TsgcWSAPI_MEXC** 구성 요소를 폼에 드롭하고(또는 코드로 생성하고), 클라이언트 인스턴스를 API 구성 요소에 할당한 다음, 클라이언트를 활성화하여 세션을 시작하십시오.

```
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:** **TsgcWebSocketClient** 전송에 대한 참조입니다. 구성 요소는 세션을 유지하기 위해 클라이언트 하트비트(15초)를 활성화합니다.
- **MEXCAPI.ApiKey / ApiSecret:** Spot API 자격 증명입니다. 키는 private 채널에 필요합니다. secret은 listen key 요청에 서명하는 데 필요합니다.
- **MEXCUserDataStreams.UserStream: True**(기본값)인 경우 구성 요소는 연결하기 전에 listen key를 검색하고 10분마다 새로 고칩니다. 공개 피드만 사용하려면 비활성화하십시오.
- **MEXCUserDataStreams.ListenKeyOnDisconnect:** listen key가 활성화된 상태에서 WebSocket이 연결을 끊을 때 실행되는 작업:
 - **blkmxodDeleteListenKey:** REST를 통해 listen key를 취소합니다(기본값).
 - **blkmxodClearListenKey:** listen key를 서버 측에 유지하고 캐시된 값을 지웁니다.
 - **blkmxodDoNothing:** 다음 연결에서 동일한 listen key를 재사용합니다.
- **ListenKey:** REST API가 반환한 활성 listen key를 노출하는 읽기 전용 속성.
- **RawMessages:** 활성화되면 수신 텍스트 프레임이 파싱되지 않고 상속된 WebSocket 이벤트를 통해 일반 JSON으로 전달됩니다.

Events

- **OnConnect / OnDisconnect:** 기본 WebSocket 세션이 설정되거나 종료될 때 발생합니다.
- **OnMEXCMarketStream:** 모든 Protobuf 시장 프레임에 대해 트리거됩니다. 처리기는 **MessageType** 속성이 페이로드(Trade, KLine, DiffDepth, LimitDepth, BookTicker, BookTickerBatch, MiniTicker 또는 MiniTickers)를 식별하는 **TsgcMEXCSpotProtoMessage**를 받습니다. 디코딩된 구조를 검사하려면 **sgcWebSocket_API_MEXC_Proto**의 도우미 클래스를 사용하십시오.
- **OnMEXCResponse:** 구독 확인 및 ping/pong 응답을 JSON 형식으로 수신합니다(**Id, Code, Message**).

- **OnMEXCUserDataStream:** private 계정 메시지(주문, deal, 잔액 업데이트)가 도착할 때 발생합니다. 채널 이름과 raw JSON 페이로드가 제공됩니다.
- **OnMEXCException:** REST 또는 WebSocket 예외(listen key 오류, HTTP 실패, 구문 분석 문제 등)를 캡처합니다.

다음 샘플은 trade 프레임을 디코딩하는 방법을 보여줍니다:

WebSocket 구독 메서드

구성 요소는 각 공개 Spot 채널에 대한 헬퍼를 노출합니다. 데이터 스트리밍을 시작하려면 **Subscribe*** 메서드를 호출하고, 중지하려면 해당 **UnSubscribe***를 호출하십시오.

공개 채널

Method	매개변수	설명
SubscribeTrade	Symbol, Interval (ms)	Protobuf publicdeals 로 전달되는 집계된 거래 체결입니다.
SubscribeKline	Symbol, Interval	구성 가능한 간격(1분에서 1개월)에 대한 Candlestick 업데이트.
SubscribeDiffDepth	Symbol, Interval (ms)	로컬 호가창 유지를 위한 주문 호가창 증분 깊이 델타입니다.
SubscribeBookDepth	Symbol, Levels	요청된 깊이(기본값 5)로 지정된 주문장의 주기적 스냅샷입니다.
SubscribeBookTicker	Symbol, Interval (ms)	단일 거래 쌍에 대한 최적 bid/ask 업데이트.
SubscribeBookTickerBatch	UTC 플래그	여러 심볼에 대한 집계된 book ticker 업데이트.
SubscribeMiniTickers	UTC 플래그	모든 활성 instrument에 대한 롤링 24시간 미니 ticker 통계입니다.
SubscribeMiniTicker	Symbol, UTC flag	단일 심볼에 대한 미니 티커입니다.

프라이빗 채널

Private 토픽에는 유효한 API 키가 필요합니다. 구성 요소는 활성 listen key를 사용하여 구독하고 주기적으로 갱신합니다.

Method	설명
SubscribeAccountUpdate	계정 잔액 및 포지션 변경.
SubscribeAccountDeals	체결된 주문에 대한 실행 보고서입니다.
SubscribeAccountOrders	주문 수명 주기 업데이트(신규, 취소됨, 거부됨 등).

스트림을 종료하려면 해당하는 **UnSubscribe*** 메서드를 사용하십시오. 구성 요소는 연결성을 모니터링하기 위해 **PING** 명령을 자동으로 보내고 반환된 **PONG**을 유효성 검사합니다.

Public REST endpoints

공개적으로 사용 가능한 REST 메서드입니다.

Method	설명
Ping	/api/v3/ping 을 호출하고 거래소가 빈 객체로 응답하면 True 를 반환하는 연결 확인입니다.

GetServerTime	/api/v3/time에서 현재 거래소 타임스탬프를 검색합니다.
GetExchangeInformation	/api/v3/exchangeInfo에서 거래 규칙과 심볼 메타데이터를 반환합니다.
GetOrderBook	/api/v3/depth에서 order book 스냅샷을 다운로드합니다. 레벨 수를 제어하려면 선택적 limit 매개변수를 설정하십시오(기본값 100).
GetTrades	선택적 limit (기본값 100)와 함께 /api/v3/trades에서 최근 공개 거래를 검색합니다.
GetHistoricalTrades	선택적 limit 및 fromId 필터로 과거 거래 데이터를 제공합니다.
GetAggregateTrades	/api/v3/aggTrades를 통해 압축/집계된 거래를 반환하며, 선택적 limit , fromId , startTime 및 endTime 필터를 지원합니다.
GetKlines	시간 범위 및 limit 필터를 지원하는 /api/v3/klines에서 캔들스틱을 다운로드합니다.
GetAveragePrice	/api/v3/avgPrice에서 현재 가중 평균 가격을 반환합니다.
Get24hrTicker	/api/v3/ticker/24hr을 통해 하나 또는 모든 심볼에 대한 24시간 ticker 통계를 검색합니다.
GetPriceTicker	/api/v3/ticker/price를 사용하여 최신 가격을 가져옵니다. 모든 거래 쌍의 가격을 검색하려면 빈 심볼을 전달하십시오.
GetBookTicker	단일 심볼 또는 전체 거래소 요청을 지원하여 /api/v3/ticker/bookTicker에서 최적의 bid/ask 시세를 가져옵니다.

비공개 REST 엔드포인트

비공개 엔드포인트는 유효한 API 키와 서명된 쿼리 문자열을 요구합니다. 구성 요소는 timestamp, recvWindow 및 HMAC 서명을 자동으로 추가합니다.

Method	설명
GetAccountInformation	/api/v3/account에서 잔액과 권한을 반환합니다.
GetOpenOrders	현재 미결제 주문을 나열합니다(선택적으로 심볼로 필터링).
GetAllOrders	선택적 시간 및 제한 필터로 과거 주문을 검색합니다.
GetOrder	orderId 또는 origClientId 를 제공하여 특정 주문의 상태를 쿼리합니다.
GetMyTrades	선택적 limit , fromId , startTime 및 endTime 필터와 함께 private 거래 실행을 나열합니다.
NewOrder	선택적 timeInForce , quantity , quoteOrderQty , price , newClientId , stopPrice , icebergQty 및 추가 매개변수와 함께 /api/v3/order에 라이브 주문(market, limit, stop 등)을 배치합니다.
TestNewOrder	NewOrder 와 동일한 매개변수 집합을 수락하면서 /api/v3/order/test에 검증 전용 요청을 보냅니다.
CancelOrder	orderId 또는 origClientId 를 사용하여 /api/v3/order에서 특정 주문을 취소합니다.
CancelAllOrders	/api/v3/openOrders를 통해 심볼에 대한 모든 미결제 주문을 일괄 취소합니다.
GetSubAccounts	관리되는 하위 계정을 나열합니다(/api/v3/sub-account/list).
GetSubAccountAssets	특정 하위 계정의 잔액을 반환합니다.
TransferSubAccount	/api/v3/sub-account/transfer를 통해 sub-account 간에 자산을 이체합니다. 금액과 선택적으로 이체 type 을 제공하십시오.
GetDepositAddress	선택적 network 필터와 함께 입금 주소를 요청합니다.
GetDepositHistory	선택적 coin , status , startTime 및 endTime 필터로 입금 기록을 가져옵니다.
GetWithdrawHistory	coin , status 및 선택적 시간 범위로 필터링된 출금 내역을 반환합니다.
Withdraw	필수 coin , address 및 amount 와 선택적 network , addressTag , withdrawOrderId 및 추가 매개변수를 포함하여 /api/v3/capital/withdraw/apply를 통해 출금 요청을 제출합니다.
BatchOrders	단일 요청으로 여러 심볼에 대한 배치 주문을 제출합니다.
GetTradeFee	특정 심볼에 대한 거래 수수료율을 가져옵니다.

GetDepositHistory	선택적 coin , status , startTime 및 endTime 필터로 입금 기록을 가져옵니다.
CancelWithdraw	출금 ID로 보류 중인 출금 요청을 취소합니다.
CreateInternalTransfer	계정 간 내부적으로 자산을 이체합니다.
GetTransferHistory	계정 간 내부 이체 내역을 가져옵니다.

API MEXC Futures

MEXC 무기한 및 인도 계약은 공식 [Futures WebSocket 사양](#)에 문서화된 전용 스트리밍 API를 노출합니다. **TsgcWSAPI_MEXC_Futures** 구성 요소는 실시간 파생 상품 데이터를 소비하는 데 필요한 연결, 로그인 핸드셰이크 및 토픽 관리를 캡슐화합니다.

MEXC는 파생 상품 거래를 위한 REST 인터페이스도 유지합니다. **sgcHTTP_API_MEXC**에 포함된 **TsgcHTTP_API_MEXC_Futures** 구성 요소는 공식 엔드포인트와 일대일로 매핑되는 HTTP 헬퍼로 WebSocket 피드를 보완합니다.

구성 요소 구성

Futures WebSocket 엔드포인트는 **wss://contract.mexc.com/edge**입니다. 메시지는 JSON 객체로 인코딩되며, 거래소는 주기적인 **ping/pong** 프레임을 요구하는데, 이는 구성 요소가 자동으로 처리합니다.

비공개 알림(주문 및 계정 데이터)을 수신하려면 인증해야 합니다. WebSocket 클라이언트를 활성화하기 전에 **MEXCAPI.ApiKey**와 **MEXCAPI.ApiSecret**을 설정하십시오. 자격 증명이 있으면 구성 요소는 소켓이 연결되는 즉시 로그인 요청에 서명합니다(HMAC SHA256).

```
TsgcWebSocketClient *WSCClient = new TsgcWebSocketClient();
TsgcWSAPI_MEXC_Futures *Futures = new TsgcWSAPI_MEXC_Futures();
Futures->Client = WSCClient;
Futures->MEXCAPI->ApiKey = "YOUR_KEY";
Futures->MEXCAPI->ApiSecret = "YOUR_SECRET";
if (WSCClient->Connect()) {
    Futures->SubscribeDepth("BTC_USDT", true);
}
```

Properties

- **Client:** API 구성 요소가 사용하는 WebSocket 전송. 소켓을 활성 상태로 유지하기 위해 하트비트 지원이 활성화되어 있습니다.
- **MEXCAPI.ApiKey / ApiSecret:** 로그인 요청에 서명하는 데 필요합니다. 자격 증명이 없으면 public 채널에만 접근할 수 있습니다.
- **RawMessages:** 활성화되면 JSON 파서를 우회하여 원시 텍스트 프레임이 기본 이벤트를 통해 노출됩니다.

Events

- **OnConnect / OnDisconnect:** WebSocket 수명 주기 알림.
- **OnMEXCLogin:** 인증된 로그인이 성공한 후 발생합니다.
- **OnMEXCSubscribed / OnMEXCUnsubscribed:** 구독 변경 확인입니다. 이벤트는 MEXC가 반환한 채널 이름을 노출합니다.
- **OnMEXCMessage:** 모든 시장 또는 계정 업데이트를 채널 식별자와 함께 전달합니다.
- **OnMEXCError:** 서버가 **rs.error** 페이로드(잘못된 매개변수, 인증 실패 등)로 응답할 때 트리거됩니다.
- **OnMEXCException:** JSON 데이터를 처리하거나 네트워크 작업을 수행하는 동안 발생한 로컬 예외를 보고합니다.

다음 핸들러는 구독 확인을 기록하고 들어오는 깊이 스냅샷을 인쇄합니다:

WebSocket 구독 메서드

각 헬퍼는 공식 API에 설명된 대로 **sub/unsub** 요청을 감쌉니다. 스트림을 취소하려면 해당하는 **UnSubscribe*** 메서드를 사용하십시오.

Method	매개변수	설명
SubscribeDeal	Symbol	계약에서 체결된 거래입니다(채널: deal).
SubscribeTickers	-	모든 계약에 대한 글로벌 티커 통계입니다.
SubscribeTicker	Symbol	단일 상품에 대한 Ticker 요약입니다.
SubscribeDepth	Symbol, Compress	중분 주문서 업데이트
SubscribeDepthFull	Symbol, Level	구성 가능한 레벨 수(기본값 20)의 전체 depth 스냅샷.
SubscribeKline	Symbol, Interval	지원되는 타임프레임(Min1, Min5, Min15, Min30, Min60, Hour4, Hour12, Day1, Week1, Month1)의 캔들스틱 데이터입니다.
SubscribeFundingRate	Symbol	최신 펀딩 비율 공지입니다.
SubscribeIndexPrice	Symbol	기초 인덱스 가격 스트림.
SubscribeFairPrice	Symbol	거래소가 푸시하는 Mark(fair) price 업데이트입니다.

서버가 요청을 확인하면 **OnMEXCSubscribed** 이벤트가 채널 이름(예: **rs.sub.depth.BTC_USDT**)과 함께 발생합니다. 오류는 문제 해결을 위한 서버 메시지를 포함하여 **OnMEXCError**를 통해 전달됩니다.

프라이빗 채널

비공개 채널에는 유효한 API 키가 필요합니다. 자격 증명이 제공되면 구성 요소가 자동으로 인증합니다.

Method	설명
SubscribePersonalOrder	개인 주문 업데이트.
SubscribePersonalOrderDeal	개인 주문 체결(fill) 업데이트.
SubscribePersonalPosition	개인 포지션 업데이트.
SubscribePersonalPlanOrder	개인 플랜(트리거) 주문 업데이트입니다.
SubscribePersonalStopOrder	개인 stop order 업데이트.
SubscribePersonalStopPlanOrder	개인 스탑 플랜 주문 업데이트.
SubscribePersonalRiskLimit	개인 위험 한도 업데이트.
SubscribePersonalADLLevel	개인 ADL(자동 디레버리징) 레벨 업데이트입니다.
SubscribePersonalAsset	개인 자산 업데이트.

Public REST endpoints

공개적으로 사용 가능한 REST 메서드입니다.

Method	설명
GetPing	/api/v1/ping 을 통해 API 접근 가능성을 확인합니다.
GetServerTime	/api/v1/time 에서 서버 타임스탬프를 반환합니다.
GetContracts	사용 가능한 계약 목록을 제공합니다(/api/v1/contract/detail).
GetDepth	/api/v1/contract/depth 에서 호가창 깊이를 다운로드합니다. 레벨 수를 선택하려면 선택적 limit 매개변수(기본값 50)를 사용하십시오.
GetDeals	선택적 limit (기본값 100)와 함께 /api/v1/contract/deals 를 사용하여 최근 거래를 검색합니다.
GetKlines	선택적 startTime , endTime 및 limit 필터(기본값 200)와 함께 /api/v1/contract/kline 을 통해 캔들스틱 데이터를 반환합니다.
GetIndexPrice	요청된 심볼의 기초 인덱스 가격을 가져옵니다(/api/v1/contract/indexPrice).
GetFairPrice	/api/v1/contract/fairPrice 에서 계약에 대한 공정(mark) 가격을 반환합니다.

GetFundingRate	/api/v1/contract/fundingRate를 사용하여 심볼에 대한 최신 funding rate를 보고합니다.
GetAllTickers	실시간 가격 요약과 함께 모든 선물 계약 티커를 가져옵니다.
GetFundingRateHistory	계약에 대한 과거 펀딩 비율 기록을 가져옵니다.
GetFairPriceKline	계약에 대한 공정(마크) 가격 kline/캔들스틱 데이터를 가져옵니다.
GetIndexPriceKline	계약에 대한 인덱스 가격 kline/캔들스틱 데이터를 가져옵니다.

비공개 REST 엔드포인트

프라이빗 파생 상품 엔드포인트에는 API 자격 증명이 필요합니다. 구성 요소는 timestamp와 recvWindow로 각 요청에 자동으로 서명합니다.

Method	설명
GetAccountAssets	/api/v1/private/account/assets에서 마진 잔액을 반환합니다.
GetPositionList	/api/v1/private/position/list를 통해 현재 포지션을 나열하며, 선택적으로 심볼로 필터링합니다.
SetPositionLeverage	심볼의 레버리지를 업데이트하고, 제공된 경우 마진 모드를 업데이트합니다.
PlaceOrder	symbol, side, positionSide, type, quantity 와 선택적 price, clientOrderId 및 추가 매개변수를 제공하여 /api/v1/private/order에서 선물 주문을 실행합니다.
CancelOrder	orderId 또는 clientOrderId 로 /api/v1/private/order/cancel을 사용하여 특정 주문을 취소합니다.
CancelAllOrders	/api/v1/private/order/cancel-all을 통해 심볼에 대한 모든 미결제 주문을 취소합니다.
GetOpenOrders	선택적 심볼 필터와 함께 /api/v1/private/order/list/open을 통해 현재 미결제 주문을 나열합니다.
GetOrderHistory	선택적 심볼, 시간 범위 및 제한 매개변수로 주문 기록(/api/v1/private/order/list/history)을 검색합니다.
GetFundingHistory	/api/v1/private/account/funding에서 선택적 startTime, endTime 및 limit 필터와 함께 과거 funding 지급액을 반환합니다.
GetOpenPositions	현재 미체결인 모든 포지션을 가져옵니다.
ChangeMargin	포지션의 증거금 금액을 변경합니다.
GetPositionMode	현재 포지션 모드(one-way 또는 hedge 모드)를 가져옵니다.
ChangePositionMode	one-way와 hedge 모드 사이에서 position 모드를 변경합니다.
PlaceBatchOrder	단일 배치 요청으로 여러 선물 주문을 실행합니다.
GetOrderDetail	특정 주문에 대한 상세 정보를 가져옵니다.
GetOrderDealDetails	특정 주문에 대한 fill/deal 세부 정보를 가져옵니다.
PlaceTriggerOrder	조건이 충족될 때 실행되는 트리거(계획) 주문을 발주합니다.
CancelTriggerOrder	특정 trigger order를 취소합니다.
CancelAllTriggerOrders	모든 열린 트리거 주문을 취소합니다.
GetTriggerOrders	트리거(플랜) 주문 목록을 가져옵니다.
GetStopOrders	stop order 목록을 가져옵니다.
CancelStopOrder	특정 stop 주문을 취소합니다.
CancelAllStopOrders	모든 미체결 stop 주문을 취소합니다.

API Bitget

Bitget

지원되는 API

- **WebSocket API:** websocket 서버에 연결하고 실시간 시장 데이터 업데이트, 계정 변경 등을 제공합니다.
- **REST API:** HTTP 요청을 보내 시장 데이터를 가져오고, 주문하고, 계정 데이터를 가져옵니다...

Properties

BitgetChannel 속성을 사용하여 채널 유형을 선택하십시오:

- **bgcSpot:** Spot WebSocket API에 연결합니다.
- **bgcFutures:** Futures WebSocket API에 연결합니다.

Bitget 속성에서 다음 속성을 구성할 수 있습니다.

- **ApiKey:** Bitget 계정에서 새 api 키를 요청하고, 그 값을 이 속성에 복사하기만 하면 됩니다. APIKey가 설정되면 클라이언트는 websocket private 서버에 연결합니다. 비어 있으면 WebSocket public 서버에 연결합니다.
- **ApiSecret:** api의 비밀 값입니다.
- **Passphrase:** 새 api 키를 생성할 때 정의된 사용자 지정 문자열입니다.

연결

클라이언트가 Bitget 서버에 성공적으로 연결되면 **OnConnect** 이벤트가 발생합니다. **OnConnect** 이벤트가 발생한 후 Bitget 서버로/에서 메시지를 보내고 받기 시작할 수 있습니다. private websocket 채널에 연결하는 경우, 채널을 구독하기 전에 **OnBitgetAuthentication** 이벤트가 발생할 때까지 기다리고 success 매개변수가 true인지 확인해야 합니다.

```
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");
}
```

Events

Bitget 클라이언트는 연결 흐름을 제어하고 WebSocket 서버에서 전송된 데이터를 얻기 위해 다음 이벤트를 구현합니다:

- **OnConnect:** websocket 클라이언트가 WebSocket Server에 연결할 때 발생합니다.
- **OnDisconnect:** websocket 클라이언트가 WebSocket Server에서 연결을 끊을 때 발생합니다.
- **OnBitgetAuthentication:** 클라이언트가 Private WebSocket Server에 대해 인증할 때 발생합니다.
- **OnBitgetSubscribe:** 클라이언트가 websocket 채널을 구독할 때.
- **OnBitgetUnSubscribe:** 클라이언트가 websocket 채널에서 구독 해제할 때입니다.
- **OnBitgetData:** 클라이언트가 서버로부터 데이터를 받을 때입니다.
- **OnBitgetError:** Bitget websocket 연결 중 오류가 있을 때.
- **OnBitgetHTTPException:** REST 요청 중 오류가 있을 때.

WebSocket API

websocket 피드는 주문 및 거래에 대한 실시간 시장 데이터 업데이트를 제공합니다.
다음 채널을 구독할 수 있습니다:

메서드	공개 또는 비공개	설명

SubscribeTicker	Public	상품의 ticker 스트림을 구독합니다.
SubscribeTrade	Public	최근 거래 스트림을 구독합니다.
SubscribeCandle	Public	캔들스틱 스트림을 구독합니다. 여러 간격을 지원합니다(예: candle1m, candle5m, candle1H).
SubscribeOrderBook	Public	주문서 스트림을 구독합니다. 다양한 깊이 레벨을 지원합니다.
SubscribeOrders	Private	주문 업데이트를 구독합니다. 인증이 필요합니다.
SubscribePositions	Private	포지션 업데이트를 구독합니다. 인증이 필요합니다.
SubscribeAccount	Private	계정 잔액 업데이트를 구독합니다. 인증이 필요합니다.

성공적인 인증 후 private websocket 채널을 구독하는 예제는 아래에서 확인하십시오.

```
void OnBitgetAuthentication(TObject *Sender, bool aSuccess, const string aError, const string aRawMessage)
{
    if (aSuccess == true)
    {
        oBitget->SubscribeOrders("BTCUSD");
        oBitget->SubscribeAccount();
    }
}
```

REST API

REST API는 시장 및 비공개 계정에서 데이터를 요청하는 Public 및 Private 메서드를 제공합니다. 구성 요소의 **REST_API** 속성을 통해 REST API에 액세스하십시오.

메서드	공개 / 비공개	설명
GetServerTime	Public	서버 시간을 가져옵니다.
GetTickers	Public	하나 또는 모든 심볼에 대한 티커 정보를 가져옵니다.
GetOrderBook	Public	심볼에 대한 호가창을 가져옵니다.
GetCandles	Public	심볼에 대한 캔들스틱/kline 데이터를 가져옵니다.
GetRecentTrades	Public	심볼의 가장 최근 거래를 가져옵니다.
PlaceOrder	Private	새 주문을 배치합니다.
CancelOrder	Private	기존 주문을 취소합니다.
GetOpenOrders	Private	미체결 주문 목록을 가져옵니다.
GetOrderDetail	Private	특정 주문의 세부 정보를 가져옵니다.
GetOrderHistory	Private	주문 기록을 가져옵니다.
GetAccountAssets	Private	계정 자산 정보를 가져옵니다.
GetAccountInfo	Private	계정 정보를 가져옵니다.

아래에서 REST API 사용 예제를 확인하십시오.

```
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

지원되는 API

- **WebSocket API:** websocket 서버에 연결하고 실시간 시장 데이터 업데이트, 계정 변경 등을 제공합니다.
- **REST API:** HTTP 요청을 보내 시장 데이터를 가져오고, 주문하고, 계정 데이터를 가져옵니다...

Properties

채널 유형을 선택하려면 **GateIOChannel** 속성을 사용하십시오:

- **giocSpot:** Spot WebSocket API에 연결합니다.
- **giocFutures:** Futures WebSocket API에 연결합니다.

GateIO 속성에서 다음 속성을 구성할 수 있습니다.

- **ApiKey:** Gate.io 계정에서 새 api key를 요청하고 값을 이 속성에 복사하기만 하면 됩니다. ApiKey가 설정되면 클라이언트는 private 서버에 대해 인증합니다. 비어 있으면 public 채널만 사용할 수 있습니다.
- **ApiSecret:** api의 비밀 값입니다.

연결

클라이언트가 Gate.io 서버에 성공적으로 연결되면 **OnConnect** 이벤트가 발생합니다. **OnConnect** 이벤트가 발생한 후 Gate.io 서버로/에서 메시지를 보내고 받기 시작할 수 있습니다. private websocket 채널에 연결하는 경우, private 채널을 구독하기 전에 **OnGateIOAuthentication** 이벤트가 발생할 때까지 기다리고 success 매개변수가 true인지 확인해야 합니다.

```
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");
}
```

Events

GateIO 클라이언트는 연결 흐름을 제어하고 WebSocket 서버에서 전송된 데이터를 가져오기 위해 다음 이벤트를 구현합니다:

- **OnConnect:** websocket 클라이언트가 WebSocket Server에 연결할 때 발생합니다.
- **OnDisconnect:** websocket 클라이언트가 WebSocket Server에서 연결을 끊을 때 발생합니다.
- **OnGateIOAuthentication:** 클라이언트가 비공개 WebSocket 서버에 대해 인증할 때 발생합니다.
- **OnGateIOSubscribe:** 클라이언트가 websocket 채널을 구독할 때.
- **OnGateIOUnSubscribe:** 클라이언트가 websocket 채널의 구독을 취소할 때.
- **OnGateIOData:** 클라이언트가 서버로부터 데이터를 받을 때.
- **OnGateIOError:** GateIO websocket 연결 중 오류가 발생할 때.
- **OnGateIOHTTPException:** REST 요청 중에 오류가 있을 때 발생합니다.

WebSocket API

websocket 피드는 주문 및 거래에 대한 실시간 시장 데이터 업데이트를 제공합니다. 다음 채널을 구독할 수 있습니다:

메서드	공개 또는 비 공개	설명
SubscribeTicker	Public	심볼에 대한 ticker 스트림을 구독합니다.

SubscribeTrades	Public	최근 거래 스트림을 구독합니다.
SubscribeCandlesticks	Public	캔들스틱 스트림을 구독합니다. 여러 간격(예: 1m, 5m, 15m, 30m, 1h)을 지원합니다.
SubscribeOrderBook	Public	호가창 스트림을 구독합니다. 구성 가능한 depth 레벨과 업데이트 간격을 지원합니다.
SubscribeOrders	Private	주문 업데이트를 구독합니다. 인증이 필요합니다.
SubscribeBalances	Private	잔액 업데이트를 구독합니다. 인증이 필요합니다.

공개 websocket 채널을 구독하는 예제는 아래에서 확인하십시오.

```
void OnConnect(TsgcWSConnection *Connection)
{
    oGateIO->SubscribeTicker("BTC_USDT");
    oGateIO->SubscribeOrderBook("BTC_USDT", "20", "100ms");
}
```

REST API

REST API는 시장 및 비공개 계정에서 데이터를 요청하는 Public 및 Private 메서드를 제공합니다. 구성 요소의 **REST_API** 속성을 통해 REST API에 액세스하십시오.

메서드	공개 / 비공개	설명
GetCurrencyPairs	Public	사용 가능한 통화 쌍에 대한 정보를 가져옵니다.
GetTickers	Public	하나 또는 모든 통화 쌍에 대한 ticker 정보를 가져옵니다.
GetOrderBook	Public	통화 쌍에 대한 주문서를 가져옵니다.
GetCandlesticks	Public	통화 쌍에 대한 캔들스틱/kline 데이터를 가져옵니다.
GetTrades	Public	통화 쌍에 대한 가장 최근 거래를 가져옵니다.
PlaceOrder	Private	새 주문을 넣습니다. limit 및 market 주문을 지원합니다.
CancelOrder	Private	기존 주문을 취소합니다.
GetOrder	Private	특정 주문의 세부 정보를 가져옵니다.
GetOpenOrders	Private	미체결 주문 목록을 가져옵니다.
GetSpotAccounts	Private	spot 계정 정보를 가져옵니다.

아래에서 REST API 사용 예제를 확인하십시오.

```
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

Deribit은 Bitcoin과 Ethereum에 대한 선물 및 옵션 거래를 제공하는 암호화폐 파생상품 거래소입니다.

지원되는 API

- **WebSocket API:** websocket 서버에 연결하고 실시간 시장 데이터 업데이트, 계정 변경 등을 제공합니다.
- **REST API:** HTTP 요청을 보내 시장 데이터를 가져오고, 주문하고, 계정 데이터를 가져옵니다...

Properties

Deribit 속성에서 다음 속성을 구성할 수 있습니다.

- **ApiKey:** Deribit 계정에서 새 api key를 요청하여 그 값을 이 속성에 복사하기만 하면 됩니다. ApiKey가 설정되면 클라이언트는 프라이빗 서버에 대해 인증합니다. 비어 있으면 공개 채널만 사용할 수 있습니다.
- **ApiSecret:** api의 비밀 값입니다.
- **TestNet:** 활성화되면 Deribit TestNet 환경에 연결됩니다(기본적으로 비활성화됨). 실제 자금을 위험에 빠뜨리지 않고 테스트하는 데 유용합니다.

연결

클라이언트가 Deribit 서버에 성공적으로 연결되면 **OnConnect** 이벤트가 발생합니다. **OnConnect** 이벤트가 발생한 후 Deribit 서버와 메시지를 송신하고 수신할 수 있습니다. 비공개 websocket 채널에 연결하는 경우, 비공개 채널을 구독하기 전에 **OnDeribitAuthentication** 이벤트가 발생할 때까지 기다리고 success 매개변수가 true인지 확인해야 합니다.

```
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");
}
```

Events

Deribit 클라이언트는 연결 흐름을 제어하고 WebSocket 서버에서 전송된 데이터를 가져오기 위해 다음 이벤트를 구현합니다:

- **OnConnect:** websocket 클라이언트가 WebSocket Server에 연결할 때 발생합니다.
- **OnDisconnect:** websocket 클라이언트가 WebSocket Server에서 연결을 끊을 때 발생합니다.
- **OnDeribitAuthentication:** 클라이언트가 Private WebSocket Server에 대해 인증할 때 발생합니다.
- **OnDeribitSubscribe:** 클라이언트가 websocket 채널을 구독할 때.
- **OnDeribitUnSubscribe:** 클라이언트가 websocket 채널을 구독 취소할 때.
- **OnDeribitData:** 클라이언트가 서버로부터 데이터를 수신할 때.
- **OnDeribitError:** Deribit websocket 연결 중에 오류가 있을 때.
- **OnDeribitHTTPException:** REST 요청 중에 오류가 있을 때.

WebSocket API

websocket 피드는 주문 및 거래에 대한 실시간 시장 데이터 업데이트를 제공합니다.

다음 **Public** 채널을 구독할 수 있습니다:

메서드	설명
SubscribeTicker	instrument의 ticker 스트림을 구독합니다. 실시간 가격 및 거래량 데이터를 제공합니다.
SubscribeTrades	상품의 최근 거래 스트림을 구독합니다.

SubscribeOrderBook	order book 스트림을 구독합니다. 구성 가능한 그룹화, depth 레벨 및 업데이트 간격을 지원합니다.
SubscribeCandle	candlestick 스트림을 구독합니다. 여러 해상도(예: 1, 3, 5, 10, 15, 30, 60, 120, 180, 360, 720, 1D)를 지원합니다.
SubscribePerpetual	계약(instrument)에 대한 무기한 계약 데이터를 구독합니다. 펀딩 비율 및 기타 무기한 관련 정보를 제공합니다.
SubscribeBookChange	instrument에 대한 주문 장부 변경 알림을 구독합니다.

다음 비공개 채널을 구독할 수 있습니다(인증 필요):

메서드	설명
SubscribeOrders	상품의 주문 업데이트를 구독합니다.
SubscribePositions	포지션 업데이트를 구독합니다. 통화 및 상품 종류별 필터링을 지원합니다.
SubscribeAccountChanges	통화에 대한 계정 잔액 변경을 구독합니다.
SubscribeUserTrades	instrument에 대한 사용자 거래 업데이트를 구독합니다.

웹소켓 채널을 구독하는 예제를 아래에서 확인하십시오.

```
void OnDeribitAuthentication(TObject *Sender, bool aSuccess, const string aError, const string aRawMessage)
{
    if (aSuccess == true)
    {
        oDeribit->SubscribeOrders("BTC-PERPETUAL");
        oDeribit->SubscribePositions("BTC");
    }
}
```

REST API

REST API는 시장, 거래, 계정, wallet에서 데이터를 요청하는 Public 및 Private 메서드를 제공합니다. 구성 요소의 **REST_API** 속성을 통해 REST API에 액세스하십시오.

Market Data (Public)

메서드	설명
GetInstruments	통화에 사용 가능한 instrument를 가져옵니다.
GetTicker	상품에 대한 ticker 정보를 가져옵니다.
GetOrderBook	instrument의 호가창을 가져옵니다.
GetTrades	상품에 대한 가장 최근 거래를 가져옵니다.
GetCurrencies	지원되는 통화 목록을 가져옵니다.
GetIndexPrice	인덱스 이름에 대한 현재 인덱스 가격을 가져옵니다.
GetFundingRateHistory	상품의 펀딩 비율 기록을 가져옵니다.
GetFundingRateValue	상품의 현재 펀딩 비율 값을 가져옵니다.
GetBookSummaryByCurrency	통화의 모든 상품에 대한 book 요약 가져옵니다.
GetBookSummaryByInstrument	특정 상품의 호가 요약을 가져옵니다.

Trading (Private)

메서드	설명
Buy	매수 주문을 배치합니다. market 및 limit 주문 유형을 지원합니다.
Sell	매도 주문을 넣습니다. 시장가 및 지정가 주문 유형을 지원합니다.
CancelOrder	order ID로 기존 주문을 취소합니다.
CancelAllOrders	모든 미체결 주문을 취소합니다.
CancelAllByInstrument	특정 instrument에 대한 모든 미체결 주문을 취소합니다.
EditOrder	기존 주문을 편집합니다(수량 또는 가격 변경).
GetOpenOrders	통화에 대한 미체결 주문 목록을 가져옵니다.

GetOrderState	특정 주문의 상태를 가져옵니다.
GetOrderHistory	통화에 대한 주문 기록을 가져옵니다.

Account (Private)

메서드	설명
GetPositions	통화에 대한 포지션을 가져옵니다.
GetAccountSummary	통화에 대한 계정 요약을 가져옵니다.
GetSubAccounts	하위 계정 목록을 가져옵니다.
GetTransactionLog	통화의 트랜잭션 로그를 가져옵니다.

Wallet (Private)

메서드	설명
GetDeposits	통화에 대한 입금 내역을 가져옵니다.
GetWithdrawals	통화에 대한 출금 기록을 가져옵니다.
GetTransfers	통화의 이체 내역을 가져옵니다.

아래에서 REST API 사용 예제를 확인하십시오.

```
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](#)

지원되는 API

- **WebSocket API:** websocket 서버에 연결하고 실시간 시장 데이터 업데이트, 계정 변경 등을 제공합니다.
- **REST API:** HTTP 요청을 보내 시장 데이터를 가져오고, 주문하고, 계정 데이터를 가져옵니다...

Properties

채널 유형을 선택하려면 **Channel** 속성을 사용하십시오:

- **cccMarket:** public 시장 데이터를 위해 Market WebSocket API에 연결합니다.
- **cccUser:** 비공개 계정 데이터를 위한 User WebSocket API에 연결합니다(인증 필요).

CryptoCom 속성에서 다음 속성을 구성할 수 있습니다.

- **ApiKey:** Crypto.com 계정에서 새 api 키를 요청하고, 그 값을 이 속성에 복사하기만 하면 됩니다. private 채널 및 인증된 REST API 호출에 필요합니다.
- **ApiSecret:** api의 비밀 값입니다.

연결

클라이언트가 Crypto.com 서버에 성공적으로 연결되면 **OnConnect** 이벤트가 발생합니다. **OnConnect** 이벤트가 발생한 후, Crypto.com 서버로부터 메시지를 **보내고 받기** 시작할 수 있습니다. User 채널에 연결하는 경우, 프라이빗 채널을 구독하기 전에 **OnCryptoComAuthentication** 이벤트가 발생할 때까지 기다리고 success 매개변수가 true인지 확인해야 합니다.

```
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");
}
```

Events

Crypto.com 클라이언트는 연결 흐름을 제어하고 WebSocket 서버에서 전송된 데이터를 가져오기 위해 다음 이벤트를 구현합니다:

- **OnConnect:** websocket 클라이언트가 WebSocket Server에 연결할 때 발생합니다.
- **OnDisconnect:** websocket 클라이언트가 WebSocket Server에서 연결을 끊을 때 발생합니다.
- **OnCryptoComAuthentication:** 클라이언트가 Private WebSocket Server에 대해 인증할 때 발생합니다.
- **OnCryptoComSubscribe:** 클라이언트가 websocket 채널에 구독할 때 발생합니다.
- **OnCryptoComUnSubscribe:** 클라이언트가 websocket 채널에서 구독을 취소할 때.
- **OnCryptoComData:** 클라이언트가 서버로부터 데이터를 수신할 때 발생합니다.
- **OnCryptoComError:** Crypto.com websocket 연결 중 오류가 발생할 때.
- **OnCryptoComHTTPException:** REST 요청 중에 오류가 있을 때 발생합니다.

WebSocket API

websocket 피드는 주문 및 거래에 대한 실시간 시장 데이터 업데이트를 제공합니다.

다음 **Market 채널**(공개)을 구독할 수 있습니다:

메서드	설명
SubscribeTicker	instrument의 ticker 스트림을 구독합니다. 실시간 가격 및 거래량 데이터를 제공합니다.

SubscribeTrade	상품의 최근 거래 스트림을 구독합니다.
SubscribeCandlestick	캔들스틱 스트림을 구독합니다. 여러 간격(예: 5m, 15m, 30m, 1h, 4h, 1D)을 지원합니다.
SubscribeBook	order book 스트림을 구독합니다. 구성 가능한 depth 레벨을 지원합니다(예: 10, 50).

다음 **User 채널**(비공개, 인증 필요)을 구독할 수 있습니다:

메서드	설명
SubscribeOrders	주문 업데이트를 구독합니다. 선택적으로 instrument 이름으로 필터링합니다.
SubscribeUserTrades	사용자 거래 업데이트를 구독합니다. 선택적으로 상품 이름으로 필터링할 수 있습니다.
SubscribeBalance	계정의 잔액 업데이트를 구독합니다.

아래에서 시장 채널을 구독하는 예제를 확인하십시오.

```
void OnConnect(TsgcWSConnection *Connection)
{
    oCryptoCom->SubscribeTicker("BTC_USDT");
    oCryptoCom->SubscribeBook("BTC_USDT", "10");
}
```

REST API

REST API는 시장 및 비공개 계정에서 데이터를 요청하는 Public 및 Private 메서드를 제공합니다. 구성 요소의 **REST_API** 속성을 통해 REST API에 액세스하십시오.

메서드	공개 / 비공개	설명
GetInstruments	Public	사용 가능한 상품 목록을 가져옵니다.
GetTicker	Public	하나 또는 모든 instrument에 대한 ticker 정보를 가져옵니다.
GetOrderBook	Public	instrument의 호가창을 가져옵니다.
GetTrades	Public	상품에 대한 가장 최근 거래를 가져옵니다.
GetCandlestick	Public	상품의 candlestick/kline 데이터를 가져옵니다.
CreateOrder	Private	새 주문을 생성합니다. 지정가 및 시장가 주문을 지원합니다.
CancelOrder	Private	기존 주문을 취소합니다.
CancelAllOrders	Private	instrument에 대한 모든 미체결 주문을 취소합니다.
GetOpenOrders	Private	미체결 주문 목록을 가져옵니다.
GetOrderDetail	Private	특정 주문의 세부 정보를 가져옵니다.
GetOrderHistory	Private	주문 기록을 가져옵니다.
GetAccountSummary	Private	계정 요약 정보를 가져옵니다.

아래에서 REST API 사용 예제를 확인하십시오.

```
TsgcWS_API_CryptoCom *oCryptoCom = new TsgcWS_API_CryptoCom();
oCryptoCom->CryptoCom->ApiKey = "your_api_key";
oCryptoCom->CryptoCom->ApiSecret = "your_api_secret";
// Get instruments
oCryptoCom->REST_API->GetInstruments();
// Create an order
oCryptoCom->REST_API->CreateOrder("BTC_USDT", "BUY", "MARKET", "", "0.001");
// Get account summary
oCryptoCom->REST_API->GetAccountSummary();
```

API HTX

HTX

HTX(이전 Huobi)는 국제적인 다국어 cryptocurrency exchange입니다. HTX API 구성 요소는 기존 Huobi WebSocket API를 보완하는 HTTP REST API를 제공합니다.

지원되는 API

- **WebSocket API:** websocket 서버에 연결하여 실시간 시장 데이터 업데이트, 계정 변경 등을 제공합니다. WebSocket 구독 메서드는 Huobi API 문서를 참조하십시오.
- **REST API:** HTTP 요청을 보내 시장 데이터를 가져오고, 주문하고, 계정 데이터를 가져옵니다...

Properties

Huobi 속성(WebSocket용) 또는 HTXOptions 속성(REST API용)에서 다음 속성을 구성할 수 있습니다.

- **ApiKey:** HTX 계정에서 새 api key를 요청하고, 그 값을 이 속성에 복사하면 됩니다. ApiKey가 설정되면 클라이언트는 websocket private 서버에 연결됩니다. 비어 있으면 WebSocket public 서버에 연결됩니다.
- **ApiSecret:** api의 비밀 값입니다.

연결

HTX 구성 요소는 Huobi WebSocket API 클라이언트를 확장합니다. 클라이언트가 HTX 서버에 성공적으로 연결되면 **OnConnect** 이벤트가 발생합니다. ApiKey가 비어 있지 않으면 클라이언트는 private websocket 서버에 연결을 시도하므로 private 메서드만 사용할 수 있습니다. ApiKey가 비어 있으면 클라이언트는 public websocket 서버에 연결하고 public 메서드만 사용할 수 있습니다.

```
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");
}
```

Events

HTX 클라이언트는 연결 흐름을 제어하고 WebSocket 서버에서 전송된 데이터를 가져오기 위해 다음 이벤트를 구현합니다:

- **OnConnect:** websocket 클라이언트가 WebSocket Server에 연결할 때 발생합니다.
- **OnDisconnect:** websocket 클라이언트가 WebSocket Server에서 연결을 끊을 때 발생합니다.
- **OnHuobiSubscribed:** 성공적인 구독 후 호출되는 이벤트입니다.
- **OnHuobiUnSubscribed:** 성공적인 구독 취소 후 호출되는 이벤트입니다.
- **OnHuobiUpdate:** 데이터(kline, market depth 등)에 업데이트가 있을 때마다 이 이벤트가 호출됩니다.
- **OnHuobiError:** HTX API에 오류가 있으면 이 이벤트가 오류에 대한 정보를 제공합니다.

WebSocket API

WebSocket API는 실시간 시장 데이터 업데이트를 제공합니다. 공개 채널(SubscribeKLine, SubscribeMarketDepth, SubscribeTradeDetail 등)과 프라이빗 채널(SubscribeOrderUpdates, SubscribeTradeClearing, SubscribeAccountChange)을 포함한 WebSocket 구독 메서드의 전체 목록은 Huobi API 문서를 참조하십시오.

REST API

REST API는 시장 및 비공개 계정에서 데이터를 요청하기 위한 Public 및 Private 메서드를 제공합니다. REST API는 **TsgcHTTP_API_HTX** 구성 요소를 통해 액세스됩니다.

Market Data (Public)

메서드	설명
GetServerTime	서버 시간을 가져옵니다.
GetSymbols	사용 가능한 거래 심볼 목록을 가져옵니다.
GetCurrencies	지원되는 통화 목록을 가져옵니다.
GetMarketTickers	모든 거래 심볼의 티커를 가져옵니다.
GetMarketDetail	심볼에 대한 24시간 시장 세부 정보를 가져옵니다.
GetMarketDepth	심볼에 대한 주문서 깊이를 가져옵니다. 구성 가능한 깊이 유형 및 깊이 수준을 지원합니다.
GetMarketTrade	심볼에 대한 가장 최근 거래를 가져옵니다.
GetMarketHistoryTrade	심볼에 대한 가장 최근 거래 기록을 가져옵니다.
GetMarketHistoryKline	캔들스틱/kline 데이터를 가져옵니다. 여러 기간(1min, 5min, 15min, 30min, 60min, 4hour, 1day, 1mon, 1week, 1year)을 지원합니다.

Trading (Private)

메서드	설명
PlaceOrder	새 주문을 넣습니다. buy-market, sell-market, buy-limit, sell-limit 및 기타 주문 유형을 지원합니다.
CancelOrder	order ID로 기존 주문을 취소합니다.
GetOrder	특정 주문의 세부 정보를 가져옵니다.
GetOpenOrders	계정의 미체결 주문 목록을 가져옵니다.
GetOrderHistory	심볼에 대한 주문 내역을 가져옵니다.

Account (Private)

메서드	설명
GetAccounts	계정 목록을 가져옵니다.
GetAccountBalance	특정 계정의 잔액을 가져옵니다.
GetAccountAssetValuation	계정 유형(예: spot)에 대한 총 자산 평가액을 가져옵니다.

아래에서 REST API 사용 예제를 확인하십시오.

```
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

WhatsApp Business Platform의 클라우드 호스팅 버전을 사용하여 메시지를 전송하고 및 수신합니다. Cloud API를 사용하면 자체 서버 호스팅 비용 없이 WhatsApp Business API를 구현할 수 있으며, 비즈니스 메시징을 더 쉽게 확장할 수 있습니다. Cloud API는 초당 최대 80개의 메시지(텍스트 및 미디어 메시지 포함) 전송 및 수신을 지원합니다.

WhatsApp Business API를 사용하면 중대형 기업이 대규모로 고객과 소통할 수 있습니다. API를 사용하여, 기업은 수천 명의 고객을 상담원 또는 봇과 연결하는 시스템을 구축하여 프로그래밍 방식과 수동 통신을 모두 가능하게 할 수 있습니다. 또한, CRM 및 마케팅 플랫폼과 같은 수많은 백엔드 시스템과 API를 통합할 수 있습니다.

Features

기업은 Cloud API에서 모든 새 기능을 더 빠르게 얻을 수 있습니다. 현재 WhatsApp Business Cloud API에는 WhatsApp Business API에서 사용 가능한 모든 기능이 포함되어 있습니다.

WhatsApp Cloud API의 유용한 기능:

- **CRM, analytics** 및 서드파티 앱과 같은 도구와 WhatsApp 메시징을 통합합니다
- **Green Tick**, 검증된 WhatsApp Business 프로필
- **WhatsApp Broadcast & Bulk Messaging**
- 앱 또는 인터페이스 없음, BSP 또는 CRM을 통해 사용
- 서드파티 앱을 사용한 **WhatsApp 챗봇 & 채팅 자동화**
- WhatsApp 메시지를 대규모로 예약하십시오
- 대화형 메시징 기능에는 목록 메시지, 응답 버튼, CTA 메시지가 포함됩니다

Most common uses

- 구성
 - [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](#)

시작하기

테스트 번호를 사용하여 첫 번째 메시지를 송수신하려면 다음 단계를 완료하십시오:

1. Developer Assets 및 Platform Access 설정

- [Meta Developer](#)로 등록
- 계정에 2단계 인증 활성화
- **Meta 앱 생성**: [developers.facebook.com](#) > **My Apps** > **Create App**으로 이동하십시오. "Business" 유형을 선택하고 화면의 안내를 따르십시오.

App Dashboard에서 WhatsApp에 연결하려는 앱을 클릭하십시오. 아래로 스크롤하여 "WhatsApp" 제품을 찾고 **Set up**를 클릭하십시오.

다음으로, 기존 Business Manager를 선택하는 옵션(보유한 경우)이 표시되거나, 원하는 경우 온보딩 프로세스가 자동으로 생성할 수 있습니다(나중에 필요한 경우 비즈니스를 사용자 지정할 수 있습니다). 선택한 다음 **Continue**를 클릭하십시오.

Continue를 클릭하면, 온보딩 프로세스는 다음 작업을 수행합니다:

- 앱이 선택했거나 자동으로 생성된 Business Manager와 연결됩니다.
- WhatsApp 테스트 전화번호가 비즈니스에 추가됩니다. 이 테스트 전화번호를 사용하여 실제 전화번호를 등록하거나 마이그레이션하지 않고 WhatsApp Business Platform을 탐색할 수 있습니다. 테스트 전화번호는 최대 5명의 수신자(전 세계 어디든 가능)에게 무제한 메시지를 보낼 수 있습니다.

2. 테스트 메시지 보내기

이제 IDE를 열고 새 프로젝트를 생성할 수 있습니다. TsgcWhatsapp_Client 구성 요소를 배치하고 다음 속성을 채워주세요.

- **WhatsappOptions.PhoneNumberId**: 메시지를 보내는 데 사용되는 Phone Number의 ID입니다.
- **WhatsappOptions.Token**: 24시간 동안 유효한 Temporary Access Token입니다.

이 2개의 속성이 올바르게 구성되면 **SendTest** 메서드를 호출하여 **Whatsapp Business Platform**을 사용하여 전화번호로 첫 번째 메시지를 보내십시오.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendTest("34605889421");
```

3. Webhook 구성

메시지를 받거나 메시지의 상태가 변경되었을 때 알림을 받으려면 앱에 Webhooks 엔드포인트를 설정해야 합니다. Webhooks를 설정해도 전화번호의 상태에는 영향을 주지 않으며 메시지 전송 또는 수신을 방해하지 않습니다.

시작하려면 먼저 엔드포인트를 생성해야 하므로, 먼저 WhatsApp Client 구성 요소의 **ServerOptions** 속성을 구성하고 다음 속성을 구성하십시오:

- **ServerOptions**: 여기에서 바인딩할 IP 주소, 수신 포트, SSL 사용 여부(WebHook은 보안 서버에서 실행되어야 합니다. 서버를 SSL을 사용하도록 구성하거나 WebHook 요청을 HTTPs가 아닌 서버로 프록시할 수 있습니다)를 구성할 수 있습니다. 서버는 **TsgcWebSocketHTTPServer**를 기반으로 합니다.
 - **WebhookOptions**: 이 속성을 사용하면 나중에 개발자 facebook 계정에서 구성될 Webhook 속성을 설정할 수 있습니다.
 - **Endpoint**: 엔드포인트의 이름이며 기본값은 /webhook입니다. 예: 서버가 https://www.esegece.com에서 수신 중이면 엔드포인트는 "https://www.esegece.com/webhook"이 됩니다.
 - **Token**: 사용자가 정의한 모든 값을 포함할 수 있는 보안 문자열입니다. Webhook 등록이 올바른지 검증하는 데 사용됩니다.

서버를 구성한 후 **StartServer** 메서드를 사용하여 서버를 시작하고 들어오는 요청을 수락할 수 있습니다.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->ServerOptions->WebhookOptions->PhoneNumberId = "/webhook";
oClient->ServerOptions->WebhookOptions->Token = "MySecretToken";
oClient->StartServer();
```

엔드포인트가 준비되면 App Dashboard로 이동하십시오.

App Dashboard에서 WhatsApp 제품을 찾아 **Configuration**을 클릭하십시오. 그런 다음 webhooks 섹션을 찾아 **Configure a webhook**을 클릭하십시오. 클릭하면 화면에 대화 상자가 나타나 두 가지 항목을 요청합니다.

- **Callback URL**: Meta가 이벤트를 보낼 URL입니다.
- **Verify Token**: 이 문자열은 webhook 엔드포인트를 생성할 때 사용자가 설정합니다.

정보를 추가한 후 **Verify and Save**를 클릭하십시오.

App Dashboard로 돌아가서 왼쪽 패널에서 **WhatsApp > Configuration** 을 클릭하십시오. Webhooks 아래에서 **Manage** 를 클릭하십시오. 알림을 받을 수 있는 모든 객체가 있는 대화 상자가 열립니다. 사용자로부터 메시지를 받으려면 **messages** 에 대해 **Subscribe** 를 클릭하십시오.

4. 테스트 메시지 수신

새 메시지가 수신될 때마다 클라이언트 이벤트 **OnMessageReceived**가 호출됩니다.

```
void OnMessageReceived(TObject *Sender, TsgcWhatsapp_Receive_Message *aMessage, ref bool aMarkAsRead)
{
    DoLog("Received: " + aMessage->Messages->_Message[0]->Id);
}
```

이제 Webhook이 설정되었으므로, 사용한 테스트 번호로 메시지를 보내십시오. 메시지 내용과 함께 Webhooks 알림을 즉시 받아야 합니다!

WhatsApp API는 이전에(최근 24시간 이내) 연락한 적이 없는 전화로 자유 텍스트 메시지를 보내는 것을 허용하지 않습니다. 개발자 계정 번호로 문자를 보낸 적이 없는 전화로 텍스트 메시지를 보내는 유일한 방법은 (Meta가 사전 승인한) Template을 보내는 것입니다. 자유 텍스트 메시지 테스트를 위해 이 제한을 재정의하려면, 먼저 대상 번호에서 개발자 계정 번호로 WhatsApp 메시지를 보내십시오. 그러면 24시간 동안 자유 텍스트 메시지를 보낼 수 있습니다.

Events

OnBeforeSendMessage

이 이벤트는 메시지가 WhatsApp 서버로 전송되기 전에 호출됩니다. RawMessage 매개변수를 통해 내부 메시지에 액세스할 수 있습니다.

OnBeforeSubscribe

이 이벤트는 서버가 토픽을 구독하기 전에 호출됩니다. Accept 매개변수를 사용하여 구독을 허용하거나 거부하십시오. 기본적으로 서버는 요청된 모든 이벤트를 구독합니다.

OnRawMessage

이 이벤트는 서버가 아직 파싱되지 않은 새 메시지를 수신할 때 호출되므로 원시 메시지에 접근할 수 있습니다.

OnMessageReceived

이 이벤트는 서버가 새 메시지를 수신하고 구문 분석한 후에 호출됩니다. MarkAsRead 매개변수를 True로 설정하면, 발신자는 이중 확인 표시를 받게 됩니다.

OnMessageSent

이 이벤트는 서버가 이전에 보낸 메시지에 대한 새 상태 메시지를 수신할 때마다 호출됩니다. 메시지가 전송, 전달 또는 읽음 처리되었는지 알려면 Status 속성을 읽으십시오.

WhatsApp Create App

developers.facebook.com으로 이동하여 **Create App**을 수행하십시오.

앱 유형으로 **Business Type**을 선택하고 진행하십시오.

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**

앱의 이름을 제공하십시오(“WhatsApp” 또는 “Facebook”과 같은 상표 이름 사용을 피하십시오).

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**

앱이 생성되면, 다음 화면에서 **WhatsApp 버튼**을 클릭하여 앱에 WhatsApp 전송 기능을 추가하십시오.



다음 화면에서 WhatsApp 앱을 Facebook 비즈니스 계정에 연결해야 합니다. 아직 계정이 없는 경우 새 비즈니스 계정을 생성하는 옵션도 있습니다.

WhatsApp Phone Number Id

WhatsApp Cloud API에 등록하면, Facebook이 애플리케이션의 기본 전송 주소가 될 Test WhatsApp 전화번호를 제공합니다. 수신자의 경우, 개발 단계 동안 결제 없이 최대 5개의 전화번호를 추가할 수 있는 옵션이 있습니다.

나중에 5개 전화번호 제한을 피하기 위해 자체 전화번호를 등록할 수 있습니다.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();  
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
```

WhatsApp Token

WhatsApp Cloud API는 Cloud API를 사용하여 메시지를 보내려면 유효한 토큰이 필요합니다.

Facebook은 최대 5개의 전화번호로 메시지를 보낼 수 있는 Test WhatsApp 전화번호를 제공합니다. 나중에 자신의 전화번호를 등록하여 이 제한을 재정의할 수 있습니다.

WhatsApp API는 23 시간 동안 유효한 **임시 액세스 토큰**을 제공합니다. 이 토큰은 메시지 전송을 허용하기 위해 TsgcWhatsApp_Client 구성 요소에 구성되어야 합니다.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();  
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
```

오래 유효한 token이 필요한 경우 system 사용자를 만들거나 업데이트하고 **whatsapp_business_messaging** permission으로 새 token을 생성할 수 있습니다. 이를 통해 23시간마다 token을 업데이트하지 않고도 WhatsApp 메시지를 보내고 받을 수 있습니다.

WhatsApp Webhook

비즈니스가 수신하는 메시지와 고객 프로필 업데이트에 대한 알림을 받으려면 Webhook을 구독하십시오.

Create Endpoint

알림을 받기 시작하려면 알림을 수신할 엔드포인트를 서버에 생성해야 합니다.

엔드포인트는 두 가지 유형의 HTTPS 요청을 처리할 수 있어야 합니다: Verification Request와 Event Notification. 두 요청 모두 HTTPS를 사용하므로, 서버에는 올바르게 구성되고 설치된 유효한 TLS 또는 SSL 인증서가 있어야 합니다. 자체 서명 인증서는 지원되지 않습니다.

WhatsApp Settings에서 Webhook을 구성할 때, 서버가 수신 대기하는 엔드포인트와 임의의 값이 될 수 있는 Token을 정의해야 합니다. 이 token은 webhook 엔드포인트를 등록하고 구독자가 유효한지 검증할 때 사용됩니다.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->ServerOptions->WebhookOptions->PhoneNumberId = "/webhook";
oClient->ServerOptions->WebhookOptions->Token = "MySecretToken";
oClient->StartServer();
```

Webhook이 구성되면, 새 메시지가 수신될 때마다 알림을 받기 위해 **Messages** Webhook 필드를 구독하십시오.

[SSL Server](#) 구성에 대해 자세히 읽을 수 있습니다.

WhatsApp Security

새 메시지가 수신되거나 메시지의 새 상태가 있을 때마다, 서버는 [Webhook](#)에 구성된 엔드포인트에서 알림을 받습니다. 요청이 WhatsApp Cloud API Servers에서 오는지 확인하기 위해, 요청에는 서명이 있는 헤더가 포함됩니다. 메시지를 처리하기 전에 서명을 검증하도록 WhatsApp 클라이언트를 구성할 수 있습니다.

이를 수행하려면, 먼저 **ServerOptions.Application.Secret** 속성에 Application Secret을 설정하고 **VerifySignature** 속성을 활성화해야 합니다.

구성되면 새 메시지가 수신될 때마다 먼저 서명이 검증되며, 서명이 잘못된 경우 오류 500을 반환하고 메시지는 처리되지 않습니다.

WhatsApp Send Messages

모든 API 호출은 Access Token으로 인증되어야 합니다. 개발자는 **App Dashboard > WhatsApp > Getting Started**에서 생성된 액세스 토큰으로 API 호출을 인증할 수 있습니다

API 호출은 Message Id를 문자열로 반환합니다.

Text Messages

SendMessageText 메서드를 호출하고 다음 매개변수를 전달하십시오.

- **aTo:** 전화번호
- **aText:** 메시지의 텍스트입니다.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendMessageText("34605889421", "Hello from sgcWebSockets!!!");
```

Image Messages

SendMessageImage 메서드를 호출하고 다음 매개변수를 전달하십시오:

- **aTo:** 전화번호
- **aLink:** 보낼 이미지가 있는 url입니다.
- **aCaption:** 이미지의 제목(선택적).

```
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

SendMessageDocument 메서드를 호출하고 다음 매개변수를 전달하십시오:

- **aTo:** 전화번호
- **aLink:** 보낼 문서가 있는 url
- **aCaption:** 문서의 제목(선택 사항).
- **aFileName:** 파일 이름(선택 사항).

```
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

SendMessageAudio 메서드를 호출하고 다음 매개변수를 전달하십시오:

- **aTo:** 전화번호
- **aLink:** 보낼 오디오가 있는 url

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendMessageAudio("34605889421", "https://www.audio.com/audio.mp3");
```

비디오 메시지

SendMessageVideo 메서드를 호출하고 다음 매개변수를 전달하십시오:

- **aTo:** 전화번호
- **aLink:** 전송할 비디오가 있는 url

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendMessageVideo("34605889421", "https://www.video.com/audio.mp4");
```

스티커 메시지

SendMessageSticker 메서드를 호출하고 다음 매개변수를 전달하십시오:

- **aTo:** 전화번호
- **aLink:** 보낼 스티커가 있는 url입니다

```
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

SendMessageLocation 메서드를 호출하고 다음 매개변수를 전달하십시오:

- **aTo:** 전화번호
- **aLongitude:** 위치의 경도입니다.
- **aLatitude:** 위치의 위도입니다.
- **aName:** 위치의 이름.
- **aAddress:** 위치의 주소. **aName**이 설정된 경우에만 표시됩니다.

```
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

SendMessageContact 메서드를 호출하고 다음 매개변수를 전달하십시오:

- **aTo:** 전화번호
- **aName:** 일반적으로 표시되는 전체 이름입니다(필수).
- **aPhone:** 전화번호입니다(선택적).
- **aEmail:** 이메일입니다(선택 사항).

```
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 대화형 메시지 보내기

대화형 메시지는 사용자가 WhatsApp에서 비즈니스로부터 원하는 것을 찾고 선택할 수 있는 더 간단한 방법을 제공합니다. 테스트 중에 대화형 메시징 기능을 사용하는 챗봇은 텍스트 기반 챗봇에 비해 훨씬 높은 응답률과 전환율을 달성했습니다.

다음 메시지가 대화형으로 간주됩니다.

- **List Messages:** 최대 10개의 옵션 메뉴를 포함하는 메시지입니다. 이 유형의 메시지는 사용자가 비즈니스와 상호 작용할 때 선택을 더 간단하고 일관되게 할 수 있는 방법을 제공합니다.
- **Reply Buttons:** 최대 3개의 옵션을 포함하는 메시지로, 각 옵션은 버튼입니다. 이 유형의 메시지는 사용자가 비즈니스와 상호 작용할 때 메뉴에서 선택을 더 빠르게 할 수 있는 방법을 제공합니다. Reply 버튼은 버튼이 있는 대화형 템플릿과 동일한 사용자 경험을 가집니다.

Interactive Message Specifications

- 대화형 메시지는 동일한 흐름에서 함께 결합될 수 있습니다.
- 사용자는 목록 또는 버튼 메시지에서 동시에 둘 이상의 옵션을 선택할 수 없지만, 뒤로 가서 이전 메시지를 다시 열 수 있습니다.
- List 또는 reply 버튼 메시지는 알림으로 사용할 수 없습니다. 현재는 사용자가 마지막으로 보낸 메시지에서부터 24시간 이내에만 전송할 수 있습니다. 24시간 창 밖에서 메시지를 보내려고 하면 오류 메시지가 표시됩니다.

사용해야 하는 경우

목록 메시지는 다음과 같이 여러 옵션을 제시하는 데 가장 적합합니다:

- 고객 지원 또는 FAQ 메뉴
- 테이크아웃 메뉴
- 근처 매장 또는 위치 선택
- 사용 가능한 예약 시간
- 반복할 최근 주문 선택

응답 버튼은 다음과 같이 제한된 옵션 집합에서 빠른 응답을 제공하는 데 가장 적합합니다:

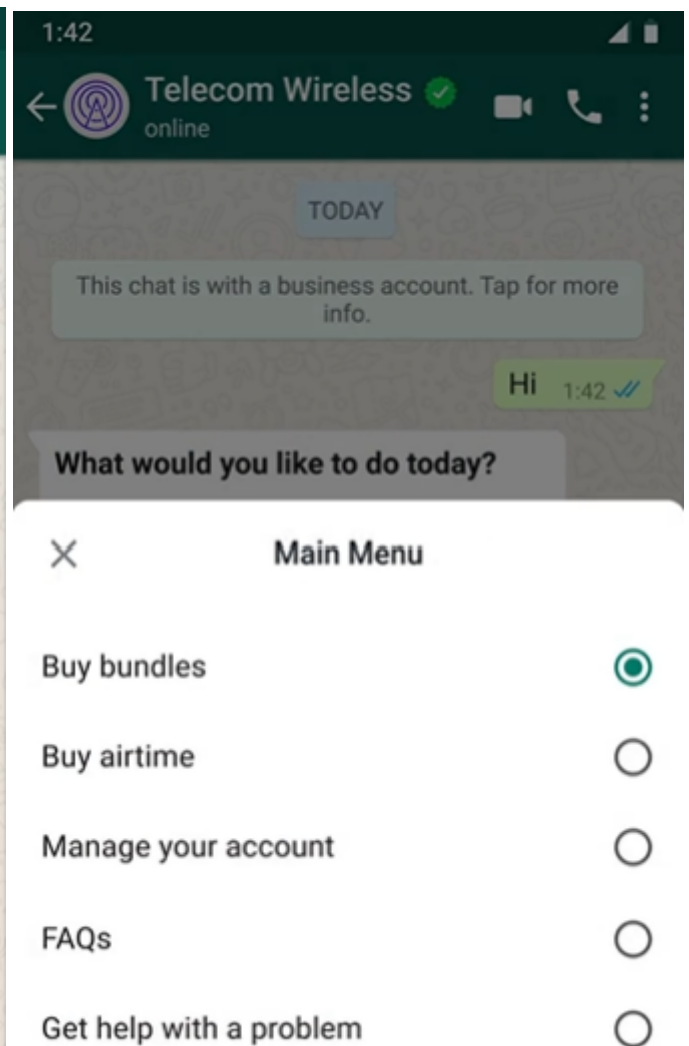
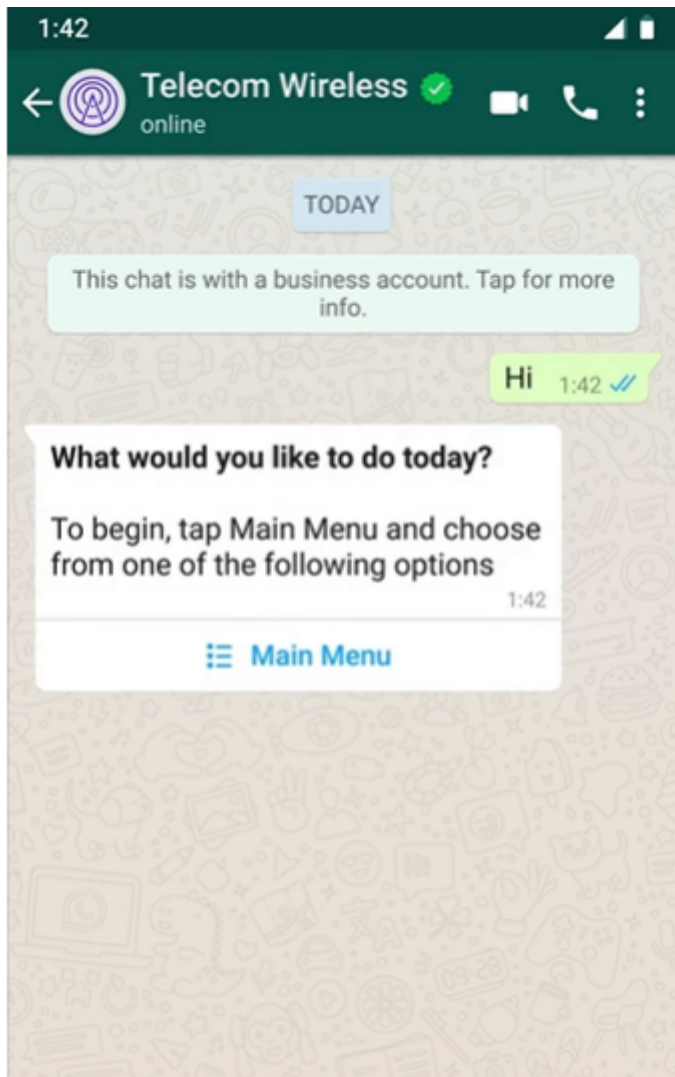
- 통화 충전(Airtime recharge)
- 개인 정보 변경
- 이전 주문 재정렬
- 반품 요청
- 음식 주문에 선택적 추가 항목 추가
- 결제 방법 선택

회신 버튼은 일반적인 응답이 적절하지 않은 '개인화된' 사용 사례에 특히 유용합니다.

Interactive List

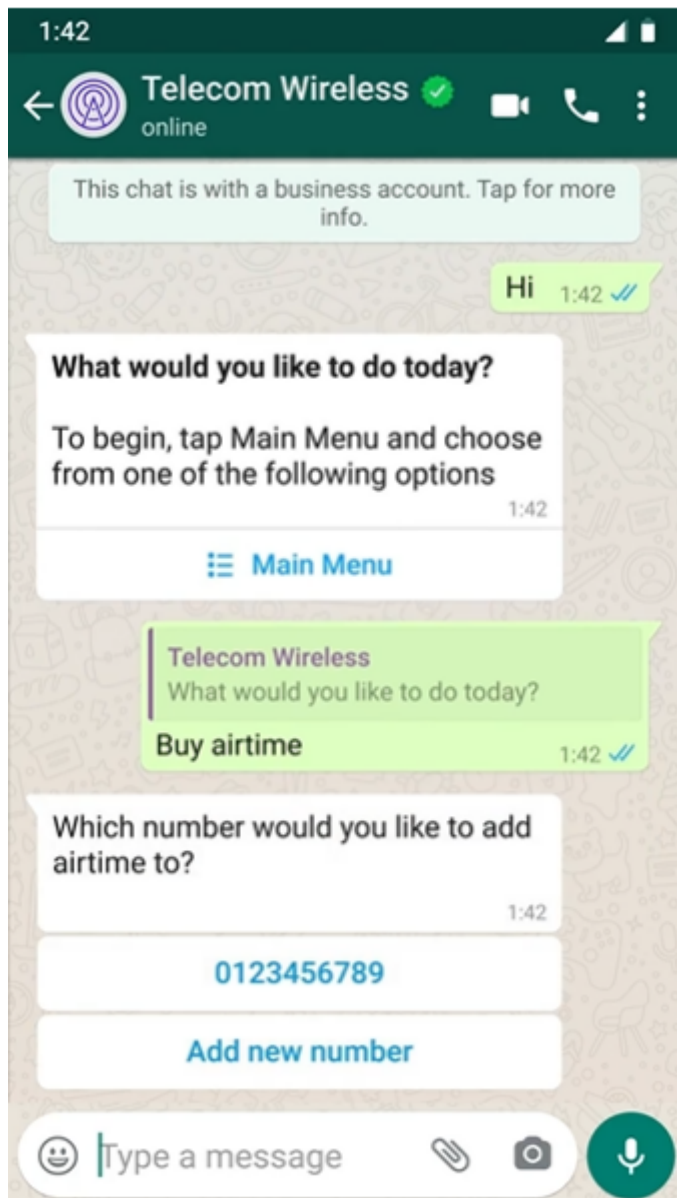
```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendMessageInteractiveList("34605889421
```

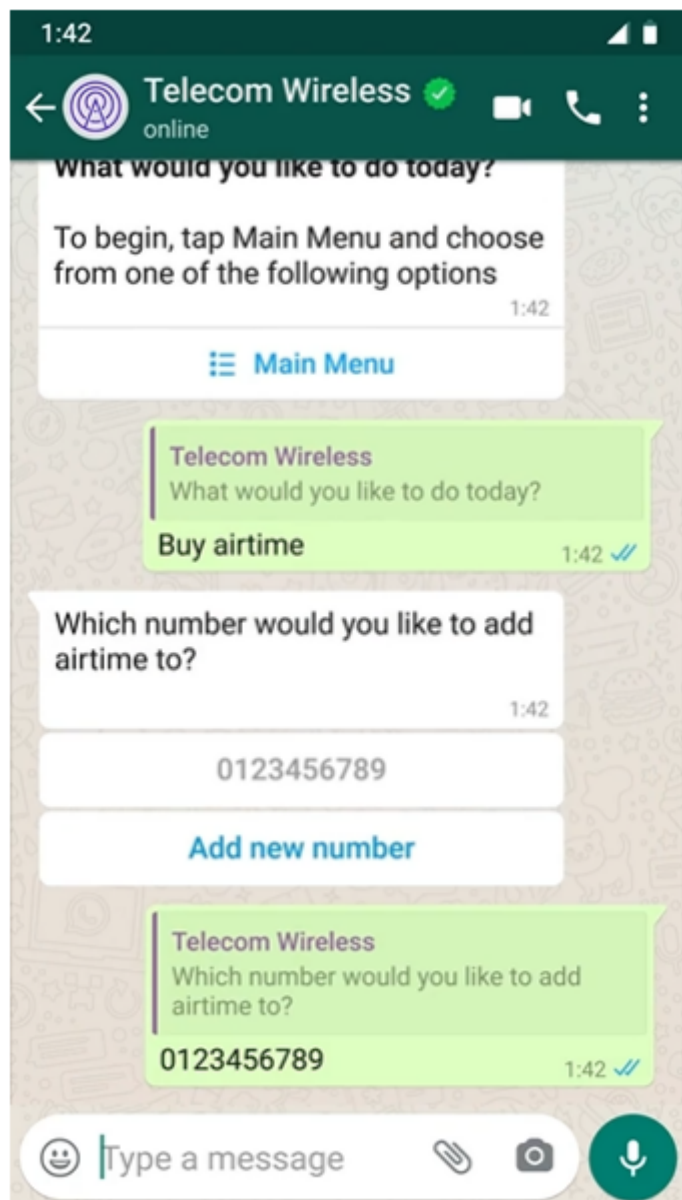
```
", "What would you like to do today?", "To begin, Tap Main Menu and choose from of the following options", "", "M
```



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

SendMessageTemplate 메서드를 호출하고 다음 매개변수를 전달하십시오:

- **aTo:** 전화번호
- **aTemplate:** 템플릿 식별자.
- **aLanguageCode:** 템플릿 언어.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendMessageTemplate("34605889421", "hello_world", "en_US");
```

Template Message Parameters

템플릿에는 매개변수를 포함할 수 있습니다. 아래에서 매개변수가 있는 기본 템플릿의 예제를 확인하십시오.

```
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

아래에서 이미지에 대한 링크를 사용하는 대신, 먼저 이미지를 서버에 업로드한 다음 문서의 Id를 설정하는 템플릿의 예제를 확인하십시오.

```
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

비즈니스가 수신하는 메시지 및 고객 프로필 업데이트에 대한 알림을 받으려면 [Webhooks](#)를 구독하십시오.

트리거 이벤트가 발생할 때마다, WhatsApp Business Platform은 이벤트를 확인하고 이전에 지정한 Webhook URL로 알림을 보냅니다. 두 가지 유형의 알림을 받을 수 있습니다:

- **Received messages:** 이 알림은 메시지를 받았을 때 알려 줍니다.
- **메시지 상태 및 가격 알림:** 이 알림은 메시지의 상태가 변경되었을 때(예: 메시지가 읽히거나 전달됨) 알려줍니다.

Received Messages

새 메시지가 수신될 때마다 **OnMessageReceived** 이벤트가 호출되며, 여기서 메시지의 내용에 액세스하고 메시지를 읽음으로 표시할 수 있습니다.

아래 예제를 참조하십시오. 새 텍스트 메시지가 수신되면, 이를 보낸 사용자에게 그대로 에코됩니다.

```
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;
            }
        }
    }
}
```

Sent Messages

WhatsApp Business Platform은 사용자와 사용자 간 메시지의 상태를 알리기 위해 알림을 보냅니다. 메시지가 성공적으로 전송되면 메시지가 전송, 전달 및 읽힐 때 알림을 받습니다. 앱에서 이러한 알림의 순서는 메시지 상태의 실제 타이밍을 반영하지 않을 수 있습니다. 필요한 경우 타이밍을 확인하려면 타임스탬프를 보십시오.

- **sent:** 비즈니스가 사용자 시작 대화의 일부로 메시지를 보낼 때 다음 알림이 수신됩니다 (해당 대화가 무료 진입점에서 시작되지 않은 경우):
- **delivered:** 다음 알림은 비즈니스의 메시지가 전달되고 해당 메시지가 사용자가 시작한 대화의 일부일 때(해당 대화가 무료 진입점에서 시작되지 않은 경우) 수신됩니다:
- **read:** 사용자가 메시지를 읽으면 다음 알림이 수신됩니다.

새 상태가 수신될 때마다 **OnMessageSent** 이벤트가 호출됩니다.

```
void OnWhatsAppMessageSent(TObject *Sender, const TsgcWhatsApp_Receive_Message *aMessage, TsgcWhatsAppSendMessage *aSendMessage)
{
    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

모든 API 호출은 Access Token으로 인증되어야 합니다. 개발자는 **App Dashboard > WhatsApp > Getting Started**에서 생성된 액세스 토큰으로 API 호출을 인증할 수 있습니다

API 호출은 Message Id를 문자열로 반환합니다.

WhatsApp API를 사용하여 파일을 보낼 때, 먼저 메시지가 WhatsApp 서버에 업로드되고 파일 업로드 후 반환된 객체 id와 함께 새 메시지가 전송됩니다.

Image Messages

SendMessageImage 메서드를 호출하고 다음 매개변수를 전달하십시오:

- **aTo:** 전화번호
- **aFileName:** 보낼 이미지 파일의 전체 파일명(경로 포함)입니다.
- **aFileType:**
 - image/jpeg
 - image/png
- **aCaption:** 이미지의 제목(선택적).

```
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

SendMessageDocument 메서드를 호출하고 다음 매개변수를 전달하십시오:

- **aTo:** 전화번호
- **aFileName:** 보낼 문서 파일의 전체 파일 이름(경로 포함)입니다.
- **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:** 문서의 제목(선택 사항).

```
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

SendMessageAudio 메서드를 호출하고 다음 매개변수를 전달하십시오:

- **aTo:** 전화번호

- **aFileName:** 보낼 오디오 파일의 전체 파일 이름(경로 포함)입니다.
- **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");
```

비디오 메시지

SendMessageVideo 메서드를 호출하고 다음 매개변수를 전달하십시오:

- **aTo:** 전화번호
- **aFileName:** 보낼 비디오 파일의 전체 파일 이름(경로 포함).
- **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");
```

스티커 메시지

SendMessageSticker 메서드를 호출하고 다음 매개변수를 전달하십시오:

- **aTo:** 전화번호
- **aFileName:** 보낼 스티커 파일의 전체 파일 이름(경로 포함)입니다.
- **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

미디어 파일 링크가 있는 메시지를 수신하면 **DownloadMedia** 메서드를 사용하여 미디어 파일을 다운로드할 수 있습니다.

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->DownloadMedia("38923878928822", "c:\\whatsapp\\media\\image.png");
```

이전에 업로드한 미디어 파일을 삭제하려면 **DeleteMedia** 를 호출하고 객체 id를 인수로 전달하기만 하면 됩니다.

API Telegram

Telegram

Telegram은 두 가지 종류의 API를 제공합니다. 하나는 봇과 HTTPs를 프로토콜로 사용하는 프로그램을 만들 수 있게 하는 **Bot API**입니다. **Telegram API**와 **TDLib**는 사용자 정의 Telegram 클라이언트를 빌드할 수 있게 하며 Bot API보다 훨씬 강력합니다.

sgcWebSockets는 **tdjson** 라이브러리를 통해 TDLib를 지원하므로, 자신만의 telegram 클라이언트를 빌드할 수 있습니다. TDLib는 모든 네트워크 구현 세부 사항, 암호화 및 로컬 데이터 저장소를 처리합니다. TDLib는 모든 Telegram 기능을 지원합니다.

TDLib (Telegram Database Library) Advantages

- **크로스 플랫폼:** Windows, Android, iOS, MacOS, Linux 등에서 사용할 수 있습니다...
- **사용 편의성:** 애플리케이션과 telegram 간에 통신하기 위해 json 메시지를 사용합니다.
- **고성능:** Telegram Bot API에서 각 TDLib 인스턴스는 24000개 이상의 봇을 처리합니다.
- **Consistent:** TDLib는 모든 업데이트가 올바른 순서로 전달됨을 보장합니다.
- **Reliable:** TDLib는 느리고 신뢰할 수 없는 인터넷 연결에서도 안정적으로 유지됩니다.
- **Secure:** 모든 로컬 데이터는 사용자가 제공한 암호화 키를 사용하여 암호화됩니다.
- **완전 비동기:** TDLib에 대한 요청은 서로를 차단하지 않습니다. 응답은 사용 가능해질 때 전송됩니다.

구성

Windows

TDLib에는 다른 서드파티 라이브러리가 필요합니다: OpenSSL 및 ZLib. 이 라이브러리들은 tdjson 라이브러리와 함께 배포해야 합니다.

* Windows 버전에는 VCRuntime이 필요하며, Microsoft에서 다운로드할 수 있습니다: <https://www.microsoft.com/en-us/download/details.aspx?id=52685>. 설치 후에도 문제가 지속되면 다음 dll을 애플리케이션이 있는 동일한 폴더에 복사해 보십시오: VCRUNTIME140.dll 및 VCRUNTIME140_1.dll.

다음 라이브러리를 애플리케이션이 위치한 동일한 디렉터리에 복사하십시오:

Windows 32	Windows 64
tdjson.dll	tdjson.dll
libcrypto-1_1.dll	libcrypto-1_1-x64.dll
libssl-1_1.dll	libssl-1_1-x64.dll
zlib1.dll	zlib1.dll

OSX64

libtdjson.dylib 라이브러리를 장치에 배포하면 SetTDJsonPath를 사용하여 라이브러리 위치를 설정할 수 있습니다. 예:

"Contents\MacOS\"에 배포하는 경우, TPath.GetDirectoryName(ParamStr(0)) 폴더에 경로를 설정해야 합니다.

OSXARM64

libtdjson.dylib 라이브러리를 장치에 배포하면 SetTDJsonPath를 사용하여 라이브러리 위치를 설정할 수 있습니다. 예:

"Contents\MacOS\"에 배포하는 경우, TPath.GetDirectoryName(ParamStr(0)) 폴더에 경로를 설정해야 합니다.

Linux64

-

libtdjson.so 라이브러리를 디바이스에 배포하고 SetTDJsonPath 메서드를 호출하여 라이브러리 경로를 설정하십시오.

Android

libtdjsonandroid.so 라이브러리를 장치에 배포하십시오. 예: Android64 라이브러리를 배포하는 경우 Project/Deployment 의 RemotePath 를 "library/lib/arm64-v8a"로 설정하십시오. Android32 인 경우 RemotePath 를 "library/lib/armeabi-v7a"로 설정하십시오

iOS64

libtdjson.a 라이브러리를 다음 디렉터리에 복사하십시오:

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

Telegram 애플리케이션 만들기

API id를 얻고 Telegram API를 사용하여 자체 애플리케이션을 개발하려면 다음을 수행해야 합니다:

- 아무 애플리케이션이나 사용하여 Telegram에 가입하십시오.
- Telegram core에 로그인하십시오: <https://my.telegram.org>.
- **API development tools**로 이동하여 폼을 작성하십시오.
- 사용자 권한 부여에 필요한 기본 주소와 **api_id** 및 **api_hash** 매개변수를 받게 됩니다.
- 현재 각 번호는 하나의 **api_id**만 연결할 수 있습니다.

이 값들은 Telegram 구성 요소의 **Telegram.API** 속성에 설정해야 합니다. 인증하려면 사용자 또는 봇으로 인증할 수 있으며, Telegram에 로그인하기 위해 설정할 수 있는 2개의 속성이 있습니다:

- **PhoneNumber:** 사용자로 로그인하는 경우 **전화번호**(국제 코드 포함)를 설정해야 합니다. 예: +34699123456
- **BotToken:** 봇으로 로그인하는 경우, 이 속성에 토큰을 설정하십시오(telegram이 제공한 대로).
- **DatabaseDirectory:** tdlib 데이터베이스의 위치를 지정할 수 있습니다. 비워 두면 기본 구성이 사용됩니다.

다음 매개변수를 구성할 수 있습니다:

- **ApplicationVersion:** 애플리케이션 버전, 예: 1.0
- **DeviceModel:** 디바이스 모델, 예: desktop
- **LanguageCode:** 사용자 언어 코드, 예: en.
- **SystemVersion:** 운영 체제의 버전, 예: windows.

선택적으로 **SetTDJsonPath** 메서드를 사용하여 tdjson 라이브러리가 위치한 경로를 구성할 수 있습니다. 새 telegram 세션 을 시작하기 전에 경로를 전달하기만 하면 됩니다.

Telegram 구성 요소를 구성한 후, Active 속성을 true로 설정하면 프로그램이 Telegram에 연결을 시도합니다.

샘플 코드

```
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;
```

권한 부여

인증 코드(SMS가 아닌 Telegram 애플리케이션에서 전달됨)를 가져오거나 비밀번호를 제공하기 위해 라이브러리에서 호출할 수 있는 두 가지 이벤트가 있습니다.

OnAuthenticationCode

이 이벤트는 Telegram이 Telegram 애플리케이션에 Authorization Code를 보내고 사용자가 이 코드를 복사하여 이 이벤트의 Code 인수에 설정해야 할 때 호출됩니다.

```
void OnAuthenticationCode(TObject *Sender, ref string Code)
{
    Code = InputBox("Telegram Code", "Introduce code", "");
}
```

OnAuthenticationPassword

이 이벤트는 Telegram이 사용자가 비밀번호를 설정하도록 요구할 때 호출됩니다.

권한 부여 상태

권한 부여가 시작되면 **OnAuthorizationStatus** 이벤트로 권한 부여 상태를 확인할 수 있으며, 이 이벤트는 권한 부여 상태가 변경될 때마다 호출됩니다. Status의 일부 값은 다음과 같습니다:

- authorizationStateWaitTdlibParameters
- authorizationStateWaitEncryptionKey
- authorizationStateWaitPhoneNumber
- authorizationStateWaitCode
- authorizationStateLoggingOut
- authorizationStateClosed
- authorizationStateReady

연결 상태

연결이 시작되면 **OnConnectionStatus** 이벤트로 연결 상태를 확인할 수 있습니다. 이 이벤트는 연결 상태가 변경될 때마다 호출됩니다. Status의 일부 값은 다음과 같습니다:

- connectionStateConnecting
- connectionStateUpdating
- connectionStateReady

주요 메서드

TsgcTDLib_Telegram API 구성 요소는 여러 Telegram 메서드를 지원합니다. 아래에서 가장 많이 사용되는 것을 확인하십시오.

메서드	매개변수	Description
SendTextMessage	aChatId: 메시지가 전송될 Chat의 Id aText: Message의 텍스트. InlineKeyboard: 선택적 버튼(보만 해당).	Chat에 텍스트 메시지를 보냅니다.
SendRichTextMessage	aChatId: 메시지가 전송될 Chat의 Id입니다 aText: 메시지 텍스트입니다. InlineKeyboard: 선택적 버튼(보만 해당).	채팅에 Rich Text 메시지를 보냅니다. Markdown 구문: • Bold: **bold** • Italic: <i>__italic__</i> • Strike: ~~--strike--~~ • Underline: <u>~~underline~~</u> • Code: <code>##code##</code>

SendDocumentMessage	aChatId: 메시지를 보낼 Chat의 Id입니다 aFilePath: 문서의 전체 파일 경로입니다 aInlineKeyboard: 선택적 Buttons(봇만 해당).	Chat에 문서를 보냅니다.
SendPhotoMessage	aChatId: 메시지가 전송될 채팅의 Id aFilePath: 사진의 전체 파일 경로 Width: 사진의 너비. Height: 사진의 너비. InlineKeyboard: 선택적 버튼(봇만 해당).	Chat에 Photo를 보냅니다.
SendVideoMessage	aChatId: 메시지를 보낼 Chat의 Id aFilePath: 비디오의 전체 파일 경로 aWidth: 비디오 너비. Height: 비디오의 너비입니다. aDuration: 비디오 길이(초). aInlineKeyboard: 선택적 버튼(봇 전용).	Chat에 비디오를 보냅니다.
SendInvoiceMessage	aChatId: 메시지가 전송될 Chat의 Id aInvoice: 메시지 텍스트. aInlineKeyboard: 선택적 버튼(봇 전용).	Invoice를 보냅니다(Bot이고 Private Channel인 경우에만 사용 가능).
EditTextMessage	aChatId: 메시지가 전송될 Chat의 Id입니다. aMessageId: 수정할 메시지의 Id Text: 메시지 텍스트입니다. InlineKeyboard: 선택적 버튼(봇만 해당). ShowKeyboard: 선택적 버튼(봇 전용).	메시지의 텍스트(또는 game 메시지의 텍스트)를 편집합니다
AddChatMember	aChatId: 메시지를 보낼 Chat의 Id입니다 aUserId: 사용자의 식별자입니다. aForwardLimit: 새 멤버에게 전달될 채팅의 이전 메시지 수입니다 (최대 100). supergroup 및 채널의 경우 무시됩니다.	채팅에 새 멤버를 추가합니다. 멤버는 비공개 또는 비밀 채팅에 추가될 수 없습니다. 채팅 상태가 서버와 동기화될 때까지 멤버가 추가되지 않습니다.
AddChatMembers	aChatId: 메시지가 전송될 Chat의 Id aUserIds: 채팅에 추가될 사용자의 식별자.	채팅에 여러 새 멤버를 추가합니다. 현재 이 옵션은 슈퍼그룹 및 채널에서만 사용할 수 있습니다. 이 옵션은 채팅에 참여하는 데 사용할 수 없습니다. 채널에 200명 이상의 멤버가 있으면 멤버를 추가할 수 없습니다. 채팅 상태가 서버와 동기화될 때까지 멤버가 추가되지 않습니다.
GetChatMember	aChatId: Chat 식별자입니다. aUserId: User 식별자입니다.	채팅의 단일 멤버에 대한 정보를 반환합니다.
GetBasicGroupFullInfo	aGroupId: Basic Group Identifier	식별자로 기본 그룹에 대한 전체 정보를 반환합니다.
GetSupergroupMembers	aSuperGroupId: supergroup 또는 채널의 식별자입니다. aSupergroupMembersFilter: 반환할 사용자 유형입니다. 기본값은 null aOffset: 건너뛴 사용자 수입니다. aLimit: 반환할 최대 사용자 수입니다. 최대 200까지 가능합니다.	supergroup 또는 channel의 멤버 또는 차단된 사용자에 대한 정보를 반환합니다.

JoinChatByInviteLink	aLink: 가져올 초대 링크입니다.	가능한 경우 초대 링크를 사용하여 현재 사용자를 채팅에 추가합니다. 채팅 상태가 서버와 동기화될 때까지 새 멤버는 추가되지 않습니다.
CreateNewSecretChat	aUserId: 사용자의 식별자입니다.	새 비밀 채팅을 생성합니다.
CreateNewBasicGroupChat	aUserIds: 채팅에 추가할 사용자의 식별자입니다. aTitle: 새 기본 그룹의 제목입니다	새 기본 그룹을 생성합니다
CreateNewSupergroupChat	aTitle: 새 SuperGroup의 제목 alsChannel: True인 경우 채널 채팅이 생성되어야 합니다. aDescription: 채팅 설명.	새 슈퍼그룹 또는 채널을 생성합니다.
CreatePrivateChat	aUserId: 사용자의 식별자. aForce: true이면 네트워크 요청 없이 채팅이 생성됩니다. 이 경우 유형, 제목 및 사진을 제외한 채팅에 대한 모든 정보가 부정확할 수 있습니다	지정된 사용자에게 해당하는 기존 채팅을 반환합니다
GetChats	aOffsetOrder: 채팅을 반환할 채팅 순서 aOffsetChatId: 채팅을 반환할 채팅 식별자 aLimit: 반환할 최대 채팅 수입니다.	정렬된 채팅 목록을 반환합니다. 채팅은 (order, chat_id) 쌍을 기준으로 내림차순으로 정렬됩니다(보통으로 로그인한 경우 사용할 수 없음).
GetChat	aChatId: Chat 식별자	식별자로 chat에 대한 정보를 반환합니다
GetChatHistory	aChatId: 채팅 식별자 aFromMessageId: 기록을 가져오기 시작할 메시지의 식별자입니다. 마지막 메시지부터 결과를 얻으려면 0을 사용하십시오. aOffset: from_message_id에서 정확히 결과를 얻으려면 0을 지정하거나, 추가로 일부 새 메시지를 얻으려면 최대 99까지의 음수 offset을 지정하십시오. aLimit: 반환할 최대 메시지 수입니다	채팅의 메시지를 반환합니다. 메시지는 역시간순으로 반환됩니다
GetUser	aUserId: 사용자 식별자	식별자로 사용자에게 대한 정보를 반환합니다.
AddProxyHTTP	aServer: 프록시의 서버 이름입니다. aPort: 프록시 포트 번호입니다. aUserName: 로그인용 사용자 이름이며, 비어 있을 수 있습니다. aPassword: 로그인 비밀번호입니다. 비어 있을 수 있습니다. aHTTPOnly: 프록시가 HTTP 요청만 지원하고 HTTP CONNECT 메시지를 통한 투명한 TCP 연결을 지원하지 않는 경우 true를 전달하십시오.	네트워크 요청에 대한 HTTP 프록시 서버를 추가합니다. 인증 전에 호출할 수 있습니다.
AddProxyMTProto	aServer: 프록시의 서버 이름입니다. aPort: 프록시 포트 번호. aSecret: 16진수 인코딩으로 된 프록시의 시크릿.	네트워크 요청을 위한 MTProto 프록시 서버를 추가합니다. 권한 부여 전에 호출할 수 있습니다.
AddProxySocks5	aServer: 프록시의 서버 이름입니다. aPort: 프록시 포트 번호입니다.	네트워크 요청에 Socks5 프록시 서버를 추가합니다.

	aUserName: 로그인용 사용자 이름이며, 비어 있을 수 있습니다. aPassword: 로그인용 비밀번호입니다. 비어 있을 수 있습니다.	권한 부여 전에 호출할 수 있습니다.
EnableProxy	aid: 프록시의 ID입니다.	프록시를 활성화합니다. 한번에 하나의 프록시만 활성화할 수 있습니다. 권한 부여 전에 호출할 수 있습니다.
DisableProxy		현재 활성화된 프록시를 비활성화합니다. 권한 부여 전에 호출할 수 있습니다.
RemoveProxy	aid: 프록시의 ID입니다.	프록시 서버를 제거합니다. 권한 부여 전에 호출할 수 있습니다.
GetProxies		현재 설정된 proxy 목록을 반환합니다. 권한 부여 전에 호출할 수 있습니다.
getChatSponsoredMessage	aChatId: 채팅의 ID입니다.	채팅에 표시될 sponsored 메시지를 반환합니다. 채널 채팅 전용입니다. 채팅에 sponsored 메시지가 없으면 404 오류를 반환합니다.
ViewMessage	aSponsorChatId: 스폰서 Chat의 ID aMessageId: 메시지의 ID	사용자가 메시지를 보고 있음을 TDLib에 알립니다. 많은 유용한 활동은 메시지가 현재 보고 있는지 여부에 따라 달라집니다
Logout		Telegram에서 로그아웃합니다.
TDLibSend	aRequest: JSON Request.	JSON 프로토콜로 모든 Request를 보냅니다.

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");
```

구현되지 않은 메서드를 보내는 방법의 예

TDLibSend 메서드를 사용하여 모든 JSON 메시지를 보낼 수 있습니다(예: telegram 채팅 참가).

```
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\"}");
```

Events

OnBeforeReadEvent

이 이벤트는 Telegram API 구성 요소에 의해 JSON 메시지가 수신되었지만 아직 처리되지 않았을 때 호출됩니다. 이 이벤트를 수동으로 처리하거나 구성 요소가 이벤트를 처리하지 않게 하려면 `Handled` 속성을 `True`로 설정하십시오. 이 이벤트를 사용하여 모든 메시지를 로깅할 수도 있습니다.

OnMessageText

이 이벤트는 새 메시지 텍스트가 수신되었을 때 호출됩니다. 메시지 텍스트 속성에 액세스하려면 `MessageText` 매개변수를 읽으십시오.

- **ChatId:** Chat 식별자.
- **MessageId:** Message 식별자입니다.
- **SenderId:** 송신자 식별자입니다.
- **Text:** 메시지의 텍스트입니다.

OnMessageDocument

새 Document Message가 수신될 때 이 이벤트가 호출됩니다. Document 속성에 액세스하려면 `MessageDocument`에 액세스하십시오.

- **ChatId:** Chat 식별자.
- **MessageId:** Message 식별자입니다.
- **SenderId:** 송신자 식별자입니다(tdlib 1.7.+부터 `SenderChat` 및 `SenderUser`를 읽습니다).
- **FileName:** Document의 이름입니다.
- **DocumentId:** Document Identifier.
- **LocalPath:** 존재하는 경우 로컬 파일의 전체 경로입니다.
- **MimeType:** 문서의 Mime-type입니다.
- **Size:** 문서의 크기.
- **RemoteDocumentId:** Remote Document Identifier입니다.

OnMessagePhoto

이 이벤트는 새 사진 메시지가 수신될 때 호출됩니다. 사진 속성에 접근하려면 `MessagePhoto`에 접근하십시오.

- **ChatId:** Chat 식별자.
- **MessageId:** Message 식별자입니다.
- **SenderId:** 송신자 식별자입니다(tdlib 1.7.+부터 `SenderChat` 및 `SenderUser`를 읽습니다).
- **PhotoId:** 사진 식별자.
- **LocalPath:** 존재하는 경우 로컬 파일의 전체 경로입니다.
- **Size:** 사진의 크기입니다.
- **RemotePhotoId:** 원격 사진 식별자.

OnVideoPhoto

이 이벤트는 새 Video Message가 수신될 때 호출됩니다. Video 속성에 액세스하려면 `MessageVideo`에 액세스하십시오.

- **ChatId:** Chat 식별자.
- **MessageId:** Message 식별자입니다.
- **SenderId:** 송신자 식별자입니다(tdlib 1.7.+부터 `SenderChat` 및 `SenderUser`를 읽습니다).
- **VideoId:** 사진 식별자입니다.
- **LocalPath:** 존재하는 경우 로컬 파일의 전체 경로입니다.
- **Width:** 비디오의 너비입니다.
- **Height:** 비디오의 높이입니다.
- **Duration:** 비디오의 지속 시간(초).
- **Size:** 비디오의 크기입니다.
- **RemoteVideoId:** 원격 사진 식별자입니다.

OnMessageSponsored

이 이벤트는 새 Sponsored Message가 수신되었을 때(`getChatSponsoredMessage` 메서드 호출 후) 호출됩니다.

- **SponsorChatId:** 스폰서 채팅 식별자입니다.
- **MessageId:** Message 식별자입니다.
- **Text:** 메시지의 텍스트입니다.

OnNewChat

이 이벤트는 새 채팅이 수신될 때 호출됩니다.

- **ChatId:** Chat 식별자.
- **ChatType:** 채팅 유형(chatTypeSupergroup, chatTypePrivate...)
- **Title:** 채팅 이름.
- **SuperGroupId:** SuperGroup인 경우 Group Id.
- **IsChannel:** 채널인지 여부를 반환합니다.

OnNewCallbackQuery

이 이벤트는 새 콜백 쿼리가 수신될 때 호출됩니다. 봇 전용입니다.

- **Id:** 고유 쿼리 식별자입니다.
- **SenderUserId:** 쿼리를 보낸 사용자의 식별자입니다.
- **ChatId:** 쿼리가 전송된 채팅의 식별자입니다.
- **MessageId:** 쿼리가 발생한 메시지의 식별자입니다.
- **ChatInstance:** 메시지가 전송된 채팅에 고유하게 해당하는 식별자입니다.
- **PayloadData:** 일반 콜백 버튼의 페이로드.
 - **Data:** 콜백 버튼에 첨부된 데이터.

OnEvent

이 이벤트는 API 구성 요소가 새 이벤트를 받을 때 호출됩니다. API 구성 요소에서 구현하지 않은 일부 이벤트를 처리하는 데 사용할 수 있습니다.

- **Event:** 이벤트 이름(updateOption, updateUser 등과 같은 이벤트)
- **Text:** 전체 JSON 메시지

OnException

이 이벤트는 Telegram API 데이터를 처리할 때 예외가 있는 경우 호출됩니다.

Properties

MyId: 현재 사용자의 User Identifier를 반환합니다.

전체 코드 샘플

Telegram API에 연결하는 방법, 사용자에게 Code를 입력하도록 요청하는 방법(Telegram API에서 요구하는 경우), 연결이 준비되면 메시지를 보내고 수신된 텍스트 메시지를 로깅하는 방법을 보여주는 다음 코드 샘플을 확인하십시오.

```
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 | 인라인 버튼이 있는 Telegram 메시지 보내기

Telegram API는 옵션을 답변으로 선택하기 위한 인라인 버튼이 있는 메시지를 보낼 수 있게 합니다(이 옵션은 봇에만 사용 가능합니다).

메시지를 보내기 전에 **TsgcTelegramReplyMarkupInlineKeyboard** 클래스의 인스턴스를 생성하고 생성하려는 각 버튼에 대해 **AddButtonTypeCallback** 또는 **AddButtonTypeUrl** 메서드를 호출하십시오.

예제

메시지를 좋아하는지 여부와 설문 조사에 답할 링크를 사용자에게 묻는 새 메시지를 생성합니다. **OnNewCallbackQuery** 이벤트를 사용하여 응답을 처리하십시오.

```
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

Telegram API를 사용하면 사용자에게 데이터를 요청하는 버튼이 있는 메시지를 보낼 수 있습니다(이 옵션은 봇에서만 사용할 수 있습니다).

메시지를 보내기 전에 **TsgcTelegramReplyMarkupShowKeyboard** 클래스의 인스턴스를 생성하고, 생성하려는 모든 버튼에 대해 **AddButtonTypeRequestLocation**, **AddButtonTypeRequestPhoneNumber** 또는 **AddButtonTypeText** 메서드를 호출하십시오.

예제

사용자에게 PhoneNumber 제공을 요청하는 새 메시지를 생성합니다

```
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

굵게, 기울임꼴 등의 스타일을 사용하여 텍스트 메시지를 강조할 수 있습니다. 스타일 기능이 있는 텍스트 메시지를 보내려면 **SendRichTextMessage** 메서드를 사용하십시오. 이 메서드는 텍스트 메시지를 파싱하고 API Telegram에 필요한 엔터티를 자동으로 추가합니다.

Markdown Syntax

- Bold [*]

```
**This is Bold**
```

- Italic [_]

```
__This is Italic__
```

- Strike [-]

```
--This is Strike--
```

- 밑줄 [~]

```
~~This is Underline~~
```

- Code [#]

```
##This is Monospace##
```

Telegram | Chat not found as Bot

봇으로 로그인하면 GetChats 메서드를 사용할 수 없으므로 사용 가능한 모든 chat을 가져오지 못합니다. 봇으로 처음 로그인하고 알려진 Chat에 메시지를 보내려고 하면 다음 오류가 발생합니다:

```
{"@type":"error","code":5,"message":"Chat not found"}
```

해결책은 telegram 메시지를 보내기 전에 **GetChat** 메서드를 호출하고 **ChatId**를 매개변수로 전달하는 것입니다. Chat 데이터를 얻으면 평소처럼 telegram 메시지를 보낼 수 있습니다.

참고로, 이 채팅에서 봇 메시지를 받은 적이 없는 경우 메시지를 보내기 전 처음에만 **GetChat**을 호출하면 됩니다. 애플리케이션을 닫고 다시 시작하면 Chat이 이미 telegram 데이터베이스에 저장되어 있으므로 먼저 GetChat을 호출할 필요가 없습니다.

Telegram | Sponsored Messages

사용자가 채널을 열 때마다, 이 채널에 사용 가능한 sponsored 메시지를 받기 위해 `channels.getSponsoredMessages`를 호출해야 합니다. 결과는 5분 동안 캐시되어야 합니다.

스폰서 메시지 표시

스폰서 메시지는 사용자가 마지막 메시지를 지나 더 아래로 스크롤한 후 채널의 다른 모든 게시물 아래에 표시되어야 합니다. `from_id` 필드에 지정된 홍보 채널 또는 봇이 메시지 작성자로 표시되어야 합니다. 메시지에는 또한 하단에 다음 버튼 중 하나가 포함되어야 합니다:

- **View Bot:** 봇이 홍보되는 경우입니다. 버튼을 탭하면 봇과의 채팅이 열려야 합니다. `start_param`이 지정되면 앱은 딥 링크 메커니즘을 사용하여 봇을 열어야 합니다.
- **View Channel:** 채널이 홍보되는 경우입니다. 버튼을 탭하면 채널이 열려야 합니다.
- **View Post:** 채널이 홍보 중이고 `channel_post`가 지정된 경우입니다. 버튼을 탭하면 특정 채널 게시물이 열려야 합니다.

전체 텍스트가 화면에 표시되면(버튼 제외), 이 sponsored message의 `random_id`로 **ViewMessage** 메서드를 호출해야 합니다.

Get Sponsored Messages

채널에 스폰서 메시지가 있는지 묻는 요청을 보내려면 **GetChatSponsoredMessage** 메서드를 호출하기만 하면 됩니다.

```
TsgcTDLlib_Telegram *oTelegram = new TsgcTDLlib_Telegram();
oTelegram->Telegram->API->ApiHash = "ABCDEFGH IJKLMN";
oTelegram->Telegram->API->ApiId = "1234";
oTelegram->PhoneNumber = "008745744155";
oTelegram->Active = true;
oTelegram->getChatSponsoredMessage("100");
```

채팅에 sponsored 메시지가 있으면, Sponsored 메시지의 내용과 함께 **OnMessageSponsored** 이벤트가 호출됩니다. 메시지가 없으면 404 오류가 반환됩니다.

```
private void(TObject *Sender, TsgcTelegramMessageSponsored *MessageSponsored)
{
    DoLog(MessageSponsored->Text);
}
```

Sponsored Messages가 사용자에게 표시된 후 **ViewMethod** 메서드를 호출하십시오.

```
oTelegram->ViewMessage("100", "54653256245");
```

Telegram | Send Telegram Invoice Message

봇이 inline 모드를 지원하는 경우, 사용자는 봇을 통해 다른 채팅(다른 사용자와의 일대일 채팅 포함)으로 인보이스를 보낼 수 있습니다.

Invoice 메시지는 제품의 사진과 설명, 그리고 눈에 띄는 Pay 버튼을 특징으로 합니다. 이 버튼을 탭하면 Telegram 앱에서 특수 결제 인터페이스가 열립니다

bot은 **SendInvoiceMessage** 메서드를 사용하여 invoice를 메시지로 보낼 수 있습니다.

```
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

Telegram API를 사용하면 SuperGroup의 멤버에 대한 정보를 가져올 수 있습니다. supergroup 또는 채널의 멤버 또는 차단된 사용자에 대한 정보를 가져오려면 **GetSuperGroupMembers** 메서드를 사용하십시오. SupergroupFullInfo.can_get_members가 true인 경우에만 사용할 수 있습니다. 또한 일부 필터에는 관리자 권한이 필요할 수 있습니다.

기본적으로 이 메서드는 그룹의 모든 멤버를 반환하지만, Filter 매개변수를 사용하여 반환되는 멤버를 필터링할 수 있습니다. 다음 매개변수가 있습니다:

tsgmFilterNone

기본값으로, 멤버가 필터링되지 않음을 의미합니다.

tsgmFilterAdministrators

생성자와 관리자를 반환합니다.

tsgmFilterBanned

슈퍼그룹 또는 채널에서 차단된 사용자를 반환합니다. 관리자만 사용할 수 있습니다. aSuperGroupMembersQuery 인수를 사용하여 쿼리로 검색할 수 있습니다.

tsgmFilterBots

슈퍼그룹 또는 채널의 봇 멤버를 반환합니다.

tsgmFilterContacts

supergroup 또는 channel의 멤버인 사용자의 연락처를 반환합니다. aSuperGroupMembersQuery 인수를 사용하여 쿼리로 검색할 수 있습니다.

tsgmFilterMention

슈퍼그룹에서 멘션할 수 있는 사용자를 반환합니다.

tsgmFilterRecent

최근 활성 사용자를 역순(최신순)으로 반환합니다.

tsgmFilterRestricted

제한된 supergroup 멤버를 반환합니다. 관리자만 사용할 수 있습니다. aSuperGroupMembersQuery 인수를 사용하여 쿼리로 검색할 수 있습니다.

tsgmFilterSearch

(문자열) 쿼리를 통해 슈퍼그룹 또는 채널 멤버를 검색하는 데 사용됩니다. aSuperGroupMembersQuery 인수를 사용하여 쿼리로 검색할 수 있습니다.

event = "chatMembers"로 필터링하여 OnEvent 콜백을 사용해 결과를 읽을 수 있습니다.

```
Telegram->GetSupergroupMembers(1452979380);
```

```
private void OnTelegramEvent(TObject *Sender, const string Event, const string Text)
{
    if (Event == "chatMembers")
    {
        ReadJSON(Text);
    }
}
```

Telegram | Add Telegram Proxy

Telegram Client는 프록시를 사용하도록 구성할 수 있습니다. 현재 Telegram은 3가지 유형의 프록시를 지원합니다:

1. HTTP
2. MTProto
3. Socks5

프록시 추가

HTTP Proxy를 구성하려면, 먼저 telegram 구성에 프록시를 추가해야 합니다. 이렇게 하려면 **AddProxyHTTP**를 호출하고 성공하면 새 프록시가 추가되었다는 메시지가 반환됩니다. 프록시가 목록에 추가되면, **EnableProxy**를 호출하고 확인 메시지에서 수신된 **프록시의 ID**를 전달하기만 하면 됩니다.

```
Telegram->AddProxyHTTP("8.8.8.8", 8080, "", "", true);  
// ... read the confirmation message and save the ID of the proxy.  
Telegram->EnableProxy(2);
```

Remove Proxy

RemoveProxy 메서드를 호출하고 제거하려는 프록시의 ID를 전달하십시오.

Telegram | Register Telegram User

Telegram에서 새 사용자를 등록하는 프로세스는 매우 간단합니다. API Id, API Hash 및 새 계정의 전화번호가 필요합니다.

telegram 클라이언트를 구성하십시오:

- API Id
- API Hash
- 새 telegram 계정의 전화번호입니다.

클라이언트를 시작하면 새 코드가 전화로 전송되고, 클라이언트가 telegram 코드를 요청하며, 코드가 올바르면 OnRegisterUser 이벤트가 호출됩니다. 이 이벤트에서 사용자의 First Name과 Last Name을 설정하면 등록이 완료됩니다.

```
TsgcTDLib_Telegram *oTelegram = new TsgcTDLib_Telegram();
oTelegram->Telegram->API->ApiHash = "ABCDEFGHJKLMN";
oTelegram->Telegram->API->ApiId = "1234";
oTelegram->PhoneNumber = "008745744155";
oTelegram->Active = true;

void OnTelegramAuthenticationCode(TObject *Sender, ref string Code)
{
    Code = "code sent to phone";
}

void OnTelegramRegisterUser(TObject *Sender, ref string FirstName, ref string LastName)
{
    FirstName = "first name";
    LastName = "last name";
}
```

RCON

RCON

Source RCON Protocol은 Source Dedicated Server에서 사용하는 TCP/IP 기반 통신 프로토콜로, "원격 콘솔" 또는 RCON을 통해 서버에 콘솔 명령을 발행할 수 있도록 합니다. RCON의 가장 일반적인 용도는 서버 소유자가 서버가 실행 중인 머신에 직접 접근하지 않고도 게임 서버를 제어할 수 있도록 하는 것입니다.

구성

RCON_Options를 사용하면 다음 속성을 구성할 수 있습니다:

- **Host:** 서버 원격 주소입니다.
- **Port:** 서버 수신 대기 포트입니다.
- **Password:** 서버에 대해 인증하는 데 사용되는 비밀 문자열입니다

Connect

서버에서 연결 / 연결 해제하려면 **Active** 속성을 사용하십시오.

Active가 True로 설정되면 클라이언트는 서버에 연결을 시도하며, 연결할 수 있으면 제공된 비밀번호를 사용하여 인증을 시도합니다.

서버는 인증 요청에 응답을 보냅니다. **OnAuthenticate** 이벤트가 호출되며 Authenticate 매개변수를 사용하여 인증이 성공했는지 여부를 읽을 수 있습니다.

명령 보내기

서버에 명령을 보내려면 **ExecCommand** 메서드를 사용하십시오. 응답은 **OnResponse** 이벤트에서 사용할 수 있습니다.

```
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

CryptoHopper는 Bitcoin, Ethereum, Litecoin 등에 대한 거래 및 포트폴리오 관리를 자동화할 수 있는 자동화된 암호화폐 거래 봇입니다.

구성

개발자 계정이 필요하며, 승인을 받으면 새 App을 만들기 시작할 수 있습니다. API는 OAuth2를 사용하여 인증하므로 App에서 **client_id**와 **client_secret**을 검색할 수 있습니다.

```
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";
```

Methods

CryptoHopper는 API에 요청을 보내는 프로토콜로 HTTPs를 사용합니다. 일부 메서드는 인증이 필요하며(주문 발주, 사용자 데이터 조회 등) 일부는 공개입니다(예: 거래소 데이터 가져오기).

함수는 CryptoHopper 응답을 반환하며 오류가 있으면 예외가 발생합니다.

Hoppers

기본 Hopper 작업을 관리합니다.

메서드	Arguments	설명
GetHoppers		사용자의 Hopper를 가져옵니다.
CreateHopper	aBody: 구성 json 텍스트입니다.	새 Hopper를 생성합니다.
GetHopper	aid: hopper id	Retrieve Hopper
DeleteHopper	aid: hopper id	Delete Hopper
UpdateHopper	aid: hopper id aBody: 구성 json 텍스트.	Update Hopper

Orders

Hopper의 주문을 관리합니다.

메서드	Arguments	설명

GetOpenOrders	ald: hopper id	hopper의 모든 미결제 주문을 검색합니다.
CreateNewOrder	ald: hopper id aOrder: TsgcHTTPCTHOrder의 인스턴스	새 매수 또는 매도 주문을 생성합니다. 매도의 경우 sell 엔드포인트를 사용합니다.
PlaceMarketOrder	ald: hopper id aOrderSide: cthosBuy 또는 cthosSell. aCoin: 코인 이름, 예: EOS aAmount: 주문 크기.	시장가 주문을 넣습니다.
PlaceLimitOrder	ald: hopper id aOrderSide: cthosBuy 또는 cthosSell. aCoin: 코인 이름, 예: EOS aAmount: 주문 크기. aPrice: limit 가격.	지정가 주문을 넣습니다.
DeleteOrder	ald: hopper id aOrderId: order id	선택한 hopper의 주문을 삭제합니다.
DeleteAllOrders	ald: hopper id	선택한 hopper의 모든 미결제 주문을 삭제합니다.
GetOpenOrder	ald: hopper id aOrderId: order id	id로 hopper의 미결제 주문을 가져옵니다.
CancelOrder	ald: hopper id aOrderId: order id	미결제 주문을 취소합니다.

Position

Hopper의 Positions를 관리합니다.

메서드	Arguments	설명
GetPosition	ald: hopper id	hopper의 미결제 포지션을 가져옵니다.

Trade

Hopper의 거래 내역.

메서드	Arguments	설명
GetTradeHistory		hopper의 거래 내역을 가져옵니다.

GetTradeHistoryById	aId: hopper id aTradeId: trade id	hopper의 id로 거래를 가져옵니다.

Exchange

CryptoHopper에서 사용 가능한 익스체인지 정보 가져오기

메서드	Arguments	설명
GetExchange		Cryptohopper에서 사용 가능한 모든 익스체인지를 가져옵니다.
GetAllTickers	aExchange: 익스체인지 이름	모든 페어의 ticker를 가져옵니다
GetMarketTicker	aExchange: exchange 이름 aPair: pair 이름	market 쌍에서 ticker를 가져옵니다.
GetOrderBook	aExchange: exchange 이름 aPair: pair 이름 aDepth: 주문서 깊이	선택된 거래소, 마켓 및 오더북 depth에 대한 오더북을 가져옵니다.

Webhooks

Hopper의 거래 내역.

메서드	Arguments	설명
CreateWebhook	aURL: webhook url aMessageTypes: 심표로 구분된 메시지 유형.	Webhook을 업데이트하거나 생성합니다
DeleteWebhook	aURL: webhook url	기존 Webhook을 삭제합니다.

Signals

CryptoHopper API에 Signal을 보냅니다.

메서드	Arguments	설명
SendSignal	aSignal: 신호를 전송하는 데 필요한	Signal을 보냅니다

	모든 필드가 있는 클래스입니다.	
SendTestSignal	aSignal: 신호를 전송하는 데 필요한 모든 필드가 있는 클래스입니다.	테스트 신호를 보냅니다
GetSignalStats	aSignalId: signal의 id입니다. aExchange: 선택 사항, exchange의 이름입니다.	일부 신호 통계를 검색합니다.

Cryptohopper Config를 업데이트하는 방법

Hopper 구성을 업데이트하려면 UpdateHopper 메서드를 사용하십시오. 이 메서드는 오버로드되어 있으므로 JSON 문자열을 전달하거나 TsgcHTTPCTHopper 객체를 사용하고 속성을 사용하여 Hopper 속성을 활성화하거나 비활성화할 수 있습니다.

```
public string EnableHopper()
{
    TsgcHTTPCTHopper *oHopper = new TsgcHTTPCTHopper();
    try
    {
        oHopper->Enabled = 1;
        result = Cryptohopper->UpdateHopper("1234", oHopper);
    }
    __finally
    {
        oHopper->Free();
    }
}
```

Webhook 구성 방법

Webhook을 사용하면 hopper에서 어떤 일이 발생할 때 알림을 받을 수 있습니다. Webhook에는 cryptohopper가 보내는 모든 메시지를 URL 주소에서 수신할 공개 HTTPs 서버가 필요합니다. 공개 서버는 SSL 인증서로 보호되어야 합니다(자체 서명된 인증서는 허용되지 않습니다).

먼저 webhook을 생성해야 하므로, 서버가 수신 대기할 Host와 Port를 설정하여 Cryptohopper 클라이언트의 Webhook 속성을 구성하십시오. 그런 다음 SSLOptions 속성에서 인증서를 구성하십시오.

예제: 공인 IP 주소는 1.1.1.1이고 수신 포트는 443입니다. 인증서는 비밀번호 없이 sgc.pem 파일 이름의 PEM 파일로 저장됩니다.

```
/* 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

RTCMultiConnection은 peer-to-peer 애플리케이션(화면 공유, 오디오/비디오 회의, 파일 공유, 미디어 스트리밍 등)을 위한 WebRTC JavaScript 라이브러리입니다.

구성

RTCMultiConnection은 Signaling 을 위해 WebSocket 서버가 필요하므로, RTCMultiConnection의 server 속성을 WebSocket 서버(예: [TsgcWebSocketHTTPServer](#))에 연결하십시오. 아래에서 구성해야 하는 속성을 확인하십시오.

Server

Host: WebSocket 서버의 공인 IP 주소 또는 DNS 이름입니다.
Port: WebSocket 서버의 수신 대기 포트입니다.

IceServers

피어 간 통신을 허용하기 위한 ICE 서버 (STUN/TURN)의 구성입니다. 예:

```
[
  {
    "urls": "stun:www.yourstun.com"},
  {
    "urls": "turn:www.yourturn.com",
    "username": "user",
    "credential": "secret"
  }
]
```

VideoResolution

여기에서 화상 회의의 비디오 해상도를 구성할 수 있습니다. 해상도가 높을수록 연결에 더 많은 대역폭이 필요합니다.

HTMLDocuments

이 콘텐츠를 제공하는 HTML 페이지의 이름인 모든 Application에 대해 구성하십시오.

예: 서버가 웹 사이트 www.webrtc.com의 포트 8443에서 실행 중이고 HTMLDocuments.VideoConferencing = /RTCMultiConnection-VideoConferencing.html이면, VideoConferencing에 액세스하는 url은 다음과 같습니다

<https://www.webrtc.com:8443/RTCMultiConnection-VideoConferencing.html>

WebRTC는 보안 연결(HTTPS)을 요구하므로 인증서 사용이 필요합니다. 자세한 내용은 [Server SSL](#)을 읽어 보십시오.

Applications

Name	설명
VideoConferencing	메시 네트워킹 모델을 사용한 다중 사용자(다대다) 비디오 채팅.
ScreenSharing	스타 토폴로지를 사용한 다중 사용자(일대다) 화면 공유.
VideoBroadcasting	스타 토폴로지를 사용한 다중 사용자(일대다) 비디오 브로드캐스팅입니다.

WebPush

RFC 8030
RFC 8291

WebPush 프로토콜은 **RFC 8030**(HTTP Push를 사용한 전달)과 **RFC 8291**(메시지 암호화)에 의해 정의됩니다.

Web Push는 **웹 브라우저에 푸시 알림을 전달하기 위한 표준화된 프로토콜**입니다. 웹사이트가 푸시 메시지를 등록하고 수신할 수 있도록 하는 표준 웹 API인 Push API를 사용합니다. Push API는 사용자가 웹사이트를 적극적으로 탐색하지 않을 때도 웹사이트가 사용자의 브라우저로 푸시 메시지를 보낼 수 있도록 합니다.

Web Push를 사용하려면 웹사이트는 먼저 **사용자의 브라우저로부터 push subscription을 얻어야** 합니다. subscription은 고유한 endpoint URL과 암호화 키로 구성됩니다. endpoint URL은 웹사이트가 사용자의 브라우저에 push 메시지를 보내는 데 사용할 수 있는 URL이며, 암호화 키는 push 메시지를 암호화하고 복호화하는 데 사용됩니다.

웹사이트가 **푸시 구독을 획득**하면, 엔드포인트 URL에 HTTP 요청을 보내 사용자의 브라우저로 **푸시 메시지를 보낼** 수 있습니다. 푸시 메시지는 Web Push Protocol Message라는 특수 형식으로 전송되며, 이는 헤더 집합과 페이로드로 구성됩니다. 헤더에는 암호화 키와 메시지의 TTL(time-to-live) 같은 정보가 포함되고, 페이로드에는 메시지의 실제 내용이 포함됩니다.

사용자의 브라우저가 push 메시지를 수신하면, 먼저 암호화 키를 사용하여 **메시지를 복호화**합니다. 그런 다음 알림 해제 또는 웹사이트 열기와 같이 사용자가 취할 수 있는 추가 작업과 함께 **사용자에게 알림을 표시**합니다.

푸시 메시지의 보안과 개인정보를 보장하기 위해 Web Push는 종단간 암호화를 사용하며 푸시 구독이 안전한 연결(예: HTTPS)을 통해 획득되도록 요구합니다. 또한 이 프로토콜은 푸시 메시지 발신자를 인증하고 남용을 방지하는 메커니즘(예: 웹사이트가 사용자에게 보낼 수 있는 푸시 메시지 수 제한)을 제공합니다.

구성 요소

WebPush를 지원하는 2개의 구성 요소가 있습니다:

- **TsgcWSAPIServer_WebPush**: 서버 측에서 WebPush 프로토콜을 구현하여 사용자에게 권한을 요청하고, 구독을 등록하고, 알림을 보내는 등의 작업을 할 수 있습니다. 이 구성 요소는 이미 알림을 보내기 위한 WebPush 클라이언트를 캡슐화하고 있습니다.
- **TsgcWebPush_Client**: 클라이언트 측에서 WebPush 프로토콜을 구현하여 데스크톱 및 모바일 웹을 통해 사용자에게 알림을 전송할 수 있게 합니다. 이미 키와 엔드포인트가 있고 구독한 클라이언트에 WebPush 메시지만 게시하려는 경우 유용합니다.

TsgcWSAPIServer_WebPush

TsgcWSAPIServer_WebPush 구성 요소입니다.

소개

TsgcWSServer_API_WebPush는 WebPush 구독 처리 기능을 제공하는 구성 요소입니다. WebPush는 브라우저에서 실행되는 웹 애플리케이션에 실시간 알림을 전달하기 위한 프로토콜입니다. 이 구성 요소는 구독을 관리하고 구독된 클라이언트에게 알림을 보내는 데 사용할 수 있습니다. **TsgcWSServer_API_WebPush** 클래스가 제공하는 속성, 이벤트 및 메서드와 함께 사용 방법을 보여주는 코드 예제를 아래에서 확인하십시오.

구성

1. **TsgcWSServer_API_WebPush**를 연결하려면 **Server** 속성을 사용하여 WebSocket 서버에 연결하십시오.
2. **WebPush.VAPID** 속성에서 **공개 키**와 **개인 키**를 구성하십시오. (등록된 사용자는 windows용 VAPID 키를 생성하는 실행 파일을 다운로드할 수 있습니다.)
3. **OpenSSL 3.0.0 버전** 배포가 필요합니다
4. **WebPush.Endpoints** 속성에서 WebPush 구독을 처리할 자체 엔드포인트를 정의할 수 있습니다. 기본적으로 `"/sgcWebPush.html"` 엔드포인트에 접근하면 WebPush 알림을 구독할 수 있는 간단한 웹페이지가 표시됩니다.
5. 서버를 시작하고 테스트하도록 구성된 엔드포인트에 액세스하십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 3개의 속성.
- 메서드 — 3개의 public 메서드입니다.
- 이벤트 — 처리할 수 있는 3개의 이벤트입니다.

TsgcWebPush_Client

*TsgcWebPush_Client*는 구독 세부 정보를 얻으면 알림을 보낼 수 있게 해주는 클래스입니다.

소개

*TsgcWebPush_Client*는 구독 세부 정보를 얻으면 알림을 보낼 수 있게 해주는 클래스입니다.

아래에서 WebPush 구독의 endpoint, public key, authentication secret이 주어졌을 때 WebPush 클라이언트를 사용하여 알림을 보내는 예제를 확인하십시오.

```
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();
    }
}
```

참조

- 속성 — 구성 요소에서 사용 가능한 3개의 속성.
- 메서드 — 1개의 공용 메서드입니다.
- 이벤트 — 처리할 수 있는 0개의 이벤트입니다.

Extensions

WebSocket 프로토콜은 확장되도록 설계되었습니다. WebSocket 클라이언트는 확장을 요청할 수 있고 WebSocket 서버는 클라이언트가 요청한 일부 또는 모든 확장을 수락할 수 있습니다.

지원되는 확장:

1. [Deflate-Frame](#): WebSocket 프레임을 압축합니다.
2. [PerMessage-Deflate](#): WebSocket 메시지를 압축합니다.

Extensions | PerMessage-Deflate

PerMessage는 WebSocket 프로토콜 확장이며, 이 확장이 서버와 클라이언트에 의해 지원되는 경우 양쪽 모두 전송되는 메시지를 압축할 수 있습니다:

- 압축 방법으로 Deflate를 사용합니다.
- 압축은 Application 데이터에만 적용됩니다(제어 프레임 및 헤더은 영향을 받지 않음).
- 서버와 클라이언트는 어떤 메시지를 압축할지 선택할 수 있습니다.

Max Window Bits

이 확장을 사용하면 LZ77 알고리즘에서 사용하는 슬라이딩 윈도우의 서버 및 클라이언트 크기를 사용자 지정할 수 있습니다(8에서 15 사이). 이 값이 클수록 중복을 찾아 제거할 가능성이 높지만 더 많은 메모리와 CPU 사이클을 소비합니다. 15가 기본값입니다.

No Context Take Over

기본적으로 이전 메시지가 압축 및 압축 해제에 사용됩니다. 메시지가 유사하면 압축률이 향상됩니다. 활성화하면 각 메시지가 자체 메시지 데이터만 사용하여 압축됩니다. 기본적으로 비활성화되어 있습니다.

MemLevel

이 값은 서버와 클라이언트 간에 협상되지 않습니다. 1로 설정하면 가장 적은 메모리를 사용하지만 압축 알고리즘을 느리게 하고 압축률을 낮춥니다. 9로 설정하면 가장 많은 메모리를 사용하고 최상의 성능을 제공합니다. 기본적으로 1로 설정됩니다.

** Rad Studio XE2와 함께 제공되는 Indy 버전은 초기화 함수와의 zlib 버전 불일치로 인해 예외를 발생시킵니다. 이를 수정하려면 Indy 버전을 최신 버전으로 업데이트하기만 하면 됩니다.*

Extensions | Deflate-Frame

이것은 Chrome이나 Safari와 같은 WebKit 브라우저에서 지원되는, WebSocket 프로토콜로 전송되는 프레임의 압축을 허용하는 WebSocket 프로토콜 확장입니다. 이 확장은 Server 및 Client 구성 요소에서 지원됩니다.

이 확장은 더 이상 사용되지 않습니다.

** Rad Studio XE2와 함께 제공되는 Indy 버전은 초기화 함수와의 zlib 버전 불일치로 인해 예외를 발생시킵니다. 이를 수정하려면 Indy 버전을 최신 버전으로 업데이트하기만 하면 됩니다.*

MCP

MCP (Model Context Protocol)는 언어 모델과 외부 시스템 간의 **정보의 컨텍스트 교환**을 정의하고 관리하도록 설계된 표준화된 프로토콜입니다.

이는 AI 모델이 네이티브 환경을 넘어 외부 도구나 데이터 소스를 **이해**하고, **확장**하고, **상호 작용**할 수 있게 하는 구조화되고 안전하며 동적인 통신을 가능하게 합니다.

MCP는 컨텍스트 메타데이터를 전달하기 위한 **통일된 인터페이스**를 제공하여, *WebAuthn*이 인증 흐름을 표준화하는 것과 정신적으로 유사하게, 모델이 명령을 해석하고, 데이터를 검색하고, 작업을 예측 가능하고 상호 운용 가능한 방식으로 실행할 수 있도록 합니다.

목적

MCP는 모델, 클라이언트 및 도구 간의 **컨텍스트 분열**과 **통합 불일치** 문제를 해결하기 위해 개발되었습니다.

MCP를 사용하면, 호환되는 모든 클라이언트가 공유되고 잘 정의된 구조를 통해 모든 MCP 호환 모델 또는 서비스와 통신할 수 있습니다.

일반적인 사용 사례에는 다음이 포함됩니다:

- 모델 쿼리에 **컨텍스트 메타데이터**(사용자 세션, 앱 상태, 권한 등)를 연결합니다.
- **function** 또는 **도구 정의**를 동적으로 교환합니다.
- 모델 세션 전반에 걸쳐 **상태 저장 상호 작용**을 관리합니다.
- LLM, SDK 및 백엔드 서비스 간의 **안전한 상호운용성** 확립.

구성 요소

- **TsgcWSAPIServer_MCP**: 이 구성 요소는 Tools, Prompts 및 Resources 요청, 인증, 이벤트를 사용한 플로 제어 등의 기능을 갖춘 MCP 서버를 구현하기 위한 간단하지만 강력한 솔루션을 제공합니다.
- **TsgcWSAPIClient_MCP**: 이 구성 요소는 Tools, Prompts 및 Resources 요청 등의 기능으로 MCP 클라이언트를 구현하기 위한 간단하지만 강력한 솔루션을 제공합니다.

TsgcWSAPIServer_MCP

TsgcWSAPIServer_MCP 구성 요소.

소개

TsgcWSAPIServer_MCP는 sgcWebSockets HTTP 서버 엔드포인트를 통해 Model Context Protocol(MCP)을 노출합니다. 구성 요소는 들어오는 HTTP 요청을 TsgcAI_MCP_Server 엔진과 연결하여 MCP 호환 클라이언트가 세션을 협상하고, prompt/resource/tool을 열거하고, JSON-RPC 스타일 호출을 통해 tool을 호출할 수 있도록 합니다.

구성

- **구성 요소 배치:** TsgcWSAPIServer_MCP 구성 요소는 Server 속성을 사용하여 HTTP 서버, [TsgcWebSocketHTTP_Server](#) 또는 [TsgcWebSocketServer_HTTPAPI](#)에 첨부되어야 합니다.
- **엔드포인트 선택:** 기본 /mcp 경로를 변경해야 하는 경우 EndpointOptions.Endpoint를 조정하십시오. 문서가 이 경로와 동일한 요청은 MCP 호출로 처리됩니다. 다른 요청은 상위 API의 리소스 처리로 폴백합니다.
- **MCP 설정 구성:** **MCPOptions.ServerInfo**를 사용하여 서버의 이름/버전을 지정하고, **MCPOptions.SessionTimeout**을 사용하여 세션 수명을 제어하고, **MCPOptions.AuthenticationOptions**를 사용하여 JSON-RPC 호출이 처리되기 전에 API 키 또는 사용자 지정 HTTP 헤더를 요구하십시오.
- **프롬프트, 도구, 리소스 및 리소스 템플릿 등록:** 구성 요소가 노출하는 읽기 전용 **Tools, Prompts, Resources** 및 **ResourceTemplates** 목록을 채웁니다.
- **이벤트 처리:** 아래에 설명된 게시된 이벤트를 연결하여 세션 수명 주기, 프롬프트/리소스 해결 또는 도구 실행을 위한 비즈니스 로직을 연결하십시오. 예외를 가로채어 HTTP 응답 코드를 사용자 정의할 수 있습니다.

내장 HTTP 엔드포인트(구성 불필요)

구성 요소가 HTTP 서버에 연결되면 다음 엔드포인트와 동작이 /mcp와 함께 자동으로 제공되며, 이를 활성화하기 위한 새 속성이 필요하지 않습니다. 이를 통해 CORS를 통한 Dynamic Client Registration이 있는 OAuth 2.1을 요구하는 **claude.ai**와 같은 웹 기반 MCP 커넥터와 즉시 호환됩니다.

- **origin 반영이 있는 CORS 지원.** 처리된 경로에 대한 OPTIONS 사전 비행은 다음 헤더와 함께 204 No Content를 반환합니다:
 - Access-Control-Allow-Origin: 요청 Origin 헤더의 에코, 없으면 *.
 - 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
 OPTIONS가 아닌 모든 응답에도 Access-Control-Allow-Origin, Access-Control-Expose-Headers, Strict-Transport-Security가 포함됩니다.
- **www-Authenticate가 있는 401 Unauthorized.** MCPOptions.AuthenticationOptions.ApiKey(또는 CustomHeader)가 활성화되어 있고 요청에 자격 증명이 없거나 잘못된 경우, 서버는 401과 다음 형식의 헤더로 응답합니다:


```
Bearer realm="MCP", resource_metadata="https://<host>/well-known/oauth-protected-resource"
```
- **OAuth 2.1 discovery 및 flow 엔드포인트.** 다음 엔드포인트는 /mcp와 동일한 HTTP 서버에서 제공되며, PKCE(S256), authorization-code 및 refresh-token grant, 그리고 Dynamic Client Registration을 지원합니다:

Path	Spec	동작
GET /.well-known/oauth-	RFC 8414	JSON 권한 부여 서버 메타데이터입니다. PKCE(S256), authorization_code + refresh_token grant, public-client +

Path	Spec	동작
<code>authorization-server</code>		<code>client_secret_basic</code> 인증 방법, <code>/oauth/register</code> 에서의 동적 등록 및 <code>mcp</code> scope를 광고합니다.
<code>GET /.well-known/oauth-protected-resource</code>	RFC 9728	MCP 엔드포인트를 보호된 리소스로 설명하고 이 서버를 권한 부여 서버로 가리키는 JSON입니다.
<code>POST /oauth/register</code>	RFC 7591 (DCR)	JSON 클라이언트 등록 본문(<code>client_name</code> , <code>redirect_uris</code>)을 수락합니다. 생성된 <code>client_id</code> 를 발급합니다(<code>client_secret</code> 없음 — public client). 등록된 클라이언트 정보 응답과 함께 <code>201</code> 을 반환합니다.
<code>GET /oauth/authorize</code>	RFC 6749 + RFC 7636	HTML 동의 페이지를 렌더링합니다. 필수 쿼리 매개변수: <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	동의 폼을 처리합니다. 사용자는 승인하려면 <code>MCPOptions.AuthenticationOptions.ApiKey.Value</code> 와 동일한 값을 입력해야 합니다. 승인 시 <code>?code=...&state=...</code> 와 함께 <code>redirect_uri</code> 로 리디렉션합니다. 거부하거나 키가 잘못된 경우 <code>?error=access_denied&state=...</code> 로 리디렉션하거나 폼을 다시 렌더링합니다.
<code>POST /oauth/token</code>	RFC 6749 + RFC 7636	Form 인코딩된 본문입니다. <code>grant_type=authorization_code</code> (PKCE 검증: <code>BASE64URL(SHA256(code_verifier)) == code_challenge</code>) 및 <code>grant_type=refresh_token</code> 을 지원합니다. 1시간 bearer 액세스 토큰과 갱신 토큰을 발급합니다. 인증 코드는 60초 후에 만료되며 일회용입니다.

- **Bearer token 검증 확장.** `MCPOptions.AuthenticationOptions.ApiKey.Enabled`가 `True`일 때, `/mcp` 요청의 `Authorization: Bearer <x>` 헤더는 `<x>`가 구성된 정적 `ApiKey.Value` 또는 현재 유효한 OAuth 발급 access token(위 흐름을 통해 발급됨) 중 하나와 일치하면 수락됩니다.
- **메모리 내 저장소.** OAuth 클라이언트, 코드 및 토큰은 메모리에만 보관됩니다. 서버를 다시 시작하면 손실됩니다.
- **OnMCPHttpRequest를 통한 재정의.** `OnMCPHttpRequest`에 연결된 핸들러는 `Handled := True`를 설정하여 위의 모든 동작을 단축할 수 있습니다.

참조

- 속성 — 구성 요소에서 사용할 수 있는 5개의 속성.
- 메서드 — 9개의 public 메서드.
- 이벤트 — 처리할 수 있는 12개의 이벤트.

MCP Server | Sessions

서버와 클라이언트 간의 성공적인 초기화 후 새 세션이 생성됩니다. 세션은 항상 클라이언트와 서버 간의 요청 및 응답에 전송되어 요청/응답 흐름에서 어떤 세션이 사용되는지 식별합니다. HTTP 전송을 사용할 때 세션의 수명은 단일 HTTP 요청 및 응답이며, 이는 응답이 전송되면 세션이 닫힌다는 것을 의미합니다.

세션 객체는 모든 이벤트에 전달되며, 세션이 생성되거나 삭제되는 시점을 알기 위해 다음 이벤트를 사용하십시오:

- **OnMCPSessionNew**: 새 세션 id가 생성될 때마다 이 이벤트가 호출됩니다.
- **OnMCPSessionEnd**: 세션이 종료되면 이 이벤트가 호출됩니다.

OnMCPSessionNew

새 세션이 생성될 때 이벤트가 호출됩니다. Session 매개변수에 액세스하여 세션 ID, 세션 만료 시기 등을 얻을 수 있습니다.

OnMCPSessionEnd

이 이벤트는 세션이 완료되었을 때 호출됩니다.

```
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

TsgcWSAPIServer_MCP는 sgcWebSockets를 통해 Model Context Protocol(MCP) 서버 기능을 노출하는 고급 구성 요소입니다. **MCP 클라이언트가 MCP Server Tools 사양에 따라 도구를 요청할 때**, 이 구성 요소는 HTTP 게이트웨이, 세션 수명 주기 및 JSON-RPC 직렬화를 조율하여 Delphi 코드가 비즈니스 로직 구현에 집중할 수 있게 합니다.

EndpointOptions.Endpoint에 정의된 MCP 엔드포인트에 도달하는 들어오는 HTTP 요청은 **TsgcWSServer_API_MCP**에 의해 가로채집니다. 구성 요소는 MCP 세션을 생성(또는 재개)하고, 요청을 기본 TsgcAI_MCP_Server 엔진으로 전달한 다음, 사양에 필요한 mcp-session-id 헤더를 유지하면서 JSON-RPC 응답을 HTTP 클라이언트로 다시 씁니다.

도구 목록

서버는 **TsgcAI_MCP_ToolsList**에 도구의 인메모리 카탈로그를 유지합니다. 도구는 이름으로 고유함이 보장되며, 설명을 노출하고, 사양의 tools.list 메서드에 대한 응답으로 방출되는 JSON-Schema 형태의 입력 설명을 제공합니다. 클라이언트가 tools.list를 호출하면 TsgcWSAPIServer_MCP는 요청을 로드하고, 현재 카탈로그를 직렬화하여, 200 HTTP 상태 코드와 함께 다시 보냅니다.

간단한 계산기 도구를 게시하는 예제 코드:

```
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");
}
```

클라이언트에 전송되는 규격 준수 tools.list 응답은 MCP JSON 스키마 페이로드를 미러링합니다. 예를 들어:

```
{
  "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" }
          }
        }
      }
    ]
  }
}
```

도구 요청

클라이언트가 tools.call JSON-RPC 요청을 발행하면, **TsgcWSAPIServer_MCP**는 **OnMCPRequestTool** 이벤트를 발생시키기 전에 강력하게 타입이 지정된 요청 객체(도구 이름과 제공된 인수 포함)를 hydrate합니다. 핸들러는 응답 페이로드를 채우며, 이는 성공 HTTP 상태 코드와 함께 클라이언트로 다시 직렬화됩니다.

일반적인 핸들러는 다음과 같습니다:

```
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;
    }
}
```

생성된 JSON-RPC 응답은 사양의 tools.call 스키마를 따르며, 하나 이상의 content 부분과 선택적으로 구조화된 페이로드를 감쌉니다. 구성 요소가 응답을 지우고 HTTP 코드를 설정해 주므로, 추가 plumbing이 필요하지 않습니다.

Public API surface

여러 헬퍼 클래스가 도구 카탈로그와 이벤트 응답 헬퍼를 노출합니다. 다음 목록은 일반적으로 프로덕션 코드에서 호출되는 메서드를 요약합니다.

TsgcAI_MCP_ToolsList

- **AddTool(const Name, Description):** 도구 항목을 생성하거나 교체하고 기본 **TsgcAI_MCP_Tool**을 반환하므로 스키마와 기본값을 구성할 수 있습니다.
- **Clear:** 등록된 모든 도구를 제거합니다. 구성을 디스크에서 로드할 때 카탈로그를 다시 빌드하기 전에 이를 호출하십시오.
- **Count / Items[Index]:** 열거, 진단 또는 사용자 정의 직렬화를 위해 목록 내용을 노출합니다.

TsgcAI_MCP_Tool

- **InputSchema.Properties.AddProperty(Name, Required, JsonType, Description):** tools.list를 통해 광고될 JSON 입력 페이로드를 문서화합니다.
- **InputSchema.Properties.Clear:** 애플리케이션이 런타임에 도구 기능을 새로 고칠 때 스키마를 즉석에서 다시 빌드합니다.
- **Description:** 클라이언트와 공유되는 사람이 읽을 수 있는 요약을 저장하는 쓰기 가능한 속성입니다.

TsgcAI_MCP_Response_ToolsCall

- **Result.Content.AddText(Value):** 도구 응답에 텍스트 블록을 추가합니다. 여러 호출은 다중 부분 응답을 생성합니다.
- **Result.Content.AddImage(Data, MimeType):** tool 응답에 이미지 블록을 추가합니다. 여러 호출은 multi-part 응답을 생성합니다.
- **Result.Content.AddAudio(Data, MimeType):** 도구 응답에 오디오 블록을 추가합니다. 여러 번 호출하면 다중 부분 응답이 생성됩니다.
- **Result.Content.AddEmbeddedResource(Uri, Title, Text, MimeType):** 도구 응답에 임베디드 리소스 블록을 추가합니다. 여러 번 호출하면 다중 파트 응답이 생성됩니다.
- **Result.Content.AddResourceLink(Uri, Name, MimeType):** 도구 응답에 리소스 링크 블록을 추가합니다. 여러 번 호출하면 다중 파트 응답이 생성됩니다.
- **Result.Content.Clear:** 새 콘텐츠를 추가하기 전에 이전에 생성된 출력을 재설정합니다.
- **Result.IsError:** 도구 호출을 수행할 수 없을 때 이 플래그를 **True**로 설정하면 프레임워크가 MCP 오류 객체를 직렬화합니다.

고급 예제

다음 스니펫은 입력을 검증하고, 여러 텍스트 블록을 반환하고, 매개변수가 유효하지 않을 때 **IsError**를 전환하는 geo lookup 도구를 빌드합니다.

```
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

MCP에서 *prompts*는 서버가 노출하는 재사용 가능한 템플릿 또는 워크플로입니다. 이들은 사용자 제어입니다(클라이언트가 호출할 프롬프트를 선택함). 두 가지 주요 메서드는 다음과 같습니다:

- **prompts/list**: 서버가 노출하는 사용 가능한 모든 prompt 정의 목록을 요청합니다. TsgcWSAPIServer_MCP 구성 요소에서 내부적으로 처리됩니다.
- **prompts/get**: 특정 프롬프트의 콘텐츠와 선택적 메타데이터(일반적으로 클라이언트가 상호 작용하는 방법을 설명하는 메시지 시퀀스)를 검색합니다. 서버가 새 요청을 받으면 **OnMCPRequestPrompt** 이벤트가 호출됩니다.

또한 서버가 선언된 capabilities에서 prompts.listChanged = true를 설정하면, prompt 카탈로그가 업데이트될 때 notifications/prompts/list_changed와 같은 알림을 보낼 수 있습니다.

클라이언트가 prompts/get를 요청하면 인수(있는 경우)를 보내고 콘텐츠 블록(보통 텍스트이지만 리소스나 이미지를 포함할 수도 있음)이 있는 일련의 메시지(user 또는 assistant 역할)를 기대합니다.

문제가 있으면(누락된 프롬프트, 유효하지 않은 인수), 서버는 -32602(Invalid params) 또는 -32603(Internal error) 같은 JSON-RPC 오류 코드를 반환합니다.

Prompts List

서버는 **TsgcAI_MCP_PromptsList**에 프롬프트의 인메모리 카탈로그를 유지합니다. 프롬프트는 이름으로 고유함이 보장되며, 설명을 노출하고, 사양의 prompts.list 메서드에 대한 응답으로 방출되는 JSON-Schema 형태의 입력 설명을 제공합니다. 클라이언트가 prompts.list를 호출하면 TsgcWSAPIServer_MCP는 요청을 로드하고, 현재 카탈로그를 직렬화하여, 200 HTTP 상태 코드와 함께 다시 보냅니다.

코드 리뷰 프롬프트를 게시하는 예제 코드:

```
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

클라이언트가 prompts.get JSON-RPC 요청을 발행하면 **TsgcWSAPIServer_MCP**는 **OnMCPRequestPrompt** 이벤트를 발생시키기 전에 강력하게 형식화된 요청 객체(prompt 이름 및 제공된 인수 포함)를 하이드레이션합니다. 핸들러는 응답 페이로드를 채우고, 이는 성공 HTTP 상태 코드와 함께 클라이언트로 다시 직렬화됩니다.

일반적인 핸들러는 다음과 같습니다:

```
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

Prompts는 몇 가지 헬퍼 클래스를 통해 모델링됩니다. 다음 메서드는 prompt 템플릿을 큐레이션하고 **prompts/get** 응답을 구성할 때 가장 자주 사용됩니다.

TsgcAI_MCP_PromptsList

- **AddPrompt(const Name, Description):** 새 프롬프트 항목을 등록하고 인수와 메시지를 구성할 수 있도록 **TsgcAI_MCP_Prompt** 객체를 반환합니다.
- **Clear:** 카탈로그를 지웁니다. 구성 파일에서 프롬프트를 로드하는 시작 시 편리합니다.
- **Count / Items[Index]:** 진단, UI 시각화 또는 수동 직렬화를 위해 목록 내용에 대한 액세스를 제공합니다.

TsgcAI_MCP_Prompt

- **Arguments.AddArgument(Name, Description, Required):** 프롬프트가 수락하는 구조화된 입력을 선언합니다.
- **Arguments.Clear:** 런타임에 프롬프트가 변경될 때 인수 목록을 다시 작성합니다.
- **Messages.AddText(Role, Text):** 프롬프트 정의의 일부로 클라이언트에 다시 전송되는 대화 턴을 미리 시드합니다.

TsgcAI_MCP_Response_PromptsGet

- **Result.Description:** 프롬프트 페이로드에 동반되는 짧은 사람이 읽을 수 있는 요약입니다.
- **Result.Messages.AddText(Role, Text):** 프롬프트를 가져올 때 추가 메시지를 동적으로 주입합니다.
- **Result.Messages.AddImage(Data, MimeType):** prompt 응답에 이미지 블록을 추가합니다. 여러 번 호출하면 다중 부분 응답이 생성됩니다.
- **Result.Messages.AddAudio(Data, MimeType):** 프롬프트 응답에 오디오 블록을 추가합니다. 여러 번 호출하면 다중 부분 응답이 생성됩니다.
- **Result.Messages.AddEmbeddedResource(Uri, Title, Text, MimeType):** 프롬프트 응답에 임베디드 리소스 블록을 추가합니다. 여러 번 호출하면 다중 부분 응답이 생성됩니다.
- **Result.Messages.Clear:** 객체를 재사용하기 전에 응답을 재설정합니다.

고급 예제

다음 예제는 LLM을 분류 워크플로우로 안내하는 프롬프트를 게시합니다. 선택적 인수와 핸들러에서 다단계 지침을 반환하는 방법을 강조합니다.

```
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

MCP에서 **resources**는 서버가 노출하는 주소 지정 가능한 데이터 객체(예: 파일, 데이터베이스 레코드, 생성된 문서 또는 동적 API 출력)를 나타냅니다.

이들은 *클라이언트 제어*이며, 이는 클라이언트가 요청할 리소스와 콘텐츠를 해석하는 방법을 결정할 수 있음을 의미합니다.

관련된 두 가지 주요 메서드는 다음과 같습니다:

- **resources/list**: 서버가 노출하는 모든 사용 가능한 리소스 정의 목록을 요청합니다. TsgcWSAPIServer_MCP 구성 요소에 의해 내부적으로 처리됩니다.
- **resources/read**: 특정 리소스의 내용 및 선택적 메타데이터를 검색합니다. 서버가 새 요청을 수신하면 **OnMCPRequestResource** 이벤트가 호출됩니다. 리소스는 일반적으로 다음을 포함합니다:
 - **uri**: 리소스 식별자.
 - **mimeType**: 콘텐츠 유형(text/plain, application/json, image/png 등).
 - **data** 또는 **content**: 실제 페이로드(텍스트 또는 바이너리).

또한, 서버가 선언된 capabilities에서 resources.listChanged = true를 설정하면, 리소스 카탈로그가 업데이트될 때 notifications/resources/list_changed와 같은 알림을 보낼 수 있습니다.

리소스 목록

서버는 **TsgcAI_MCP_ResourcesList**에 리소스의 인메모리 카탈로그를 유지합니다. 리소스는 uri로 고유성이 보장되며, 설명을 노출하고, 사양의 resources.list 메서드에 대한 응답으로 방출되는 JSON-Schema와 유사한 입력 URI를 제공합니다. 클라이언트가 resources.list를 호출하면 **TsgcWSAPIServer_MCP**는 요청을 로드하고, 현재 카탈로그를 직렬화하여 200 HTTP 상태 코드와 함께 다시 전송합니다.

파일 리소스를 게시하는 예제 코드:

```
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
    );
}
```

Resources Request

클라이언트가 resource.read JSON-RPC 요청을 발행하면, **TsgcWSAPIServer_MCP**는 **OnMCPRequestResource** 이벤트를 발생시키기 전에 강력하게 형식화된 request 객체(uri resource, name 및 제공된 arguments 포함)를 하이드레이트합니다. 핸들러가 response 페이로드를 채우면, 이것이 성공 HTTP 상태 코드와 함께 클라이언트로 다시 직렬화됩니다.

일반적인 핸들러는 다음과 같습니다:

```
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

리소스는 카탈로그 항목(**TsgcAI_MCP_Resource**)과 **OnMCPRequestResource** 내에서 사용되는 스트리밍 도우미를 결합합니다. 다음은 가장 자주 사용하게 될 메서드입니다.

TsgcAI_MCP_ResourcesList

- **AddResource(Uri, Name, Title, Description, MimeType):** 정적 또는 동적으로 생성된 리소스를 등록합니다.
- **Clear:** 모든 리소스 정의를 제거합니다. 구성 파일에서 목록을 다시 빌드하기 전에 유용합니다.
- **Count / Items[Index]:** 열거 또는 진단 보기를 위해 카탈로그를 노출합니다.

TsgcAI_MCP_Resource

- **MimeType / Title / Description:** 클라이언트에 반환되는 리소스 페이로드를 설명하는 쓰기 가능한 속성입니다.
- **Name / Uri:** 리소스 항목을 다시 등록하지 않고 식별자를 업데이트하려면 이러한 속성을 사용하십시오.

TsgcAI_MCP_Response_ResourcesRead

- **Result.Contents.AddContentText(Uri, Name, Title, MimeType, Text):** 호출자에게 텍스트 페이로드를 다시 전송합니다.
- **Result.Contents.AddContentBlob(Uri, Name, Title, MimeType, Text):** 호출자에게 텍스트 페이로드를 다시 보냅니다.
- **Result.Contents.Clear:** 응답 객체를 재사용하기 전에 나가는 페이로드를 재설정합니다.
- **Result.IsError:** 리소스를 생성할 수 없을 때 응답을 오류로 표시합니다.

고급 예제

아래 예제는 집계된 메트릭을 반환하는 JSON 리소스를 게시합니다. 들어오는 URI를 검증하고 요청에 따라 다른 응답을 만드는 방법을 보여줍니다.

```

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

Roots는 클라이언트가 서버에게 탐색을 허용하는 파일 시스템 위치를 표시합니다. 각 항목은 선택적 표시 이름이 있는 file:// URI로, MCP 자동화가 프로젝트 경계를 존중하면서 사용자가 공유되는 내용을 제어할 수 있게 합니다. 서버는 roots/list를 발행하여 카탈로그를 쿼리하고, 클라이언트가 listChanged를 알리면 notifications/roots/list_changed를 받아 캐시된 뷰를 새로 고칠 수 있습니다.

root는 클라이언트 소유이므로, 서버는 항상 요청자 역할을 합니다. 검색 호출을 시도하기 전에 클라이언트의 initialise 페이로드에 roots 기능이 있어야 합니다. 그렇지 않으면 RequestRootsList가 CapabilityNotSupported를 발생시킵니다.

루트 카탈로그 요청

TsgcWSAPIServer_MCP는 roots/list를 직렬화하고 회신을 코드로 다시 라우팅하는 헬퍼를 노출합니다:

- **Request 헬퍼:** MCP 세션이 roots 기능을 광고하면 RequestRootsList(Session)을 호출하십시오. 메서드는 식별자를 자동으로 할당하고(제공하지 않는 한) 응답이 상관될 수 있도록 보류 중인 요청을 영속화합니다.
- **응답 이벤트:** OnMCPResponseRootsList(Sender, Session, Request, Response)를 처리하십시오. 이 이벤트는 클라이언트가 사용 가능한 디렉터리로 응답할 때 발생합니다. JSON 페이로드를 검사하려면 Response.Roots를 검사하거나 Response.Write를 기록하십시오.
- **변경 감지:** 클라이언트가 나중에 notifications/roots/list_changed를 보내면, 서버 측 캐시가 동기화 상태를 유지하도록 또 다른 RequestRootsList를 제출하십시오.

샘플 코드

아래 코드 조각은 세션이 설정되자마자 roots를 요청하고 반환된 항목을 기록합니다.

```
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

Sampling을 사용하면 서버가 클라이언트에 LLM 생성을 대신 실행하도록 요청할 수 있습니다. 요청은 대화 기록, 선택적 system prompt, 모델 기본 설정 및 token 예산을 설명합니다. 클라이언트는 어떤 공급자가 사용되는지 제어하고, human-in-the-loop 검토를 수행하며, 최종적으로 승인된 completion을 반환합니다.

클라이언트는 initialize 중에 sampling 기능을 선언해야 합니다. 그렇지 않으면 sampling 시도가 기능 오류로 거부됩니다. 사양은 사용자 승인을 권장하고 대화 기록 내에 텍스트, 이미지 또는 오디오 콘텐츠 블록을 허용합니다.

샘플링 요청 발행

TsgcWSAPIServer_MCP는 헬퍼와 응답 이벤트로 sampling/createMessage 흐름을 간소화합니다:

- **Request 헬퍼:** TsgcAI_MCP_Request_SamplingCreateMessage를 구축하고, Params.Messages, Params.ModelPreferences, Params.SystemPrompt 및 선택적으로 Params.MaxTokens를 채워십시오. 이를 RequestSamplingCreateMessage(Session, Request)에 전달하십시오. 메서드는 id를 스탬프하고(필요한 경우) 대기 중인 호출을 기록합니다.
- **응답 이벤트:** OnMCPResponseSamplingCreateMessage(Sender, Session, Request, Response)를 처리하십시오. 워크플로를 계속하려면 Response.Role, Response.Content, Response.Model 및 Response.StopReason을 검사하십시오.
- **오류 처리:** 거부되거나 취소된 요청은 JSON-RPC 오류로 도착하므로 헬퍼를 try/except로 감싸고 로깅을 위해 OnMCPException을 모니터링하십시오.

샘플 코드

아래 스니펫은 클라이언트에서 짧은 요약물을 요청하고 어시스턴트 응답을 출력합니다.

```
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

Elicitation을 사용하면 서버가 워크플로 중간에 클라이언트에게 사용자로부터 구조화된 입력을 수집하도록 요청할 수 있습니다. 요청에는 사람이 읽기 쉬운 메시지와 캡처해야 하는 필드를 설명하는 제약된 JSON 스키마가 포함됩니다. 클라이언트는 적절한 폼을 렌더링하고, 답변을 로컬에서 검증하고, 사용자가 프롬프트를 수락, 거부 또는 취소했는지 반환합니다. 클라이언트는 initialize 중에 elicitation 기능을 알려야 합니다. 스키마는 렌더링을 단순하게 유지하고 민감한 데이터를 범위에 서 제외하기 위해 의도적으로 기본 속성(string, number, boolean 또는 enum)을 가진 플랫폼 객체로 제한됩니다.

Requesting user input

TsgcWSAPIServer_MCP는 elicitation/create에 대한 헬퍼를 제공하고 사용자가 상호 작용을 마치면 response 이벤트를 발생시킵니다:

- **Request helper:** TsgcAI_MCP_Request_ElicitationCreate를 생성하고, Params._Message를 설정하고, Params.RequestedSchema를 통해 예상 페이로드를 설명하십시오. 클라이언트에 보내려면 RequestElicitationCreate(Session, Request)를 호출하십시오.
- **응답 이벤트:** OnMCPResponseElicitationCreate(Sender, Session, Request, Response)를 처리합니다. Response.Action(accept, decline 또는 cancel)을 검사하고 사용자가 수락한 경우 Response.Content를 구문 분석하십시오.
- **Validation:** 거부된 요청은 JSON-RPC 오류로 도착합니다. 콘텐츠를 소비하기 전에 항상 Action을 확인하고 사용자가 취소한 경우 다시 요청할 준비를 하십시오.

샘플 코드

아래 스니펫은 연락처 정보를 요청하고, 응답을 기다린 다음, 결과 동작과 페이로드를 기록합니다.

```
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

TsgcWSAPIClient_MCP 구성 요소입니다.

소개

TsgcWSAPIClient_MCP는 sgcWebSockets 네트워킹 구성 요소 위에 Model Context Protocol(MCP) 소비자를 구현합니다. 세션 협상, JSON-RPC 요청/응답 마샬링 및 편의 이벤트를 처리하므로 Delphi & Cbuilder 애플리케이션이 MCP 호환 서버가 노출하는 prompts, resources 및 tools를 검색할 수 있습니다.

구성

- **구성 요소 놓기:** **TsgcWSAPIClient_MCP**를 폼/데이터 모듈에 놓습니다.
- **MCP 엔드포인트 가리키기:** **MCPOptions.HttpOptions.URL**을 프로토콜을 노출하는 서버 HTTP URL로 설정하십시오(예: `https://localhost:5001/mcp`). 필요한 경우 중첩된 **TLSoptions** 및 **LogOptions** 객체를 통해 TLS 및 로깅을 구성하십시오.
- **클라이언트 설명:** **MCPOptions.ClientInfo**에 제품 *Name*, 사람이 읽을 수 있는 *Title* 및 의미론적 *Version*을 채우십시오. 이러한 값은 **initialize** 핸드셰이크 중에 전달됩니다.
- **세션 보안:** 서버가 API 키 또는 사용자 정의 HTTP 헤더를 기대할 때 **MCPOptions.AuthenticationOptions**를 구성하십시오. 자격 증명은 스트리밍 업그레이드를 포함한 모든 HTTP 요청에 첨부됩니다.
- **keep-alive 조정:** 자동 ping 주기를 활성화하고 원하는 간격(초)을 정의하려면 **MCPOptions.HeartBeat**를 사용하십시오. 내부 keep-alive 타이머가 작동하면 구성 요소가 내부적으로 **Ping** 을 호출합니다.
- **이벤트 처리:** 응답을 로깅하고, 세션을 수락/거부하거나, 서버가 반환한 페이로드를 후처리하기 위해 나중에 설명하는 이벤트를 구독하십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 2개의 속성.
- 메서드 — 12개의 public 메서드.
- 이벤트 — 처리할 수 있는 12개의 이벤트.

MCP Client | Tools

TsgcWSAPIClient MCP 구성 요소는 서버가 제공하는 도구를 열거하고 도구를 호출하기 위해 Model Context Protocol(MCP)이 정의한 JSON-RPC 호출을 노출합니다. 모든 메서드는 세션이 이미 *initialized* 상태인지 검증하므로, 각 요청은 Initialize가 성공한 후에 전송되어야 합니다.

ListTools request

MCP 서버가 현재 노출하는 도구 레지스트리를 검색하려면 **ListTools** 를 호출하십시오. 구성 요소는 자동으로 tools/list JSON-RPC 요청을 빌드하고 파싱된 응답과 함께 **OnMCPListTools** 이벤트를 발생시킵니다.

- **Request:** ListTools (매개변수 없음).
- **이벤트:** OnMCPListTools(Sender, Request, Response).
- **Response:** Tools 쿼리가 있는 TsgcAI_MCP_Response_ToolsList입니다. 각 항목은 도구 Name, Title, Description 및 그 InputSchema(JSON 스키마 유형과 명명된 속성)를 노출하는 TsgcAI_MCP_Tool입니다.

카탈로그를 검사하려면 쿼리의 Count와 Item[Index] 도우미 또는 GetTool을 사용하십시오.

ToolsCall request

게시된 도구 중 하나를 호출하려면 **RequestTool**(Name, Arguments)를 호출하십시오. 구성 요소는 JSON 인수를 요청 페이지 로드에서 복사하고 응답이 파싱된 후 **OnMCPResponseTool** 를 발생시킵니다.

- **Request:** 도구 Name과 선택적 JSON Arguments가 있는 RequestTool입니다.
- **이벤트:** OnMCPResponseTool(Sender, Request, Response).
- **응답:** TsgcAI_MCP_Response_ToolsCall. Result.IsError, Result.StructuredContent를 검사하고 구조화된 출력 조각을 위해 Result.Content를 반복하십시오. 도구 응답에는 텍스트, 이미지, 오디오 blob, 임베디드 리소스 또는 리소스 링크가 포함될 수 있습니다.

샘플 코드

다음 코드 조각은 도구 목록을 요청하고, 도구를 실행하고, 결과를 읽는 방법을 보여 줍니다.

```
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

MCP Client 구성 요소는 서버가 제공하는 프롬프트를 중심으로 하는 두 가지 JSON-RPC 메서드를 제공합니다. 사용 가능한 프롬프트 템플릿을 나열하는 것과 프롬프트 실행을 요청하는 것입니다. 이 호출을 실행하기 전에 클라이언트가 초기화되었는지 확인하십시오.

PromptsList request

ListPrompts 는 현재 프롬프트 카탈로그를 요청합니다. 응답이 수신되면 구성 요소가 **OnMCPListPrompts**를 발생시킵니다.

- **요청:** ListPrompts (매개변수 없음).
- **이벤트:** OnMCPListPrompts(Sender, Request, Response).
- **Response:** TsgcAI_MCP_Response_PromptsList가 Prompts 큐를 노출합니다. 각 TsgcAI_MCP_Prompt에는 Name, Title, Description 및 Arguments 컬렉션(인수 이름, 설명, required 플래그)이 포함됩니다.

PromptsGet request

RequestPrompt(Name, Arguments)는 제공된 JSON 인수로 프롬프트 템플릿을 실행합니다. 서버 응답이 준비되면 구성 요소는 **OnMCPResponsePrompt** 를 발생시킵니다.

- **Request:** prompt **Name** 및 선택적 JSON **Arguments**가 있는 RequestPrompt.
- **이벤트:** OnMCPResponsePrompt(Sender, Request, Response).
- **응답:** TsgcAI_MCP_Response_PromptsGet. 요약은 Result.Description을 검사하고 각 TsgcAI_MCP_Response_PromptsGet_Result_Message를 읽으려면 Result.Messages를 반복하십시오. 메시지는 다형적입니다 — 기본 내용과 주석에 액세스하려면 특정 자손(..._Text, ..._Image, ..._Audio, ..._EmbeddedResource)를 사용하십시오.

샘플 코드

아래 예제는 프롬프트 목록을 로드하고, 프롬프트를 호출하며, 반환된 메시지를 출력합니다.

```
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

MCP 리소스 API를 사용하면 세션이 초기화된 후 **TsgcWSAPIClient_MCP** 구성 요소를 통해 클라이언트가 서버 측 아티팩트를 발견하고 그 콘텐츠를 검색할 수 있습니다.

ResourcesList 요청

리소스 카탈로그를 얻으려면 **ListResources** 를 사용하십시오. 구성 요소는 JSON-RPC 응답을 파싱한 후 **OnMCPListResources** 를 발생시킵니다.

- **요청:** ListResources (매개변수 없음).
- **Event:** OnMCPListResources(Sender, Request, Response).
- **Response:** Resources 큐를 노출하는 TsgcAI_MCP_Response_ResourcesList입니다. 모든 TsgcAI_MCP_Resource 항목에는 리소스 Uri, Name, Title, Description 및 MimeType가 포함됩니다.

ResourcesRead 요청

특정 리소스를 다운로드하려면 **RequestResource(Uri)**를 호출하십시오. 구성 요소는 디코딩된 페이로드와 함께 **OnMCPResponseResource** 를 발생시킵니다.

- **Request:** 대상 Uri가 있는 RequestResource.
- **이벤트:** OnMCPResponseResource(Sender, Request, Response).
- **Response:** TsgcAI_MCP_Response_ResourcesRead. 각 TsgcAI_MCP_Response_Resources_ReadResult_Content 블록에 접근하려면 Result.Contents를 읽으십시오. 이 블록에는 Uri, Name, Title, MimeType과 Text 또는 Blob 데이터 중 하나가 포함됩니다.

샘플 코드

이 스니펫은 리소스를 나열하고 단일 파일을 검색합니다.

```
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

MCP 사양은 클라이언트가 서버에 노출할 의향이 있는 파일 시스템 위치를 설명할 수 있도록 **roots** 기능을 도입합니다. 각 root는 선택적 표시 이름이 동반되는 file:// URI입니다. 서버는 roots/list로 카탈로그를 쿼리하고, 지원되는 경우 선택이 변경될 때마다 notifications/roots/list_changed 메시지를 받습니다. 이 핸드셰이크를 통해 도구는 사용자가 서버가 접근할 수 있는 것을 제어하도록 유지하면서 프로젝트 경계, 작업 공간 및 다중 리포지토리 레이아웃을 존중할 수 있습니다.

roots 이해하기

Roots는 서버 측 기능의 범위를 지정하는 가드레일 역할을 합니다. 클라이언트는 initialize 중에 roots 기능과 선택적으로 listChanged 플래그를 활성화하여 동적 업데이트를 푸시할 수 있도록 지원을 알립니다. roots/list에서 반환되는 일반적인 페이로드에는 각각 Uri와 Name을 가진 하나 이상의 항목이 포함되며, 작업 공간, 리포지토리 또는 기타 큐레이팅된 폴더를 나타냅니다.

roots/list 요청 처리

TsgcWSAPIClient_MCP 는 수신 JSON-RPC 메시지를 파싱하고 서버가 현재 카탈로그를 요청할 때마다 **OnMCPListRoots** 를 발생시킵니다. 이 이벤트를 사용하여 애플리케이션이 유지 관리하는 작업 공간 모델을 프로토콜 응답과 동기화하십시오.

- **Request:** TsgcAI_MCP_Request_RootsList. 메시지에는 매개변수가 포함되지 않지만, 응답에서 미러링해야 하는 JSON-RPC 식별자를 제공합니다.
- **이벤트:** OnMCPListRoots(Sender, Request, Response). 이것은 구성 요소가 세션을 검증한 후 트리거됩니다.
- **응답:** TsgcAI_MCP_Response_RootsList. Roots.AddRoot(Uri, Name)를 호출하거나(또는 TsgcAI_MCP_Root 인스턴스를 수동으로 빌드하여) 액세스 가능한 각 위치를 설명하십시오. 핸들러가 반환되면 구성 요소가 채워진 객체를 직렬화하므로 목록이 현재 상태를 반영하는지 확인하십시오. 진단 또는 로깅에는 Response.Write를 사용하십시오.

애플리케이션이 사용자가 런타임에 작업 공간을 추가하거나 제거할 수 있도록 하는 경우, 서버가 캐시된 카탈로그를 새로 고치도록 해당하는 notifications/roots/list_changed 알림을 발생시키는 것을 잊지 마십시오.

샘플 코드

다음 스니펫은 서버가 요청할 때 두 개의 root를 게시하고 다시 전송될 JSON 페이로드를 로깅합니다.

```
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

Sampling을 사용하면 MCP 서버가 공급자 자격 증명을 공유하지 않고 언어 모델 생성을 클라이언트에 위임할 수 있습니다. 서버는 대화 상태, 모델 기본 설정 및 maxTokens와 같은 선택적 제약을 캡슐화하는 `sampling/createMessage` 요청을 발행합니다. 클라이언트는 사람 검토자를 참여시키고, 가드레일을 적용하고, 궁극적으로 선택된 모델, 생성된 콘텐츠 및 중지 이유를 포함하는 응답을 반환할 수 있습니다.

샘플링 이해하기

initialize 중에 sampling 기능이 광고되면 서버는 워크플로 중간에 생성을 요청할 수 있습니다. 요청은 메시지 배열(텍스트, 이미지 또는 오디오 블록), systemPrompt, 선택적 토큰 제한 및 숫자 우선순위와 힌트 문자열을 통해 비용, 속도, 지능을 조정하는 ModelPreferences 구조를 묶습니다. 클라이언트는 이러한 힌트를 사용 가능한 모델에 매핑하고, 필요할 때 사용자에게 프롬프트하며, 응답하기 전에 안전하지 않은 완료를 필터링할 책임이 있습니다.

sampling/createMessage 요청 처리

`TsgcWSAPIClient_MCP` 는 `OnMCPSamplingCreateMessage`를 통해 서버가 시작한 생성을 표시합니다. 핸들러는 제공된 컨텍스트를 검사하고, 호출을 원하는 모델(또는 사람 검토자)로 라우팅하고, 응답 페이로드를 채워야 합니다.

- **Request:** `TsgcAI_MCP_Request_SamplingCreateMessage`. `Params.Messages`를 검사하여 대화를 재생하고, hint 우선순위를 위해 `Params.ModelPreferences`를, guardrail을 위해 `Params.SystemPrompt`를 검사하십시오.
- **Event:** `OnMCPSamplingCreateMessage(Sender, Request, Response)`. 구성 요소가 MCP 세션을 검증하고 요청을 역직렬화한 후에 발생합니다.
- **Response:** `TsgcAI_MCP_Response_SamplingCreateMessage`. 어시스턴트 Role을 제공하고, 콘텐츠 객체(텍스트, 이미지, 오디오 또는 리소스 링크)를 할당하며, 선택적으로 구체적인 Model과 StopReason을 기록하십시오. `Response.Write`를 호출하여 디스패치되기 전에 JSON 페이로드를 미리 볼 수 있습니다.

샘플 코드

아래 코드 조각은 들어오는 메시지 수를 기록하고, 고정 응답을 생성하며, 모델과 중지 이유를 주석으로 추가합니다. 프로덕션에서는 하드코딩된 내용을 실제 모델 호출 또는 사람 승인 루프로 대체하십시오.

```
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", ARRAYOFCONST((ARequest->Params->Messages->Count))), 0);
    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

Elicitation을 사용하면 서버가 MCP 클라이언트를 통해 사용자에게 구조화된 입력을 요청할 수 있습니다. 요청에는 사람이 읽을 수 있는 메시지와 예상 필드를 정의하는 제약된 JSON Schema가 포함됩니다. 클라이언트는 상호 작용에 대한 제어를 유지하고, 제공된 데이터를 검증하며, 사용자가 프롬프트를 수락, 거부 또는 취소했는지 나타내는 action을 반환합니다.

elicitation 이해하기

elicitation 기능은 initialize 중에 협상됩니다. 활성화되면 서버는 기본 유형(문자열, 숫자/정수, 부울 및 선택적 표시 이름이 있는 enum)으로 제한된 평면 객체 스키마를 포함하는 elicitation/create 호출을 보낼 수 있습니다. 각 속성은 표시 메타데이터, 길이 또는 범위 제약 조건 및 지원되는 형식(email, uri, date, date-time)을 지정할 수 있습니다. 사용자는 항상 누가 정보를 요청하는지 알 수 있어야 하고, 페이로드를 편집할 기회를 받아야 하며, 요청을 완전히 거부하거나 취소할 수 있어야 합니다.

elicitation/create 요청 처리

TsgcWSAPIClient_MCP 는 서버가 추가 입력을 요구할 때마다 **OnMCPElicitationCreate** 를 발생시킵니다. 이 이벤트는 프롬프트 메시지, 요청된 스키마, 그리고 사용자 결정에 따라 채울 수 있는 빈 응답 객체에 대한 액세스를 제공합니다.

- **요청:** TsgcAI_MCP_Request_ElicitationCreate. prompt를 표시하려면 Params.Message를 검사하고, UI 컨트롤이나 검증을 빌드하려면 Params.RequestedSchema.Properties를 검사하십시오.
- **Event:** OnMCPElicitationCreate(Sender, Request, Response). 클라이언트가 JSON-RPC 페이로드를 역직렬화한 후 트리거됩니다.
- **Response:** TsgcAI_MCP_Response_ElicitationCreate입니다. Action을 accept, decline 또는 cancel로 설정하십시오. 수락할 때는 수집된 값을 Content(TsgcJSON 헬퍼)에 작성하여 요청된 스키마와 일치하도록 하십시오. 최종 JSON 문서를 검사하려면 Response.Write를 사용하십시오.

샘플 코드

아래 예제는 사용자에게 GitHub 핸들을 묻습니다. operator가 값을 제공하면 응답이 수락되고 필드가 반환됩니다. 그렇지 않으면 요청이 거부됩니다.

```
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

OpenAI는 인공 지능(AI)을 인류 전체에 이익이 되는 방식으로 개발하고 이끄는 것을 목표로 하는 민간 연구소입니다. OpenAI는 다음 프로젝트를 개발했습니다:

- **GPT-3:** 이 강력한 언어 모델은 다른 OpenAI 제품의 기반이 됩니다. 사람이 생성한 텍스트를 분석하여 스스로 유사한 텍스트를 생성하는 법을 학습합니다.
- **DALL-E 및 DALL-E 2:** 이러한 생성형 AI 플랫폼은 사용자가 생성하기를 원하는 이미지의 텍스트 기반 설명을 분석한 다음 설명한 그대로 정확하게 해당 이미지를 생성할 수 있습니다.
- **CLIP:** CLIP은 시각 자료와 관련된 텍스트를 종합하여 해당 시각 자료를 가장 정확하게 설명하는 가능한 최상의 caption을 예측하는 신경망입니다. 둘 이상의 데이터 유형(이미지와 텍스트 모두)에서 학습할 수 있기 때문에 multimodal AI로 분류될 수 있습니다.
- **ChatGPT:** ChatGPT는 현재 사람과 같은 텍스트를 생성하고 사용자 질문에 대한 답변을 생성하도록 설계된 가장 진보된 AI 챗봇입니다. 대규모 데이터셋으로 학습되어 사람처럼 답변과 응답을 생성할 수 있습니다.
- **Codex:** Codex는 소프트웨어 개발자가 코딩 프로세스를 단순화하는 데 도움을 주기 위해 다양한 프로그래밍 언어로 된 수십억 줄의 코드로 학습되었습니다. GPT-3 기술에 기반을 두고 있지만, 텍스트를 생성하는 대신 코드를 생성합니다.
- **Whisper:** Whisper는 자동 음성 인식(ASR) 도구입니다. 기술 언어와 다양한 억양을 포함하여 약 100개의 서로 다른 언어로 음성을 인식, 전사 및 번역하도록 다수의 오디오 데이터로 훈련되었습니다.

OpenAI API

OpenAI API는 자연어, 코드 또는 이미지의 이해나 생성을 포함하는 거의 모든 작업에 적용할 수 있습니다. OpenAI는 서로 다른 작업에 적합한 다양한 수준의 성능을 가진 모델 스펙트럼과 함께 자체 사용자 지정 모델을 미세 조정할 수 있는 기능을 제공합니다. 이러한 모델은 콘텐츠 생성에서 시맨틱 검색 및 분류에 이르기까지 모든 용도에 사용할 수 있습니다.

Most common uses

- Completion
 - [OpenAI Completion 예제](#)
- Chat
 - [OpenAI Chat 예제](#)
- Edit
 - [OpenAI Edit 예제](#)
- Audio
 - [OpenAI Transcribe & Translate Examples](#)
- Moderation
 - [OpenAI Moderation 예제](#)
- RealTime
 - [OpenAI RealTime 예제](#)
- Responses
 - [OpenAI Responses 예제](#)
- Speech
 - [OpenAI Speech Examples](#)
- Fine-Tuning
 - [OpenAI Fine-Tuning 예제](#)
- Batch
 - [OpenAI Batch 예제](#)
- Uploads
 - [OpenAI Uploads 예제](#)

구성

OpenAI

OpenAI API는 인증을 위해 API 키를 사용합니다. 요청에서 사용할 API 키를 검색하려면 [API Keys](#) 페이지를 방문하십시오. API 키는 비밀이라는 점을 기억하십시오! 다른 사람과 공유하거나 클라이언트 측 코드(브라우저, 앱)에 노출하지 마십시오. 프로덕션 요청은 API 키를 환경 변수 또는 키 관리 서비스에서 안전하게 로드할 수 있는 자체 백엔드 서버를 통해 라우팅해야 합니다.

이 **API Key**는 구성 요소의 **OpenAIOptions.ApiKey** 속성에 구성되어야 합니다. 선택적으로, 여러 조직에 속한 사용자의 경우 계정이 조직에 속해 있으면 **OpenAIOptions.Organization** 속성에 Organization을 설정할 수 있습니다.

API Key가 구성되면, OpenAI API와 상호작용하기 위해 사용 가능한 함수 목록을 아래에서 확인하십시오.

Azure

클라이언트는 Microsoft Azure OpenAI Services를 지원하므로 Azure 계정을 사용하여 Azure OpenAI API와도 상호 작용할 수 있습니다. Azure와 작동하도록 클라이언트를 구성하려면 아래 단계를 따르십시오:

1. 속성 **OpenAIOptions.Provider** = oapvAzure를 구성하십시오
2. ResourceName과 DeploymentId의 값을 설정하십시오(이 값은 Azure Account에서 찾을 수 있습니다).
 1. **OpenAIOptions.AzureOptions.ResourceName** = <your resource name>.
 2. **OpenAIOptions.AzureOptions.DeploymentId** = <your deployment id>.
3. Azure 계정의 API Key를 설정하십시오
 1. **OpenAIOptions.ApiKey** = <azure api key>.

모든 OpenAI 메서드가 Azure에서 지원되는 것은 아니며, 현재 다음 메서드만 지원됩니다:

1. Completion
2. Chat Completion

Properties

OpenAIOptions

- **ApiKey:** OpenAI API로 인증하기 위한 API 키입니다.
- **Organization:** 여러 조직에 속한 사용자를 위한 선택적 조직 ID입니다.
- **Provider:** 공급자를 선택하십시오: oapvOpenAI(기본값) 또는 Microsoft Azure OpenAI Services의 경우 oapvAzure.
- **AzureOptions:** Azure OpenAI Services에 대한 구성.
 - **ResourceName:** Azure 리소스 이름.
 - **DeploymentId:** Azure 배포 ID.
 - **APIVersion:** Azure API 버전.
- **HttpOptions:** HTTP 연결 설정.
 - **ReadTimeout:** HTTP 응답을 읽기 위한 타임아웃(밀리초). 기본값은 0(타임아웃 없음)입니다.
- **LogOptions:** 로깅 구성입니다.
 - **Enabled:** True인 경우 HTTP 요청 및 응답이 파일에 기록됩니다.
 - **FileName:** 로그 출력이 기록되는 파일 경로입니다.
- **RetryOptions:** 실패한 API 요청에 대한 자동 재시도 구성입니다.
 - **Enabled:** True인 경우 실패한 요청이 자동으로 재시도됩니다.
 - **Retries:** 최대 재시도 횟수.
 - **Wait:** 재시도 시도 간 대기 시간(밀리초).

모델

API에서 사용 가능한 다양한 모델을 나열하고 설명합니다.

- **GetModels:** 현재 사용 가능한 모델을 나열하고, 소유자 및 가용성과 같은 각 모델에 대한 기본 정보를 제공합니다.
- **GetModel:** 모델 인스턴스를 검색하여 소유자 및 권한과 같은 모델에 대한 기본 정보를 제공합니다.
 - **Model:** 이 요청에 사용할 모델의 ID입니다

Completions

프롬프트가 주어지면 모델은 하나 이상의 예측된 완성을 반환하며, 각 위치에서 대체 토큰의 확률도 반환할 수 있습니다.

- **CreateCompletion:** 제공된 프롬프트와 매개변수에 대한 completion을 생성합니다
 - **Model:** 사용할 모델의 ID입니다. List models API를 사용하여 사용 가능한 모든 모델을 보거나 Model overview에서 설명을 볼 수 있습니다.
 - **Prompt:** completion을 생성할 프롬프트입니다.

Chat

채팅 대화가 주어지면 모델이 chat completion 응답을 반환합니다.

- **Model:** 사용할 모델의 ID. Chat API가 지원하는 모든 모델 목록을 가져오려면 GetModels를 호출하십시오.
- **Message:** 채팅 완성을 생성할 메시지입니다.
- **Role:** 기본적으로 user이며, 다른 옵션은 system, assistant입니다.

Edits

prompt와 instruction이 주어지면 모델은 편집된 버전의 prompt를 반환합니다.

- **CreateEdit:** 제공된 입력, 명령 및 매개변수에 대한 새 편집을 생성합니다.
 - **Model:** 사용할 모델의 ID입니다. 이 엔드포인트에서 text-davinci-edit-001 또는 code-davinci-edit-001 모델을 사용할 수 있습니다.
 - **Instruction:** 모델에게 프롬프트를 어떻게 편집할지 알려주는 명령입니다.
 - **Input:** (선택적) 편집의 시작점으로 사용할 입력 텍스트입니다.

Images

prompt 및/또는 입력 이미지가 주어지면 모델이 새 이미지를 생성합니다.

- **CreateImage:** 프롬프트가 주어지면 이미지를 생성합니다.
 - **Prompt:** 원하는 이미지의 텍스트 설명입니다. 최대 길이는 1000자입니다.
- **CreateImageEdit:** 원본 이미지와 프롬프트가 주어지면 편집되거나 확장된 이미지를 생성합니다.
 - **Image:** 편집할 이미지. 유효한 PNG 파일이어야 하며, 4MB 미만이고, 정사각형이어야 합니다. mask가 제공되지 않으면 이미지에 투명도가 있어야 하며, 이것이 mask로 사용됩니다.
 - **Prompt:** 원하는 이미지에 대한 텍스트 설명. 최대 길이는 1000자입니다.
- **CreateImageVariations:** 지정된 이미지의 변형을 생성합니다.
 - **Image:** 변형의 기초로 사용할 이미지입니다. 유효한 PNG 파일이어야 하고, 4MB 미만이며, 정사각형이어야 합니다.

Embeddings

머신 러닝 모델과 알고리즘이 쉽게 사용할 수 있는 주어진 입력의 벡터 표현을 가져옵니다.

- **CreateEmbeddings:** 입력 텍스트를 나타내는 임베딩 벡터를 생성합니다.
 - **Model:** 사용할 모델의 ID.
 - **Input:** 임베딩을 얻을 입력 텍스트.

Audio

오디오를 텍스트로 변환합니다.

- **CreateTranscriptionFromFile**: 파일명에서 오디오를 입력 언어로 변환합니다.
 - **Model**: 사용할 모델의 ID입니다. 현재 whisper-1만 사용할 수 있습니다.
 - **Filename**: 다음 형식 중 하나로 변환할 오디오 파일입니다: mp3, mp4, mpeg, mpga, m4a, wav 또는 webm.
- **CreateTranscription**: X초 동안 오디오를 녹음하고 이를 텍스트로 변환합니다.
 - **Model**: 사용할 모델의 ID입니다. 현재 whisper-1만 사용할 수 있습니다.
 - **Time**: 시간(밀리초)이며, 기본값은 10초입니다.
- **CreateTranslationFromFile**: 오디오를 영어로 번역합니다.
 - **Model**: 사용할 모델의 ID입니다. 현재 whisper-1만 사용할 수 있습니다.
 - **Filename**: 번역할 오디오 파일이며, 다음 형식 중 하나입니다: mp3, mp4, mpeg, mpga, m4a, wav 또는 webm.
- **CreateTranslation**: X초 동안 오디오를 녹음하고 번역합니다.
 - **Model**: 사용할 모델의 ID입니다. 현재 whisper-1만 사용 가능합니다.
 - **Time**: 시간(밀리초), 기본값 10초.

Files

파일은 Fine-tuning과 같은 기능과 함께 사용할 수 있는 문서를 업로드하는 데 사용됩니다.

- **ListFiles**: 사용자 조직에 속한 파일 목록을 반환합니다.
- **UploadFile**: 다양한 엔드포인트/기능에서 사용할 문서가 포함된 파일을 업로드합니다. 현재 한 조직이 업로드한 모든 파일의 크기는 최대 1GB까지 가능합니다.
 - **Filename**: 업로드할 JSON Lines 파일의 이름입니다. purpose가 "fine-tune"으로 설정된 경우 각 줄은 학습 예제를 나타내는 "prompt" 및 "completion" 필드가 있는 JSON 레코드입니다.
 - **Purpose**: 업로드된 문서의 의도된 용도입니다. Fine-tuning에는 "fine-tune"을 사용하십시오.
- **DeleteFile**: 파일을 삭제합니다.
 - **FileId**: 이 요청에 사용할 파일의 ID입니다.
- **RetrieveFile**: 특정 파일에 대한 정보를 반환합니다.
 - **FileId**: 이 요청에 사용할 파일의 ID
- **RetrieveFileContent**: 지정된 파일의 내용을 반환합니다.
 - **FileId**: 이 요청에 사용할 파일의 ID.

Fine-Tunes

특정 학습 데이터에 맞게 모델을 조정하기 위해 파인튜닝 작업을 관리합니다.

- **CreateFineTune**: 주어진 데이터셋에서 지정된 모델을 fine-tune하는 작업을 생성합니다. 응답에는 작업 상태와 완료 시 fine-tune된 모델의 이름을 포함하여 대기열에 추가된 작업의 세부 정보가 포함됩니다.
 - **TrainingFile**: 학습 데이터가 포함된 업로드된 파일의 ID입니다.
- **ListFineTunes**: 조직의 파인튜닝 작업을 나열합니다
- **RetrieveFineTune**: fine-tune job에 대한 정보를 가져옵니다.
 - **FineTuneId**: fine-tune job의 ID
- **CancelFineTune**: fine-tune 작업을 즉시 취소합니다.
 - **FineTuneId**: fine-tune 작업의 ID
- **ListFineTuneEvents**: fine-tune 작업에 대한 세분화된 상태 업데이트를 가져옵니다.
 - **FineTuneId**: fine-tune 작업의 ID입니다
- **DeleteFineTuneModel**: fine-tuned 모델을 삭제합니다. 조직에서 Owner 역할이 있어야 합니다.
 - **Model**: 삭제할 모델입니다.

Moderations

입력 텍스트가 주어지면 모델이 OpenAI의 콘텐츠 정책 위반으로 분류하는지 여부를 출력합니다.

- **CreateModeration**: 텍스트가 OpenAI의 콘텐츠 정책을 위반하는지 분류합니다.
 - **Input**: 분류할 입력 텍스트

RealTime

OpenAI Realtime API는 음성 대 음성 대화 경험 및 실시간 전사를 포함한 저지연 멀티모달 상호 작용을 가능하게 합니다.

- **Transcription:** 마이크 입력 또는 파일 입력으로 전사 전용 사용 사례에 Realtime API를 사용할 수 있습니다. 예를 들어, 자막이나 전사본을 실시간으로 생성하는 데 사용할 수 있습니다. 전사 전용 모드에서는 모델이 응답을 생성하지 않습니다.

Assistants

모델을 호출하고 도구를 사용하여 작업을 수행할 수 있는 AI assistant를 빌드합니다.

- **CreateAssistant:** 모델과 지침으로 어시스턴트를 생성합니다.
- **ListAssistants:** assistant 목록을 반환합니다.
- **RetrieveAssistant:** ID로 assistant를 검색합니다.
- **ModifyAssistant:** 기존 assistant를 수정합니다.
- **DeleteAssistant:** assistant를 삭제합니다.

Threads

Thread는 대화 세션을 나타냅니다. 메시지가 thread에 추가되고, 이후 run에 의해 처리됩니다.

- **CreateThread:** 새 thread를 생성합니다.
- **RetrieveThread:** ID로 thread를 검색합니다.
- **ModifyThread:** 스레드를 수정합니다.
- **DeleteThread:** 스레드를 삭제합니다.

Thread Messages

메시지는 thread에 추가되며 대화의 콘텐츠를 포함합니다.

- **CreateMessage:** 스레드 내에서 메시지를 생성합니다.
 - **ThreadId:** 스레드의 ID입니다.
- **ListMessages:** 주어진 thread에 대한 메시지 목록을 반환합니다.
- **RetrieveMessage:** thread 및 message ID로 메시지를 검색합니다.
- **ModifyMessage:** 메시지를 수정합니다.
- **DeleteMessage:** thread에서 메시지를 삭제합니다.

Runs

Run은 assistant와 함께 thread에서의 실행을 나타냅니다. assistant는 자신의 구성과 thread 메시지를 사용하여 모델과 도구를 호출함으로써 작업을 수행합니다.

- **CreateRun:** 지정된 assistant로 thread에 대한 run을 생성합니다.
- **ListRuns:** thread에 속한 run 목록을 반환합니다.
- **RetrieveRun:** thread 및 run ID로 run을 검색합니다.
- **ModifyRun:** run을 수정합니다.
- **SubmitToolOutputsToRun:** 조치가 필요한 run에 대한 도구 출력을 제출합니다.
- **CancelRun:** 진행 중인 run을 취소합니다.

Run Steps

Run 단계는 run 실행 중에 수행된 개별 단계를 나타냅니다.

- **ListRunSteps:** run에 속한 run step 목록을 반환합니다.
- **RetrieveRunSteps:** thread, run 및 step ID로 run step을 검색합니다.

Vector Stores

벡터 저장소는 retrieval-augmented generation(RAG)을 위해 embedding을 사용하여 파일을 저장하고 검색하는 데 사용됩니다.

- **CreateVectorStore:** 벡터 저장소를 생성합니다.
- **ListVectorStores:** 벡터 저장소 목록을 반환합니다.
- **RetrieveVectorStore:** ID로 vector store를 검색합니다.
- **ModifyVectorStore:** 벡터 저장소를 수정합니다.
- **DeleteVectorStore:** 벡터 저장소를 삭제합니다.

Vector Store Files

벡터 스토어 내의 파일을 관리합니다.

- **CreateVectorStoreFile:** 벡터 저장소에 파일을 첨부합니다.
- **ListVectorStoreFiles:** vector store의 파일 목록을 반환합니다.
- **RetrieveVectorStoreFile:** 벡터 저장소 파일을 검색합니다.
- **DeleteVectorStoreFile:** 벡터 스토어에서 파일을 제거합니다.

Vector Store File Batches

벡터 저장소에 파일을 추가하기 위한 Batch 작업입니다.

- **CreateVectorStoreFileBatch:** vector store에 여러 파일을 추가하는 batch를 만듭니다.
- **RetrieveVectorStoreFileBatch:** ID로 파일 배치를 검색합니다.
- **CancelVectorStoreFileBatch:** 진행 중인 파일 배치를 취소합니다.
- **ListVectorStoreFilesBatch:** 배치의 파일을 나열합니다.

Speech

text-to-speech 모델을 사용하여 텍스트에서 음성 오디오를 생성합니다.

- **CreateSpeech:** 입력 텍스트에서 오디오를 생성합니다. 오디오 데이터를 문자열로 반환하거나 응답 스트림에 기록합니다.
 - **Model:** 사용할 TTS 모델입니다(예: tts-1, tts-1-hd).
 - **Input:** 오디오를 생성할 텍스트입니다.
 - **Voice:** 사용할 음성입니다(alloy, echo, fable, onyx, nova, shimmer).

Fine-Tuning Jobs

사용자 정의 모델을 생성하기 위해 fine-tuning 작업을 관리합니다. 이것은 레거시 Fine-Tunes 엔드포인트를 대체하는 새로운 fine-tuning API입니다.

- **CreateFineTuningJob:** 학습 파일에서 파인튜닝 작업을 생성합니다.
- **ListFineTuningJobs:** 조직의 fine-tuning 작업을 나열합니다.
- **RetrieveFineTuningJob:** fine-tuning 작업에 대한 정보를 검색합니다.
- **CancelFineTuningJob:** 파인튜닝 작업을 취소합니다.
- **ListFineTuningJobEvents:** 파인튜닝 작업의 상태 업데이트를 나열합니다.
- **ListFineTuningJobCheckpoints:** fine-tuning 작업의 checkpoint를 나열합니다.

Responses

모델 응답을 생성하고 관리합니다. Responses API는 멀티턴 대화, 도구 사용 및 구조화된 출력을 지원합니다.

- **CreateResponse:** 모델 응답을 생성합니다.
- **RetrieveResponse:** ID로 응답을 검색합니다.
- **DeleteResponse:** 응답을 삭제합니다.
- **CancelResponse:** 진행 중인 응답을 취소합니다.
- **ListInputItems:** 응답에 대한 입력 항목을 나열합니다.

Batches

더 낮은 비용으로 비동기 처리를 위한 batch API 요청을 생성하고 관리합니다.

- **CreateBatch:** 배치 작업을 생성합니다.
- **RetrieveBatch:** ID로 배치를 검색합니다.

- **ListBatches:** 모든 배치를 나열합니다.
- **CancelBatch:** 진행 중인 배치를 취소합니다.

Uploads

큰 파일을 부분으로 업로드합니다. 표준 업로드 크기 제한을 초과하는 파일에 유용합니다.

- **CreateUpload:** 업로드 세션을 생성합니다.
- **AddUploadPart:** 업로드에 부분을 추가합니다.
- **CompleteUpload:** 멀티파트 업로드를 완료하고 파일을 생성합니다.
- **CancelUpload:** 진행 중인 업로드를 취소합니다.

TsgcHTTP_API_OpenAI

Chat Completions, Images, Audio(TTS/STT), Embeddings, Files, Fine-tuning, Moderation, Assistants, Threads, Runs, Messages 및 Vector Stores를 다루는 OpenAI REST API 클라이언트입니다.

소개

Chat Completions, Images, Audio(TTS/STT), Embeddings, Files, Fine-tuning, Moderation, Assistants, Threads, Runs, Messages 및 Vector Stores를 다루는 OpenAI REST API 클라이언트입니다.

소개

Chat Completions, Images, Audio(TTS/STT), Embeddings, Files, Fine-tuning, Moderation, Assistants, Threads, Runs, Messages 및 Vector Stores를 다루는 OpenAI REST API 클라이언트입니다.

API 키와 선택적 HTTP/재시도/로그 설정으로 **OpenAIOptions** 속성을 구성한 다음, 요청 객체와 함께 `CreateMessage/`
`CreateChatCompletion`과 같은 형식화된 메서드를 호출하고 형식화된 응답 클래스를 받으십시오.

공급자에 대한 배경 정보는 [OpenAI](#) 가이드를 참조하십시오.

참조

- 속성 — 구성 요소에서 사용할 수 있는 5개의 속성.
- 메서드 — 78개의 public 메서드.
- 이벤트 — 처리할 수 있는 2개의 이벤트.

OpenAI | Completion

프롬프트가 주어지면 모델은 하나 이상의 예측된 completion을 반환하며, 각 위치에서 대체 token의 확률도 반환할 수 있습니다.

간단한 예제

text-davinci-003 모델을 사용하여 예측된 completion을 가져옵니다.

```
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"));
```

고급 예제

text-davinci-003 모델을 사용하여 더 무작위적인 출력으로 예측 completion을 얻고 각 프롬프트에 대해 2개의 completion을 생성합니다.

```
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

채팅 대화가 주어지면 모델이 chat completion 응답을 반환합니다.

간단한 예제

Hello 메시지를 보내 ChatGPT와 상호 작용합니다.

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";
ShowMessage(OpenAI->_CreateChatCompletion("gpt-3.5-turbo", "Hello!"));
```

고급 예제

gpt-3-5 모델을 사용하여 더 무작위적인 출력으로 채팅하고 각 프롬프트에 대해 2개의 completion을 생성합니다.

```
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

prompt와 instruction이 주어지면, 모델은 prompt의 편집된 버전을 반환합니다.

간단한 예제

OpenAI에 프롬프트의 철자 오류를 수정하도록 지시합니다.

```
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?"));
```

고급 예제

OpenAI에 프롬프트의 철자 오류를 더 무작위적인 출력으로 수정하고 각 프롬프트에 대해 2개의 completion을 생성하도록 지시합니다.

```
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 | Audio

Create Transcription

오디오를 입력 언어로 전사합니다.

```
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

오디오를 영어로 번역합니다.

```
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

입력 텍스트가 주어지면, 모델이 OpenAI의 콘텐츠 정책을 위반하는 것으로 분류하는지 여부를 출력합니다.

간단한 예제

다음 텍스트를 검토하십시오

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";

ShowMessage(OpenAI->_CreateModeration("I want to kill them."));
```

고급 예제

모델을 선택하여 다음 텍스트를 검열합니다.

```
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

마이크 입력이나 파일 입력으로 전사 전용 사용 사례에 Realtime API를 사용할 수 있습니다. 예를 들어 실시간으로 자막이나 전사본을 생성하는 데 사용할 수 있습니다. 전사 전용 모드에서는 모델이 응답을 생성하지 않습니다.

전사를 위해 Realtime API를 사용하려면 WebSockets를 통해 연결하여 전사 세션을 생성해야 합니다. 새 전사 세션을 시작하려면 **TsgcWSAPI_OpenAI** 및 **TsgcWebSocketClient** 구성 요소를 사용하십시오.

아래에서 OpenAI API를 사용한 실시간 전사의 예제를 확인할 수 있습니다.

```
TsgcWebSocketClient *WSClient = new TsgcWebSocketClient(NULL);
TsgcAudioRecorderWave *oAudio = new TsgcAudioRecorderWave(NULL);
TsgcWSAPI_OpenAI *OpenAI = new TsgcWSAPI_OpenAI(NULL);
OpenAI->Client = WSClient;
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

구성 요소를 사용하면 오디오 파일을 수동으로 보낼 수 있습니다. AppendInputAudioBuffer 메서드를 사용하고 오디오를 TStream으로 전달할 수 있습니다. 오디오 형식은 24 kHz 모노 PCM이어야 합니다(24000 비율만 지원됨).

OpenAI | Responses

Responses API는 OpenAI 모델과 상호 작용하기 위한 핵심 API입니다. 텍스트 및 이미지 입력, 도구 사용, 스트리밍 및 구조화된 출력을 지원합니다.

간단한 예제

Responses API를 사용하여 응답을 생성합니다.

```
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?"));
```

Methods

- **CreateResponse:** 모델과 입력이 주어지면 모델 응답을 생성합니다.
 - **Model:** 사용할 모델의 ID(예: gpt-4o, gpt-4o-mini).
 - **Input:** 모델에 대한 텍스트 또는 구조화된 입력.
- **RetrieveResponse:** 이전에 생성된 응답을 ID로 검색합니다.
 - **ResponseId:** 검색할 응답의 ID입니다.
- **DeleteResponse:** ID로 응답을 삭제합니다.
 - **ResponseId:** 삭제할 응답의 ID.
- **CancelResponse:** 진행 중인 응답을 취소합니다.
 - **ResponseId:** 취소할 응답의 ID입니다.
- **ListInputItems:** 주어진 응답에 대한 입력 항목 목록을 반환합니다.
 - **ResponseId:** 응답의 ID.

OpenAI | Speech

Text-to-Speech (TTS) API를 사용하여 텍스트에서 음성 오디오를 생성합니다. 여러 음성과 출력 형식을 지원합니다.

간단한 예제

텍스트에서 음성을 생성하여 파일 스트림에 저장합니다.

```
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;
}
```

Methods

- **CreateSpeech:** 입력 텍스트로부터 오디오를 생성합니다.
 - **Model:** 사용할 모델의 ID(tts-1 또는 tts-1-hd).
 - **Input:** 오디오를 생성할 텍스트. 최대 4096자.
 - **Voice:** 사용할 음성(alloy, echo, fable, onyx, nova, shimmer).
 - **ResponseStream:** 오디오 데이터가 작성될 스트림.
 - **ResponseFormat:** (선택 사항) 오디오 형식: mp3(기본값), opus, aac 또는 flac.
 - **Speed:** (선택 사항) 생성된 오디오의 속도(0.25 ~ 4.0, 기본값 1.0).

OpenAI | Fine-Tuning

fine-tuning 작업을 관리하여 모델을 특정 학습 데이터에 맞게 조정합니다. Fine-tuning을 사용하면 더 높은 품질의 결과, 더 많은 예제로 학습할 수 있는 능력 및 비용 절감을 제공하여 모델을 더 잘 활용할 수 있습니다.

간단한 예제

fine-tuning 작업을 생성하고 기존 작업을 나열합니다.

```
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());
```

Methods

- **CreateFineTuningJob:** 주어진 데이터셋에서 새 모델을 생성하는 프로세스를 시작하는 fine-tuning 작업을 생성합니다.
 - **Model:** fine-tune할 모델의 이름입니다.
 - **TrainingFile:** 학습 데이터가 포함된 업로드된 파일의 ID입니다.
- **ListFineTuningJobs:** 조직의 파인튜닝 작업을 나열합니다.
- **RetrieveFineTuningJob:** 파인튜닝 작업에 대한 정보를 가져옵니다.
 - **JobId:** 파인튜닝 작업의 ID.
- **CancelFineTuningJob:** fine-tuning 작업을 즉시 취소합니다.
 - **JobId:** 취소할 fine-tuning 작업의 ID입니다.
- **ListFineTuningJobEvents:** fine-tuning 작업에 대한 상태 업데이트를 가져옵니다.
 - **JobId:** fine-tuning 작업의 ID입니다.
- **ListFineTuningJobCheckpoints:** fine-tuning 작업의 checkpoint를 나열합니다.
 - **JobId:** fine-tuning 작업의 ID입니다.

OpenAI | Batch

비동기적으로 실행할 대규모 API 요청 배치를 생성합니다. Batch API는 50% 할인된 가격으로 24시간 이내에 완료 항목을 반환합니다.

간단한 예제

batch job을 생성하고 그 상태를 검색합니다.

```
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());
```

Methods

- **CreateBatch:** 업로드된 요청 파일에서 배치를 생성하고 실행합니다.
 - **InputFileId:** 배치에 대한 요청을 포함하는 업로드된 파일의 ID입니다.
 - **Endpoint:** 배치의 모든 요청에 사용할 엔드포인트입니다(예: /v1/chat/completions).
 - **CompletionWindow:** (선택적) 배치가 처리되어야 하는 시간 프레임입니다. 기본값은 24h입니다.
- **RetrieveBatch:** ID로 배치를 검색합니다.
 - **BatchId:** 검색할 배치의 ID입니다.
- **ListBatches:** 조직의 배치를 나열합니다.
- **CancelBatch:** 진행 중인 배치를 취소합니다.
 - **BatchId:** 취소할 배치의 ID입니다.

OpenAI | Uploads

큰 파일을 부분으로 업로드합니다. Uploads API를 사용하면 512 MB보다 큰 파일을 부분으로 분할하여 각 부분을 별도로 업로드할 수 있습니다.

간단한 예제

업로드를 생성하고, 파트를 추가하고, 완료합니다.

```
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\"]"));
```

Methods

- **CreateUpload:** 멀티파트 업로드를 시작하기 위한 중간 Upload 객체를 생성합니다.
 - **Filename:** 업로드할 파일의 이름입니다.
 - **Purpose:** 업로드된 파일의 의도된 용도입니다(예: fine-tune).
 - **Bytes:** 파일의 바이트 수입니다.
 - **MimeType:** 파일의 MIME 유형입니다(예: application/jsonl).
- **AddUploadPart:** Upload 객체에 part를 추가합니다.
 - **UploadId:** Upload의 ID.
 - **Filename:** 이 part의 바이트 청크.
- **CompleteUpload:** 업로드를 완료합니다. 업로드된 파일은 완료 후에만 사용할 수 있습니다.
 - **UploadId:** 업로드의 ID입니다.
 - **PartIds:** JSON 배열 문자열 형태의 Part ID 정렬된 목록입니다.
- **CancelUpload:** Upload를 취소합니다. 취소 후에는 부분을 추가할 수 없습니다.
 - **UploadId:** 취소할 Upload의 ID.

OpenAI Applications

개요

OpenAI API를 사용하면 다양한 애플리케이션을 만들 수 있으며, 다음은 몇 가지 예입니다:

- **챗봇 및 가상 비서:** 자연스럽고 사람과 같은 방식으로 인간과 대화할 수 있는 애플리케이션입니다. 고객 지원, 쿼리 처리, 웹사이트 또는 모바일 앱에서 정보 제공에 사용할 수 있습니다.
- **콘텐츠 생성:** 기사, 블로그 게시물 또는 보고서와 같은 사람과 유사한 텍스트를 생성할 수 있는 애플리케이션입니다. 예를 들어, GPT-3은 소셜 미디어용 콘텐츠 제작을 자동화하거나, 코드를 생성하거나, SEO 친화적인 콘텐츠를 만드는 데 사용될 수 있습니다.
- **번역 서비스:** 텍스트를 한 언어에서 다른 언어로 번역할 수 있는 애플리케이션입니다.
- **개인 교습 및 교육:** AI는 개인화된 학습 경험을 만들고, 숙제를 돕거나, 복잡한 개념을 쉬운 언어로 설명하는 데 사용할 수 있습니다.
- **Games:** OpenAI는 몰입형이고 인터랙티브한 게임, 특히 대화형 캐릭터나 복잡한 내러티브를 포함하는 게임을 만드는 데 사용할 수 있습니다.
- **감정 분석:** 텍스트 데이터에 표현된 감정을 분석하고 분류하는 것은 시장 조사, 브랜드 모니터링 및 고객 피드백 이해에 유용할 수 있습니다.
- **개인화된 추천:** 사용자의 과거 행동과 선호도를 기반으로, AI는 제품, 서비스 또는 콘텐츠에 대한 개인화된 추천을 생성할 수 있습니다.
- **Text Completion:** 이메일 클라이언트나 워드 프로세싱 소프트웨어에서 사용자의 문장을 완성하거나 작성 지원을 돕습니다.
- **Speech Recognition:** 음성 언어를 텍스트로 전사합니다. 전사 서비스, 음성 도우미 등에 유용합니다.
- **의료 및 법률 자문:** 전문적인 조언을 대체할 수는 없지만, AI 모델은 주어진 입력을 기반으로 예비 지침이나 제안을 제공할 수 있습니다.

구성 요소

사용 가능한 구성 요소 목록과 그에 대한 간단한 설명은 아래를 참조하십시오.

- **TsgcAIOpenAIAssistant:** Assistant는 지침을 가지며 모델, 도구 및 파일을 활용하여 사용자 쿼리에 응답할 수 있습니다.
- **TsgcAIOpenAIChatBot:** 음성을 듣고 OpenAI Whisper API를 사용하여 이를 텍스트로 변환하는 ChatBot입니다. 이 텍스트는 ChatCompletion API로 전송되어 OpenAI로부터 응답을 제공하며, 이 응답은 텍스트에서 음성으로 변환됩니다.
- **TsgcAIOpenAITranslator:** 모든 언어 음성을 영어로 번역하고 사용 가능한 모든 TextToSpeech 구성 요소를 사용하여 번역을 들을 수 있는 Translator 애플리케이션입니다.
- **TsgcAIOpenAIEmbeddings:** 이 구성 요소는 사용자 정의 데이터로 AI 애플리케이션을 구축하는 데 사용됩니다. 예: 제품 데이터가 있는 챗봇. 다음 데이터베이스가 지원됩니다:
 - **TsgcAIDatabaseVectorPinecone:** Pinecone Database Vector를 지원합니다.

다음 구성 요소는 마이크에서 오디오를 캡처하고, 오디오 파일을 재생하고, 텍스트를 음성으로 변환하는 데 사용됩니다.

- **TsgcAudioRecorderMCI:** (Windows 전용) 이 구성 요소를 사용하면 마이크에 액세스하고 음성을 wave 파일로 변환할 수 있습니다.
- **TsgcAudioPlayerMCI:** (Windows 전용) 이 구성 요소를 사용하면 mp3 파일을 재생할 수 있습니다.
- **TsgcTextToSpeechSystem:** (Windows 전용) 외부 mp3 파일을 사용할 필요 없이 텍스트를 음성으로 변환합니다.
- **TsgcTextToSpeechGoogle:** 사용 가능한 모든 Google Cloud 음성을 사용하여 텍스트를 음성으로 변환합니다.
- **TsgcTextToSpeechAmazon:** 사용 가능한 Amazon AWS 음성을 사용하여 텍스트를 음성으로 변환합니다.

TsgcAIOpenAIAssistant

Assistants API를 사용하면 자체 애플리케이션 내에서 AI 어시스턴트를 구축할 수 있습니다.

소개

Assistants API를 사용하면 자체 애플리케이션 내에서 AI 도우미를 구축할 수 있습니다. Assistant에는 지침이 있으며 사용자 쿼리에 응답하기 위해 모델, 도구 및 파일을 활용할 수 있습니다. Assistants API는 현재 세 가지 유형의 도구를 지원합니다: Code Interpreter, File Search 및 Function calling.

개요

Assistants API의 일반적인 통합은 다음 흐름을 가집니다:

- 사용자 지정 지침을 정의하고 모델을 선택하여 Assistant를 생성하십시오. 도움이 되는 경우 파일을 추가하고 Code Interpreter, File Search, Function calling과 같은 도구를 활성화하십시오.
- 사용자가 대화를 시작할 때 Thread를 생성합니다.
- 사용자가 질문할 때 Thread에 메시지를 추가합니다.
- 모델과 도구를 호출하여 응답을 생성하기 위해 Thread에서 Assistant를 실행합니다.

1단계: Assistant 생성

Assistant는 model, instructions, tools 같은 여러 매개변수를 사용하여 사용자의 메시지에 응답하도록 구성할 수 있는 엔터티를 나타냅니다.

```
// 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();
```

2단계: Thread 생성

Thread는 사용자와 하나 이상의 Assistant 간의 대화를 나타냅니다. 사용자(또는 AI 애플리케이션)가 Assistant와 대화를 시작할 때 Thread를 생성할 수 있습니다.

```
oThread = oAssistant->CreateThread();
```

3단계: Thread에 Message 추가 및 Run

사용자나 애플리케이션이 생성하는 메시지의 내용은 Message 객체로 Thread에 추가됩니다. Messages에는 텍스트와 파일이 모두 포함될 수 있습니다. Threads에 추가할 수 있는 Messages 수에는 제한이 없습니다 — 모델의 컨텍스트 창에 맞지 않는 컨텍스트는 지능적으로 잘립니다.

모든 사용자 Message가 Thread에 추가되면 어떤 Assistant로든 Thread를 Run할 수 있습니다. Run을 생성하면 Assistant와 연관된 모델과 도구를 사용하여 응답을 생성합니다. 이러한 응답은 assistant Message로 Thread에 추가됩니다.

```
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"));
                }
            }
        }
    }
}
```

참조

- 속성 — 구성 요소에서 사용 가능한 5개의 속성.
- 메서드 — 9개의 public 메서드.
- 이벤트 — 처리할 수 있는 8개의 이벤트입니다.

TsgcAIOpenAIAssistant | File Search

File Search는 독점 제품 정보나 사용자가 제공한 문서와 같은 외부 소스의 지식을 통합하여 Assistant의 기능을 향상시킵니다. 작동 방식은 다음과 같습니다:

- **Document Processing:** OpenAI는 문서를 자동으로 처리하고 관리 가능한 정보 조각으로 분할("청크")합니다.
- **임베딩 생성:** 이 청크는 임베딩, 즉 콘텐츠의 수학적 표현으로 변환되어 의미와 컨텍스트를 포착합니다.
- **저장 및 검색:** 시스템은 이러한 embeddings를 저장하여 벡터 검색(시맨틱적으로 유사한 콘텐츠 찾기)과 키워드 검색(정확한 일치 항목 찾기)을 모두 가능하게 합니다.

이 조합은 Assistant가 관련 콘텐츠를 검색하고 활용하여 사용자 쿼리에 대해 정확하고 충분한 정보에 기반한 답변을 제공할 수 있도록 보장합니다.

1단계: File Search가 활성화된 새 Assistant 생성

Assistant의 tools 매개변수에서 file_search가 활성화된 새 어시스턴트를 생성합니다. file_search 도구가 활성화되면 모델은 사용자 메시지를 기반으로 콘텐츠를 검색할 시기를 결정합니다.

```
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;
```

2단계: 파일을 업로드하고 Vector Store에 추가

파일에 액세스하기 위해 file_search 도구는 Vector Store 객체를 사용합니다. 파일을 업로드하고 이를 담을 Vector Store를 생성하십시오.

```
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
    }
}
```

3단계: run을 생성하고 출력을 확인합니다

이제 Run을 생성하고 모델이 사용자의 질문에 응답하기 위해 File Search tool을 사용하는 것을 관찰하십시오.

```
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

어시스턴트의 전체 응답을 기다리는 대신 Server-Sent Events를 사용하여 응답을 스트리밍할 수 있습니다. **CreateRun** 함수를 사용할 때 **Stream = True** 매개변수를 전달하기만 하면 응답이 스트리밍을 사용합니다.

다음 이벤트는 스트리밍 응답을 처리하는 데 사용됩니다:

- **OnStreamRun:** 이 이벤트는 run 객체에 업데이트가 있을 때 호출됩니다.
- **OnStreamMessage:** 메시지 객체에 업데이트가 있을 때 이벤트가 호출됩니다: created, in-progress, completed...
- **OnStreamMessageDelta:** Message의 일부가 스트리밍될 때 발생합니다.
- **OnStreamDone:** 스트림이 종료될 때 발생합니다.
- **OnStreamError:** 오류가 발생할 때 발생합니다. 이는 내부 서버 오류 또는 시간 초과로 인해 발생할 수 있습니다.

1단계: Assistant 생성

Assistant는 model, instructions, tools 같은 여러 매개변수를 사용하여 사용자의 메시지에 응답하도록 구성할 수 있는 엔터티를 나타냅니다.

```
// 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 n";
oAssistant->AssistantOptions->Model = "gpt-4o";
// Create the assistant
oAssistant->CreateAssistant();
```

2단계: Thread 생성

Thread는 사용자와 하나 이상의 Assistant 간의 대화를 나타냅니다. 사용자(또는 AI 애플리케이션)가 Assistant와 대화를 시작할 때 Thread를 생성할 수 있습니다.

```
oThread = oAssistant->CreateThread();
```

3단계: Thread에 메시지를 추가하고 스트리밍을 사용하여 Run

사용자나 애플리케이션이 생성하는 메시지의 내용은 Message 객체로 Thread에 추가됩니다. Messages에는 텍스트와 파일이 모두 포함될 수 있습니다. Threads에 추가할 수 있는 Messages 수에는 제한이 없습니다 — 모델의 컨텍스트 창에 맞지 않는 컨텍스트는 지능적으로 잘립니다.

모든 사용자 Message가 Thread에 추가되면, 어떤 Assistant로든 Thread를 Run할 수 있습니다. Run을 생성하면 Assistant와 연결된 모델과 도구를 사용하여 응답을 생성합니다. 응답은 Server-Sent stream을 사용하여 전송됩니다.

```
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);
}
}

```

4단계: 응답 처리

서버가 전송한 스트림 메시지를 읽으려면 OnStreamMessageDelta 이벤트를 사용하십시오.

```

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

Chat Completions API와 유사하게, Assistants API는 함수 호출을 지원합니다. 함수 호출을 사용하면 Assistants API에 함수를 설명하고 호출해야 하는 함수를 인수와 함께 지능적으로 반환하도록 할 수 있습니다.

이 예제에서는 날씨 도우미를 만들고 도우미가 호출할 수 있는 도구로 두 개의 함수 **get_current_temperature**와 **get_rain_probability**를 정의합니다. 병렬 함수 호출을 사용하는 예제에서는 도우미에게 오늘 샌프란시스코의 날씨와 비가 올 확률을 물어봅니다. 또한 스트리밍을 사용하여 도우미의 응답을 출력하는 방법도 보여줍니다.

1단계 함수 정의

assistant를 생성할 때, 먼저 assistant의 tools 매개변수 아래에 함수를 정의합니다.

```
TsgcAIOpenAIAssistant *Assistant = new TsgcAIOpenAIAssistant(nullptr);
Assistant->OpenAIOptions->ApiKey = "sk-askdjfalskdjfl23kjkjasdefasdj";
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;
```

2단계: Thread를 생성하고 Messages 추가

사용자가 대화를 시작할 때 Thread를 생성하고 사용자가 질문할 때 Thread에 Messages를 추가하십시오.

```
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
    }
}
```

3단계: OnFunctionCall Event 처리

구성 요소가 함수 매개변수 값이 필요함을 감지하면 **OnFunctionCall** 이벤트가 호출됩니다. 요청 세부 정보를 알려면 **Request._Function** 매개변수를 사용하고 응답을 보내려면 **Response.Output**을 사용하십시오.

```
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

음성 명령으로 OpenAI API를 사용하려면 다음 단계가 필요합니다:

1. 마이크 오디오를 캡처해야 하므로, OpenAI로 전송될 텍스트를 얻으려면 **speech-to-text** 시스템이 필요합니다.
 1. 마이크 오디오 캡처는 [TsgcAudioRecorderMCI](#) 구성 요소를 사용하여 수행됩니다.
 2. 오디오를 캡처하면, OpenAI Whisper API로 전송되어 오디오 파일을 텍스트로 변환합니다.
2. 음성-텍스트 변환 결과를 얻으면, ChatCompletion API를 사용하여 텍스트를 OpenAI로 보냅니다.
3. 그런 다음 OpenAI의 응답은 다음 구성 요소 중 하나를 사용하여 음성으로 변환되어야 합니다:
 1. [TsgcTextToSpeechSystem](#): (현재 Windows 전용) 운영 체제의 Windows text-to-speech를 사용합니다.
 2. [TsgcTextToSpeechGoogle](#): OpenAI의 응답을 Google Cloud 서버로 전송하면 mp3 파일이 반환되어 [TsgcAudioPlayerMCI](#)에 의해 재생됩니다.
 3. [TsgcTextToSpeechAmazon](#): OpenAI의 응답을 Amazon AWS 서버로 전송하면 mp3 파일이 반환되어 [TsgcAudioPlayerMCI](#)에 의해 재생됩니다.

구성 요소

다음 구성 요소는 마이크에서 오디오를 캡처하고, 오디오 파일을 재생하고, 텍스트를 음성으로 변환하는 데 사용됩니다.

- [TsgcAudioRecorderMCI](#): (Windows 전용) 이 구성 요소는 마이크에 액세스하고 음성을 wave 파일로 변환할 수 있게 합니다.
- [TsgcAudioPlayerMCI](#): (Windows 전용) 이 구성 요소를 사용하면 mp3 파일을 재생할 수 있습니다.
- [TsgcTextToSpeechSystem](#): (Windows 전용) 외부 mp3 파일을 사용할 필요 없이 텍스트를 음성으로 변환합니다.
- [TsgcTextToSpeechGoogle](#): 사용 가능한 Google Cloud 음성 중 하나를 사용하여 텍스트를 음성으로 변환합니다.
- [TsgcTextToSpeechAmazon](#): 사용 가능한 Amazon AWS 음성 중 하나를 사용하여 텍스트를 음성으로 변환합니다.

TsgcAudioRecorderMCI

이 구성 요소는 마이크 오디오를 캡처하여 wave 파일에 저장하는 데 사용됩니다.

소개

이 구성 요소는 마이크 오디오를 캡처하여 wave 파일에 저장하는 데 사용됩니다. 현재 Windows만 지원됩니다.

참조

- 속성 — 구성 요소에서 사용 가능한 3개의 속성.
- 메서드 — 2개의 public 메서드.
- 이벤트 — 처리할 수 있는 이벤트 0개.

TsgcAudioPlayerMCI

이 구성 요소는 *text-to-speech* 공급자로부터 받은 mp3 파일을 재생하는 데 사용됩니다.

소개

이 구성 요소는 *text-to-speech* 제공자로부터 받은 mp3 파일을 재생하는 데 사용됩니다. 현재 Windows만 지원됩니다.

참조

- 속성 — 구성 요소에서 사용 가능한 속성 1개입니다.
- 메서드 — 1개의 public 메서드입니다.
- 이벤트 — 처리할 수 있는 0개의 이벤트.

TsgcTextToSpeechSystem

이것은 운영 체제에서 제공하는 기본 Text-To-Speech이며, 현재 Windows만 지원됩니다.

소개

이것은 운영 체제에서 제공하는 기본 Text-To-Speech이며, 현재 Windows만 지원됩니다.

참조

- 속성 — 구성 요소에서 사용 가능한 1개의 속성.
- 메서드 — 1개의 public 메서드.
- 이벤트 — 처리할 수 있는 0개의 이벤트.

TsgcTextToSpeechGoogle

*Text-To-Speech*는 Google Cloud에서 제공하는 API로, 텍스트를 mp3 파일로 변환할 수 있습니다.

소개

Text-To-Speech는 Google Cloud에서 제공하는 API로, 텍스트를 mp3 파일로 변환할 수 있습니다. Google Cloud 계정과 Text-To-Speech 계정 설정이 필요합니다.

Text-To-Speech 계정이 구성되면, JSON 설정 파일을 다운로드하여 `GoogleOptions.Settings` 속성에 설정해야 합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 3개의 속성.
- 메서드 — 1개의 공용 메서드입니다.
- 이벤트 — 처리할 수 있는 0개의 이벤트.

TsgcTextToSpeechAmazon

*Text-To-Speech*는 텍스트를 mp3 파일로 변환할 수 있게 하는 Amazon AWS가 제공하는 API입니다.

소개

Text-To-Speech는 Amazon AWS가 제공하는 API로, 텍스트를 mp3 파일로 변환할 수 있게 합니다. Amazon AWS 계정과 Polly API 설정이 필요합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 3개의 속성.
- 메서드 — 1개의 public 메서드입니다.
- 이벤트 — 처리할 수 있는 이벤트 0개.

TsgcAIChat

TsgcAIChat은 모든 AI 제공자와 함께 작동하는 단일 구성 요소입니다.

소개

TsgcAIChat은 모든 AI provider와 작동하는 단일 구성 요소입니다. 한 속성을 변경하여 provider를 전환합니다. 이 구성 요소는 내장 대화 기록 관리를 포함하므로, 메시지를 수동으로 추적하지 않고도 채팅 애플리케이션을 쉽게 구축할 수 있습니다.

지원되는 공급자

- OpenAI
- Anthropic
- Gemini
- DeepSeek
- Ollama
- Grok
- Mistral

참조

- 속성 — 구성 요소에서 사용 가능한 5개의 속성.
- 메서드 — 4개의 public 메서드.
- 이벤트 — 처리할 수 있는 3개의 이벤트.

TsgcAIOpenAIChatBot

마이크 오디오를 캡처해야 하므로 OpenAI로 전송될 텍스트를 얻기 위해 speech-to-text 시스템이 필요합니다.

소개

음성 명령으로 ChatBot을 구축하려면 다음 단계가 필요합니다:

1. 마이크 오디오를 캡처해야 하므로, OpenAI로 전송될 텍스트를 가져오기 위해 speech-to-text 시스템이 필요합니다.
 1. 마이크 오디오 캡처는 [TsgcAudioRecorderMCI](#) 구성 요소를 사용하여 수행됩니다.
 2. 오디오를 캡처한 후, 오디오 파일을 텍스트로 변환하기 위해 OpenAI Whisper API로 전송됩니다.
2. 음성-텍스트 변환 결과를 얻으면, ChatCompletion API를 사용하여 텍스트를 OpenAI로 보냅니다.
3. 그런 다음 OpenAI의 응답은 다음 구성 요소 중 하나를 사용하여 음성으로 변환되어야 합니다:
 1. [TsgcTextToSpeechSystem](#): (현재 Windows 전용) 운영 체제의 Windows text-to-speech를 사용합니다.
 2. [TsgcTextToSpeechGoogle](#): OpenAI의 응답을 Google Cloud 서버로 전송하면 mp3 파일이 반환되어 [TsgcAudioPlayerMCI](#)에 의해 재생됩니다.
 3. [TsgcTextToSpeechAmazon](#): OpenAI의 응답을 Amazon AWS 서버로 전송하면 mp3 파일이 반환되어 [TsgcAudioPlayerMCI](#)에 의해 재생됩니다.

참조

- 속성 — 구성 요소에서 사용 가능한 6개의 속성입니다.
- 메서드 — 4개의 public 메서드.
- Events — 처리할 수 있는 4개의 이벤트.

TsgcAIOpenAITranslator

마이크 오디오를 캡처해야 하므로 OpenAI로 전송될 텍스트를 얻기 위해 speech-to-text 시스템이 필요합니다.

소개

음성 명령으로 Translator를 빌드하려면 다음 단계가 필요합니다.

1. 마이크 오디오를 캡처해야 하므로, OpenAI로 전송될 텍스트를 가져오기 위해 speech-to-text 시스템이 필요합니다.
 1. 마이크 오디오 캡처는 [TsgcAudioRecorderMCI](#) 구성 요소를 사용하여 수행됩니다.
 2. 오디오를 캡처한 후, 오디오 파일을 텍스트로 변환하기 위해 OpenAI Whisper API로 전송됩니다.
2. 음성-텍스트 변환 결과를 얻으면, ChatCompletion API를 사용하여 텍스트를 OpenAI로 보냅니다.
3. 그런 다음 OpenAI의 응답은 다음 구성 요소 중 하나를 사용하여 음성으로 변환되어야 합니다:
 1. [TsgcTextToSpeechSystem](#): (현재 Windows 전용) 운영 체제의 Windows text-to-speech를 사용합니다.
 2. [TsgcTextToSpeechGoogle](#): OpenAI의 응답을 Google Cloud 서버로 전송하면 mp3 파일이 반환되어 [TsgcAudioPlayerMCI](#)에 의해 재생됩니다.
 3. [TsgcTextToSpeechAmazon](#): OpenAI의 응답을 Amazon AWS 서버로 전송하면 mp3 파일이 반환되어 [TsgcAudioPlayerMCI](#)에 의해 재생됩니다.

참조

- 속성 — 구성 요소에서 사용할 수 있는 5개의 속성.
- 메서드 — 2개의 공개 메서드.
- 이벤트 — 처리할 수 있는 3개의 이벤트.

TsgcAIOpenAIEmbeddings

Embeddings는 단어, 구문, 또는 이미지나 오디오와 같은 다른 유형의 데이터를 숫자 형태로 표현하는 방법입니다.

소개

Embeddings는 단어, 구문, 또는 이미지나 오디오 같은 다른 유형의 데이터를 숫자 형식으로 표현하는 방법입니다. 컴퓨터가 더 잘 이해하고 작업할 수 있도록 단어나 데이터를 숫자로 바꾸는 것과 같습니다.

"dog", "cat", "bird"와 같은 여러 단어가 있다고 상상해 보십시오. 이러한 단어에는 의미가 있습니다. 그렇죠? 임베딩은 각 단어에 의미와 다른 단어와의 관계를 포착하는 고유한 숫자 집합(벡터)을 할당합니다.

예를 들어, "dog"이라는 단어는 [0.5, 0.2, -0.7]로, "cat"은 [0.8, -0.3, 0.1]로, "bird"는 [0.3, 0.9, 0.4]로 표현될 수 있습니다. 벡터의 숫자는 각 단어가 동물과 관련이 있는지 또는 서로 얼마나 유사한지와 같은 각 단어의 특성에 대한 정보를 전달합니다.

놀라운 점은 임베딩이 대량의 데이터로부터 학습될 수 있다는 것입니다. 따라서 단어 간의 유사점과 차이점을 자동으로 파악할 수 있습니다. 이러한 수치적 표현은 AI 알고리즘이 언어를 이해하고 복잡한 패턴을 파악하는 데 도움이 되며, 이는 언어 번역, 감정 분석, 추천 시스템과 같은 다양한 애플리케이션에서 중요합니다. 또한 AI 모델이 정보를 처리하고 더 정확한 결과를 제공하는 것을 더 쉽고 빠르게 만듭니다!

참조

- 속성 — 구성 요소에서 사용 가능한 4개의 속성.
- 메서드 — 2개의 public 메서드.
- 이벤트 — 처리할 수 있는 5개의 이벤트입니다.

TsgcAIDatabaseVectorFile

구성 요소는 데이터베이스 벡터와 프롬프트를 두 개의 텍스트 파일에 저장합니다.

소개

구성 요소는 데이터베이스 벡터와 프롬프트를 두 개의 텍스트 파일에 저장합니다. 이 구성 요소는 벡터 수가 많을 때 최적화되지 않으므로 프로덕션이 아닌 테스트 목적으로만 사용해야 합니다.

구성

- **VectorFileOptions:**
 - **InputFilename:** 입력 데이터를 저장하는 데 사용되는 파일의 이름입니다.
 - **VectorFilename:** 벡터를 저장하는 데 사용되는 파일의 이름입니다.

참조

- 속성 — 구성 요소에서 사용 가능한 2개의 속성.
- 메서드 — 4개의 public 메서드.
- 이벤트 — 처리할 수 있는 0개의 이벤트.

TsgcAIDatabaseVectorPinecone

TsgcAIDatabaseVectorPinecone 구성 요소.

소개

구성 요소는 인덱스와 벡터를 생성 / 업데이트 / 삭제할 수 있는 REST [Pinecone API client](#)를 기반으로 합니다.

구성

- **PineconeOptions:**
 - **ApiKey:** pinecone 계정에서 얻을 수 있는, pinecone이 제공하는 API Key를 여기에 구성하십시오.
 - **Environment:** 기본적으로 무료 계정 "us-west4-gcp-free"입니다.
 - **LogOptions:** HTTP 요청을 텍스트 파일에 저장하려면 여기에 구성하십시오.
- **PineconeIndexOptions:**
 - **IndexName:** 데이터를 저장하거나 쿼리하는 데 사용되는 인덱스의 이름입니다.
 - **ProjectId:** 프로젝트의 id입니다.

참조

- 속성 — 구성 요소에서 사용할 수 있는 3개의 속성.
- 메서드 — 4개의 public 메서드.
- 이벤트 — 처리할 수 있는 이벤트 0개입니다.

Embeddings | Create Vectors

임베딩을 사용하려면, 먼저 데이터를 벡터로 변환해야 합니다.

예제

PDF 파일이 있는 경우 먼저 PDF 파일을 텍스트로 변환한 다음 **CreateEmbeddingsFromFile** 메서드를 사용하여 벡터 데이터를 가져오십시오. OpenAI Embeddings 크기 제한으로 인해 파일이 너무 큰 경우 데이터가 자동으로 청크로 분할되므로, 하나의 파일에서 하나 또는 여러 개의 벡터를 얻을 수 있습니다.

아래 코드 샘플을 참조하십시오.

```
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

모든 데이터를 벡터로 변환하고 나면, 자체 모델을 구축하기 시작할 수 있습니다. 그 이면의 아이디어는 매우 간단합니다: 봇에게 질문할 때마다 먼저 질문을 벡터로 변환한 다음, 데이터베이스에서 어떤 벡터가 질문과 가장 유사한지 검색하고, 마지막으로 가장 유사한 데이터를 사용하여 이를 컨텍스트로 추가합니다.

```
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)

Pinecone은 쉽고 강력한 방식으로 벡터 데이터를 업로드, 쿼리, 삭제할 수 있는 벡터 데이터베이스입니다.

Pinecone에는 서드파티가 Pinecone을 자체 애플리케이션에 통합할 수 있도록 하는 공개 API가 있습니다. 구성 요소 TsgcHTTP_API_Pinecone는 Pinecone API 위의 래퍼입니다.

구성

시작하기 전에 Pinecone 웹사이트에 등록하고 API key를 요청해야 합니다. 이 API key는 API 요청을 보내는 데 사용되며 TsgcHTTP_API_Pinecone 구성 요소의 **PineconeOptions.ApiKey** 속성에 설정해야 합니다.

Index Operations

다음 메서드가 지원됩니다:

Method	매개변수	설명
IndexesList		이 작업은 Pinecone 인덱스 목록을 반환합니다.
IndexCreate	TsgcHTTTPineconeIndexCreate	이 작업은 Pinecone 인덱스를 생성합니다. 이를 사용하여 유사성 측정, 인덱스에 저장될 벡터의 차원, 사용할 replica 수 등을 지정할 수 있습니다.
IndexDescribe	인덱스 이름	인덱스에 대한 설명을 가져옵니다.
IndexDelete	인덱스 이름	이 작업은 기존 인덱스를 삭제합니다.
IndexConfigure	Index Name, Replicas, PodType	이 작업은 인덱스에 대한 포드 유형과 복제본 수를 지정합니다.

Collection Operations

다음 메서드가 지원됩니다:

Method	매개변수	설명
CollectionsList		이 작업은 Pinecone collection 목록을 반환합니다.
CollectionCreate	Collection Name, Source	이 작업은 Pinecone collection을 생성합니다.
CollectionDescribe	Collection Name	컬렉션에 대한 설명을 가져옵니다.
CollectionDelete	Collection Name	이 작업은 기존 컬렉션을 삭제합니다.

벡터 작업

다음 메서드가 지원됩니다:

Method	매개변수	설명
VectorsDescribeIndexStats	Index Name, Project Id, Filter	DescribeIndexStats 작업은 namespace당 벡터 수 및 차원 수를 포함하여 인덱스의 콘텐츠에 대한 통계를 반환합니다.
VectorsQuery	Index Name, Project Id, Params	Query 작업은 query 벡터를 사용하여 namespace를 검색합니다. namespace에서 가장 유사한 항목의 id를 유사도 점수와 함께 검색합니다.
VectorsDelete	Index Name, Project Id, Params	Delete 작업은 단일 네임스페이스에서 id로 벡터를 삭제합니다. 단일 네임스페이스에서 id로 항목을 삭제할 수 있습니다.
VectorsFetch	Index Name, Project Id, Ids	Fetch 작업은 단일 네임스페이스에서 ID로 벡터를 조회하고 반환합니다. 반환된 벡터에는 벡터 데이터 및/또는 메타데이터가 포함됩니다.
VectorsUpdate	Index Name, Project Id, Params	Update 작업은 네임스페이스의 벡터를 업데이트합니다. 값이 포함되면 이전 값을 덮어씁니다. set_metadata가 포함되면 거기에 지정된 필드의 값이 추가되거나 이전 값을 덮어씁니다.
VectorsUpsert	Index Name, Project Id, Params	Upsert 작업은 벡터를 네임스페이스에 씁니다. 기존 벡터 id에 대해 새 값이 upsert되면, 이전 값을 덮어씁니다.

UPSERT 예제

아래에서 id = "id1"인 단일 벡터를 UPSERT하는 예제를 확인하십시오.

```
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();
    }
}
```

Example QUERY

단일 벡터를 QUERY하는 예제를 아래에서 확인하십시오.

```
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

Anthropic은 신뢰할 수 있고, 해석 가능하며, 조정 가능한 AI 시스템을 구축하는 AI 안전 회사입니다. 대표 모델 제품군은 Claude이며, 사려 깊은 대화, 콘텐츠 생성, 복잡한 추론, 코딩 등에서 뛰어납니다. sgcWebSockets 라이브러리는 Anthropic Claude API와 상호 작용하기 위한 Delphi 구성 요소 TsgcHTTP_API_Anthropic을 제공합니다.

Anthropic API

Anthropic API는 AI 기반 애플리케이션 구축을 위한 Claude 모델에 대한 액세스를 제공합니다. API는 텍스트 생성, 비전(이미지 이해), 도구 사용(함수 호출), 확장된 사고, 문서/PDF 처리, 프롬프트 캐싱, 인용, 웹 검색, 스트리밍, 토큰 계산 및 메시지 배치를 지원합니다.

Features

- 메시지
 - [Anthropic Messages 예제](#)
- Vision
 - [Anthropic Vision Examples](#)
- Tool Use
 - [Anthropic Tool Use 예제](#)
- Models
 - [Anthropic Models 예제](#)
- 배치
 - [Anthropic Batches 예제](#)
- Extended Thinking
 - [Anthropic Extended Thinking 예제](#)
- Documents
 - [Anthropic Documents 예제](#)
- Prompt Caching
 - [Anthropic Prompt Caching 예제](#)
- Citations
 - [Anthropic Citations 예제](#)
- 웹 검색
 - [Anthropic 웹 검색 예제](#)
- Structured Outputs
 - [Anthropic Structured Outputs Examples](#)
- Files
 - [Anthropic Files API 예제](#)
- MCP Connector
 - [Anthropic MCP Connector 예제](#)

구성

Anthropic API는 인증에 API 키를 사용합니다. Anthropic Console의 [API Keys](#) 페이지를 방문하여 요청에 사용할 API 키를 검색하십시오.

API 키는 비밀임을 기억하십시오! 다른 사람과 공유하거나 클라이언트 측 코드에 노출하지 마십시오.

이 **API Key**는 구성 요소의 **AnthropicOptions.ApiKey** 속성에 구성되어야 합니다. **AnthropicOptions.AnthropicVersion** 속성은 API 버전을 지정합니다(기본값: 2023-06-01).

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "YOUR_API_KEY";
```

Properties

AnthropicOptions

- **ApiKey**: Anthropic API로 인증하기 위한 API 키입니다.

- **AnthropicVersion:** API 버전 문자열입니다(기본값: 2023-06-01).
- **BetaHeaders:** 사전 릴리스 API 기능을 활성화하는 선택적 베타 기능 헤더입니다(예: files-api-2025-04-14, mcp-client-2025-11-20).
- **HttpOptions:** HTTP 연결 설정.
 - **ReadTimeout:** HTTP 응답을 읽기 위한 타임아웃(밀리초). 기본값은 0(타임아웃 없음)입니다.
- **LogOptions:** 로깅 구성입니다.
 - **Enabled:** True인 경우 HTTP 요청 및 응답이 파일에 기록됩니다.
 - **FileName:** 로그 출력이 기록되는 파일 경로입니다.
- **RetryOptions:** 실패한 API 요청에 대한 자동 재시도 구성입니다.
 - **Enabled:** True인 경우 실패한 요청이 자동으로 재시도됩니다.
 - **Retries:** 최대 재시도 횟수.
 - **Wait:** 재시도 시도 간 대기 시간(밀리초).

메시지

텍스트 및/또는 이미지 콘텐츠가 포함된 입력 메시지의 구조화된 목록을 보내면, 모델이 대화의 다음 메시지를 생성합니다.

- **CreateMessage:** 지정된 모델과 매개변수로 메시지를 생성합니다.
 - **Model:** 사용할 모델입니다(예: claude-sonnet-4-20250514).
 - **Message:** 사용자 메시지 콘텐츠입니다.
 - **MaxTokens:** 생성할 최대 토큰 수입니다(필수, 기본값 4096).
- **CreateMessageWithSystem:** 시스템 프롬프트로 메시지를 생성합니다.
 - **System:** assistant의 동작을 설정하는 시스템 프롬프트입니다.
- **CreateMessageStream:** 스트리밍(SSE)이 활성화된 메시지를 생성합니다. 이벤트는 OnHTTPAPISSE 이벤트 처리기를 통해 전달됩니다.
- **CreateMessageJSON:** 제공된 JSON Schema를 준수하는 구조화된 JSON을 반환하는 메시지를 생성합니다.
 - **Schema:** 출력 형식을 정의하는 JSON Schema 문자열.
- **CreateMessageWithThinking:** extended thinking이 활성화된 메시지를 생성합니다.
 - **BudgetTokens:** thinking에 대한 최대 토큰 예산(최소 1024).
- **CreateDocumentMessage:** base64 문서(PDF 또는 텍스트)가 있는 메시지를 생성합니다.
 - **DocumentBase64:** base64로 인코딩된 문서 데이터.
 - **MediaType:** MIME 유형(예: application/pdf).

Vision

Claude는 메시지 내에서 base64로 인코딩된 콘텐츠 블록으로 전달된 이미지를 이해할 수 있습니다.

- **CreateVisionMessage:** 텍스트 프롬프트와 함께 이미지를 보냅니다.
 - **ImageBase64:** base64 인코딩된 이미지 데이터.
 - **MediaType:** MIME 유형(image/jpeg, image/png, image/gif, image/webp).
 - **Prompt:** 이미지에 동반되는 텍스트 프롬프트.

도구 사용

Claude는 도구(function calling)를 사용하여 외부 시스템과 상호작용할 수 있습니다. 도구를 이름, 설명 및 입력 스키마와 함께 정의하면, Claude는 도구를 호출하려고 할 때 tool_use content 블록을 생성합니다.

모델

사용 가능한 Claude 모델을 나열하고 설명합니다.

- **GetModels:** 사용 가능한 모든 모델을 나열합니다.
- **GetModel:** 특정 모델에 대한 정보를 검색합니다.
 - **ModelId:** 검색할 모델의 ID입니다.

Extended Thinking

확장 사고(Extended thinking)는 Claude가 응답하기 전에 단계별로 추론할 수 있게 하여 수학, 코딩 및 분석과 같은 복잡한 작업의 품질을 향상시킵니다.

- **ThinkingType:** thinking를 활성화하려면 'enabled'로, 끄려면 'disabled'로 설정하십시오.
- **ThinkingBudgetTokens:** thinking을 위한 토큰 예산(최소 1024, MaxTokens보다 작아야 함)입니다.
- **CreateMessageWithAdaptiveThinking:** 적응형 thinking으로 메시지를 생성하여, Claude가 thinking 깊이를 자동으로 결정하게 합니다.

Documents

Claude는 콘텐츠 블록으로 전송된 PDF 문서와 텍스트 파일을 처리할 수 있습니다. base64, 텍스트 및 URL 소스 유형을 지원합니다.

- **CreateDocumentMessage:** 텍스트 프롬프트와 함께 문서를 보냅니다.
 - **DocumentBase64:** base64 인코딩된 문서 데이터.
 - **MediaType:** MIME 유형(application/pdf, text/plain).

Prompt Caching

API 호출 간에 자주 사용되는 컨텍스트(system 프롬프트, 콘텐츠 블록, 도구 정의)를 캐시하여 캐시 읽기에서 비용을 최대 90% 절감합니다.

- **SystemCacheControl:** system prompt를 캐시하려면 요청에서 True로 설정하십시오.
- **CacheControl:** 콘텐츠 블록 또는 도구 정의에서 'ephemeral'로 설정하십시오.
- **BetaHeaders:** AnthropicOptions의 선택적 베타 기능 헤더입니다.

인용

citation이 활성화된 상태로 문서가 전송되면 Claude는 응답에 소스 참조를 포함합니다. Citation 유형에는 char_location(텍스트), page_location(PDF), content_block_location(사용자 정의 콘텐츠) 및 web_search_result_location(웹 검색)이 포함됩니다.

Web Search

Claude는 내장 web_search 도구를 사용하여 실시간 정보를 위해 웹을 검색할 수 있습니다. 다른 내장 도구로는 code_execution과 computer use가 있습니다.

- **CreateMessageWithWebSearch:** 웹 검색이 활성화된 메시지를 생성합니다.
 - **Model:** 사용할 모델.
 - **Message:** 사용자 쿼리.

Token Counting

메시지를 보내기 전에 메시지의 토큰 수를 계산합니다.

- **CountTokens:** 메시지에 대한 입력 토큰 수를 계산합니다.
 - **Model:** 토큰 계산에 사용할 모델.
 - **Message:** 토큰을 계산할 메시지 콘텐츠.

Message Batches

Message Batches API를 사용하면 대량의 메시지를 비동기적으로 처리할 수 있습니다.

- **ListBatches:** 모든 메시지 배치를 나열합니다.
- **GetBatch:** ID로 특정 배치를 검색합니다.
- **CancelBatch:** 아직 처리 중인 배치를 취소합니다.
- **GetBatchResults:** 완료된 배치의 결과를 검색합니다.

구조화된 출력(Structured Outputs)

제공된 JSON Schema를 준수하는 유효한 JSON을 반환하도록 Claude를 강제합니다. 출력 품질 대 비용을 제어하려면 Effort 매개변수와 결합하십시오.

- **CreateMessageJSON:** JSON 스키마 출력으로 메시지를 생성합니다.
 - **Schema:** 출력 형식을 정의하는 JSON Schema 문자열입니다.
- **OutputFormatSchema:** (Request 속성) 구조화된 출력을 위한 JSON Schema입니다.
- **Effort:** (Request 속성) 품질 대 비용을 제어합니다: 'low', 'medium', 'high', 'max'.
- **Strict:** (Tool 속성) True인 경우 도구 입력이 input_schema와 일치하도록 보장됩니다.

Files API

파일을 업로드, 나열, 검색, 다운로드 및 삭제합니다. 업로드된 파일은 파일 소스 유형의 문서 콘텐츠 블록을 사용하여 메시지에서 참조할 수 있습니다. 베타 헤더 files-api-2025-04-14가 필요합니다.

- **UploadFile:** 로컬 파일을 업로드합니다. 파일 메타데이터를 반환합니다.
- **ListFiles:** 페이지네이션과 함께 업로드된 파일을 나열합니다.
- **GetFile:** 특정 파일의 메타데이터를 검색합니다.
- **DownloadFile:** 파일 콘텐츠를 다운로드합니다.
- **DeleteFile:** 파일을 영구적으로 삭제합니다.

Request Parameters

Messages API에서 사용 가능한 추가 요청 매개변수입니다.

- **ServiceTier:** 우선순위 티어 사용을 제어합니다: 'auto'(기본값) 또는 'standard_only'.
- **InferenceGeo:** 추론 지역을 제어합니다: 'global'(기본값) 또는 'us'.
- **Container:** 요청 간에 코드 실행 환경을 재사용하기 위한 Container ID입니다.
- **IsError:** (콘텐츠 블록 속성) 도구 실행 실패를 나타내려면 tool_result 블록에서 True로 설정하십시오.
- **CacheTTL:** 확장된 캐시 time-to-live: '5m'(기본값) 또는 '1h'.
- **ThinkingType:** Claude가 thinking 깊이를 자동으로 결정하도록 하려면 'adaptive'로 설정하십시오.

MCP Connector

Claude를 외부 MCP(Model Context Protocol) 서버에 연결하여 서드파티 도구에 액세스합니다. 베타 헤더 `mcp-client-2025-11-20`이 필요합니다.

- **CreateMessageWithMCP:** MCP 서버 연결로 메시지를 생성합니다.
 - **MCPServerUrl:** MCP 서버의 HTTPS URL입니다.
 - **MCPServerName:** 서버의 고유한 이름입니다.
- **MCPServers:** (Request 속성) ServerType, Url, Name, AuthorizationToken을 가진 MCP 서버 정의 배열입니다.
- **MCPServerName:** (Tool 속성) ToolType이 'mcp_toolset'일 때 MCP 서버를 참조합니다.

TsgcHTTP_API_Anthropic

Claude Messages, Files, Batches 및 토큰 계산을 위한 Anthropic REST API 클라이언트.

소개

Claude Messages, Files, Batches 및 토큰 계산을 위한 Anthropic REST API 클라이언트.

소개

Claude Messages, Files, Batches 및 토큰 계산을 위한 Anthropic REST API 클라이언트.

AnthropicOptions 속성을 API 키와 선택적인 HTTP/재시도/로그 설정으로 구성한 다음, 요청 객체와 함께 `CreateMessage/`
`CreateChatCompletion`과 같은 형식화된 메서드를 호출하여 형식화된 응답 클래스를 받으십시오.
공급자에 대한 배경 정보는 [Anthropic](#) 가이드를 참조하십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 속성 5개입니다.
- 메서드 — 7개의 public 메서드.
- Events — 처리할 수 있는 2개의 이벤트.

Anthropic | Messages

텍스트 콘텐츠가 포함된 입력 메시지의 구조화된 목록을 보내면 모델이 대화에서 다음 메시지를 생성합니다.

간단한 예제

Claude에 Hello 메시지를 보냅니다.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_CreateMessage("claude-sonnet-4-20250514", "Hello!"));
```

System Prompt 예제

Claude의 동작을 제어하기 위해 system prompt가 있는 메시지를 보냅니다.

```
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?"));
```

고급 예제

메시지 매개변수에 대한 완전한 제어를 위해 형식화된 request/response 클래스를 사용하십시오.

```
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;
}
```

스트리밍 예제

Server-Sent Events (SSE)를 사용하여 응답을 실시간으로 스트리밍합니다. 스트리밍 이벤트를 수신하려면 OnHTTPAPISSE 이벤트 핸들러를 할당하십시오.

```
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

Claude는 이미지를 이해하고 분석할 수 있습니다. 콘텐츠 블록 내에서 이미지를 base64 인코딩 데이터로 보낼 수 있습니다.

지원되는 이미지 형식

- **image/jpeg** - JPEG 이미지
- **image/png** - PNG 이미지
- **image/gif** - GIF 이미지
- **image/webp** - WebP 이미지

간단한 예제

Claude에게 이미지를 설명하도록 요청하는 프롬프트와 함께 이미지를 보냅니다.

```
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"));
```

고급 예제

이미지 메시지를 더 세밀하게 제어하려면 콘텐츠 블록을 사용하십시오.

```
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();
}
```

Anthropic | Tool Use

Claude는 도구 사용(function calling)을 지원하여, Claude가 대화 중에 호출할 수 있는 도구를 정의할 수 있도록 합니다. Claude가 도구를 사용하기로 결정하면 **tool_use** 콘텐츠 블록을 반환합니다. 그런 다음 도구를 실행하고 결과를 **tool_result** 콘텐츠 블록으로 다시 보냅니다.

Tool Use Flow

1. 이름, 설명 및 input_schema(JSON Schema)로 도구를 정의합니다.
2. 정의된 tools와 함께 메시지를 보냅니다.
3. Claude는 **tool_use** 콘텐츠 블록(stop_reason = 'tool_use')으로 응답합니다.
4. 제공된 입력으로 도구를 실행합니다.
5. 출력을 포함하는 **tool_result** content 블록과 함께 새 메시지를 보냅니다.
6. Claude가 최종 답변으로 응답합니다.

예제

weather 도구를 정의하고 tool use 루프를 처리합니다.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";

// Step 1: Create request with tools
TsgcAnthropicClass_Request_Messages *oRequest = new TsgcAnthropicClass_Request_Messages(NULL);
try {
    oRequest->Model = "claude-sonnet-4-20250514";
    oRequest->MaxTokens = 4096;

    // Define user message
    TsgcAnthropicClass_Request_Message *oMessage = new TsgcAnthropicClass_Request_Message(NULL);
    oMessage->Role = "user";
    oMessage->Content = "What is the weather in San Francisco?";
    SetLength(oMessages, 1);
    oMessages[0] = oMessage;
    oRequest->Messages = oMessages;

    // Define tool
    TsgcAnthropicClass_Request_Tool *oTool = new TsgcAnthropicClass_Request_Tool(NULL);
    oTool->Name = "get_weather";
    oTool->Description = "Get the current weather in a given location";
    oTool->InputSchema =
        "{\n\"type\": \"object\", \"properties\": {\n\"location\": {\n\"type\": \"string\", \"\n\"description\": \"The city and state\"}}, \"required\": [\n\"location\"]}";
    SetLength(oTools, 1);
    oTools[0] = oTool;
    oRequest->Tools = oTools;

    // Step 2: Send request
    TsgcAnthropicClass_Response_Messages *oResponse = Anthropic->CreateMessage(oRequest);
    try {
        // Step 3: Check if Claude wants to use a tool
        if (oResponse->StopReason == "tool_use") {
            // Find the tool_use content block
            for (int i = 0; i < Length(oResponse->Content); i++) {
                if (oResponse->Content[i]->ContentType == "tool_use") {
                    vToolUseId = oResponse->Content[i]->Id;
                    vToolName = oResponse->Content[i]->Name;
                    vToolInput = oResponse->Content[i]->Input;
                    // Step 4: Execute your tool (get_weather) and get result
                    vToolResult = "72 degrees and sunny";
                    break;
                }
            }

            // Step 5: Send tool result back
            // Build new message array with assistant response + tool result
            // ... (continue the conversation with tool_result content block)
        }
    } finally {
        oResponse->Free();
    }
} finally {
    oMessage->Free();
}
```

```
oTool->Free();  
oRequest->Free();  
}
```

Anthropic | Models

사용 가능한 Claude 모델에 대한 정보를 나열하고 검색합니다.

List Models

사용 가능한 모든 Claude 모델을 나열합니다.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_GetModels());
```

Get Model

특정 모델에 대한 정보를 검색합니다.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_GetModel("claude-sonnet-4-20250514"));
```

Typed Response

모델 데이터에 대한 구조화된 액세스를 위해 typed 응답 클래스를 사용하십시오.

```
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

Message Batches API를 사용하면 대량의 메시지를 비동기적으로 처리할 수 있습니다. 이는 대량 콘텐츠 생성, 데이터 분석 또는 배치 처리 워크플로와 같이 즉각적인 응답이 필요하지 않은 작업에 이상적입니다.

List Batches

조직의 모든 메시지 batch를 나열합니다.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_ListBatches());
```

Get Batch

특정 배치에 대한 정보를 검색합니다.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_GetBatch("batch_id_here"));
```

Cancel Batch

여전히 처리 중인 batch를 취소합니다.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_CancelBatch("batch_id_here"));
```

Get Batch Results

완료된 배치의 결과를 검색합니다.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_GetBatchResults("batch_id_here"));
```

Typed Response

배치 데이터에 대한 구조화된 액세스를 위해 형식화된 응답 클래스를 사용하십시오.

```
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

Extended thinking은 Claude에게 응답을 제공하기 전에 복잡한 문제를 단계별로 사고할 수 있는 능력을 제공합니다. 활성화되면 Claude는 수학, 코딩, 분석 및 기타 복잡한 작업에 대한 응답 품질을 향상시키는 내부 추론(thinking block)을 생성합니다.

간단한 예제

편의 메서드를 사용하여 확장된 사고를 활성화한 상태로 메시지를 보냅니다. temperature는 자동으로 1.0으로 설정됩니다(사고가 활성화된 경우 API에서 요구됨).

```
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));
```

고급 예제

완전한 제어를 위해 typed request/response 클래스를 사용하십시오. ThinkingType을 'enabled'로 설정하고 ThinkingBudgetTokens를 원하는 토큰 예산(최소 1024)으로 설정하십시오. 응답에는 thinking 및 text 콘텐츠 블록이 포함됩니다.

```
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;
}
```

Thinking을 사용한 멀티턴

다중 턴 대화에서 확장된 사고(extended thinking)를 사용할 때, 이전 응답의 thinking 및 redacted_thinking 블록은 대화에서 다시 전달되어야 합니다. ContentBlocks 배열을 사용하여 이러한 블록을 포함하십시오.

```
// 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:** 확장 thinking을 활성화하려면 'enabled'로, 끄려면 'disabled'로 설정하십시오.
- **ThinkingBudgetTokens:** thinking을 위한 최대 토큰 수(최소 1024). MaxTokens보다 작아야 합니다.

Notes

- thinking이 활성화된 경우 Temperature는 1.0이어야 합니다(편의 메서드가 이를 자동으로 설정합니다).
- thinking이 활성화되면 TopK를 사용할 수 없습니다.
- BudgetTokens는 최소 1024 이상이어야 하며 MaxTokens보다 작아야 합니다.
- 응답 콘텐츠 블록에는 thinking, redacted_thinking, text 유형이 포함될 수 있습니다.

Anthropic | Documents

Claude는 메시지 내의 콘텐츠 블록으로 전송된 PDF 문서와 텍스트 파일을 처리할 수 있습니다. 문서는 base64로 인코딩된 데이터 또는 URL을 통해 전송되며, Claude는 그 내용을 분석, 요약하고 그에 관한 질문에 답할 수 있습니다.

간단한 예제

편의 메서드를 사용하여 질문이 포함된 PDF 문서를 보냅니다.

```
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"));
```

Citations를 사용한 고급 예제

인용이 활성화된 문서를 보내려면 타입이 지정된 클래스를 사용하십시오. 인용이 활성화되면 Claude의 응답에 소스 문서의 특정 부분에 대한 참조가 포함됩니다.

```
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:** document 콘텐츠 블록의 경우 'document'로 설정하십시오.
- **SourceType:** 소스 유형: 'base64'(기본값), 'text' 또는 'url'.
- **MediaType:** MIME 유형입니다(예: application/pdf, text/plain).
- **Data:** base64 인코딩된 문서 데이터, 또는 SourceType이 'url'일 때 URL입니다.
- **Title:** 참조를 위한 선택적 문서 제목.
- **Context:** 문서에 대한 선택적 컨텍스트 또는 메타데이터입니다.
- **CitationsEnabled:** 응답에서 source 인용을 활성화하려면 True로 설정하십시오.

지원되는 형식

- **PDF:** application/pdf (최대 32 MB, 요청당 최대 100페이지)
- **Plain Text:** text/plain

Anthropic | Prompt Caching

프롬프트 캐싱을 사용하면 API 호출 간에 자주 사용되는 컨텍스트를 캐시하여 캐시 읽기 비용을 최대 90%까지 절감하고 지연 시간을 개선할 수 있습니다. 캐시된 콘텐츠는 `cache_control` 매개변수로 표시되며 캐시 TTL 윈도우 내의 요청 전반에서 재사용됩니다.

System Prompt Caching

긴 시스템 프롬프트를 캐싱하여 모든 요청에서 다시 처리하는 것을 피하십시오. `SystemCacheControl`을 `True`로 설정하면 시스템 프롬프트가 `cache_control`로 자동으로 래핑됩니다.

```
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

개별 content 블록에 `CacheControl`을 'ephemeral'로 설정하여 특정 content 블록(예: 큰 문서나 컨텍스트)을 캐시하십시오.

```
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
```

Tool Definition Caching

모든 요청에서 도구 정의를 다시 처리하지 않도록 도구 정의를 캐시합니다. 요청 간에 일정하게 유지되는 도구 정의가 많은 경우 유용합니다.

```
TsgcAnthropicClass_Request_Tool *oTool = new TsgcAnthropicClass_Request_Tool(NULL);
oTool->Name = "search_database";
oTool->Description = "Search the database for records.";
oTool->InputSchema = "{\n\"type\": \"object\", \"properties\": {\n\"query\": {\n\"type\": \"string\"}\n}}";
oTool->CacheControl = "ephemeral"; // Cache this tool definition
```

Properties

- **SystemCacheControl:** system 프롬프트를 임시 cache_control로 캐싱하려면 요청에서 True로 설정하십시오.
- **CacheControl:** 콘텐츠 블록 또는 tool 정의에서 'ephemeral'로 설정하여 캐싱용으로 표시합니다.
- **CacheCreationInputTokens:** (응답) 캐시 항목을 생성하는 데 사용된 토큰 수입니다.
- **CacheReadInputTokens:** (응답) 캐시에서 읽은 토큰 수(비용 절감).

가격

- **캐시 쓰기:** 기본 입력 토큰 가격의 1.25배(5분 TTL).
- **캐시 읽기:** 기본 입력 토큰 가격의 0.1배(90% 절감).
- **Cache TTL:** 기본적으로 5분입니다. 후속 요청은 TTL을 갱신합니다.

Anthropic | Citations

인용이 활성화된 상태로 문서가 전송되면, Claude의 응답에는 소스 문서의 특정 부분에 대한 참조가 포함됩니다. 이를 통해 주장을 검증하고 정보를 그 출처까지 추적할 수 있습니다.

Citations 활성화

document 콘텐츠 블록에서 CitationsEnabled를 True로 설정하십시오. Citations는 요청의 문서 전체에서 활성화하거나 전혀 활성화하지 않아야 합니다.

```
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
```

응답에서 Citations 읽기

응답의 텍스트 콘텐츠 블록에는 소스 참조가 있는 Citations 배열이 포함될 수 있습니다.

```
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();
}
```

인용 유형

- **char_location:** 일반 텍스트 문서의 문자 수준 참조입니다. 필드: DocumentIndex, DocumentTitle, StartCharIndex, EndCharIndex.
- **page_location:** PDF 문서의 페이지 수준 참조입니다. 필드: DocumentIndex, DocumentTitle, StartPageNumber, EndPageNumber.
- **content_block_location:** 사용자 지정 콘텐츠 문서의 블록 수준 참조입니다. 필드: DocumentIndex, DocumentTitle, StartBlockIndex, EndBlockIndex.
- **web_search_result_location:** 웹 검색 결과에 대한 참조. 필드: Url, Title, EncryptedIndex.

Anthropic | Web Search

Web Search 도구를 통해 Claude는 대화 중에 실시간 정보를 위해 웹을 검색할 수 있습니다. 이것은 Anthropic이 호스팅하고 자동으로 실행하는 내장 서버 측 도구입니다.

간단한 예제

편의 메서드를 사용하여 web search가 활성화된 메시지를 생성합니다.

```
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?"));
```

고급 예제

MaxUses와 같은 web search 도구 매개변수를 완전히 제어하려면 typed 클래스를 사용하십시오.

```
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;
}
```

내장 도구 유형

- **web_search_20250305:** 서버 측 웹 검색. 속성: Name ('web_search'), MaxUses.
- **code_execution_20250825:** 샌드박스화된 코드 실행입니다. 속성: Name ('code_execution'), MaxUses.
- **computer_20250124:** Computer use(마우스, 키보드, 스크린샷). 속성: Name('computer'), DisplayWidthPx, DisplayHeightPx.

Response Content Types

내장 도구가 사용될 때, 응답에는 추가 콘텐츠 블록 유형이 포함될 수 있습니다:

- **server_tool_use:** Claude가 내장 도구를 호출했음을 나타냅니다. 필드: Id, Name, Input.
- **web_search_tool_result:** 웹 검색 결과를 포함합니다. 필드: ToolUseId, Data(원시 JSON 콘텐츠).

Anthropic | Structured Outputs

Structured Outputs는 Claude가 JSON 스키마를 준수하는 응답을 반환하도록 강제합니다. 이는 스키마 정의와 일치하는 유효하고 파싱 가능한 JSON 출력을 보장합니다.

간단한 예제

JSON 스키마 출력으로 메시지를 만들려면 편의 메서드를 사용하십시오.

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";

String vSchema = "{ \"type\": \"object\", \"properties\": { \"name\": { \"type\": \"string\" }, \"age\": { \"type\": \"integer\" } }, \"required\": [ \"name\", \"age\" ], \"additionalProperties\": false }";

ShowMessage(Anthropic->CreateMessageJSON("claude-sonnet-4-20250514",
    "Extract the name and age: John is 30 years old.", vSchema));
```

고급 예제

구조화된 출력을 effort 매개변수와 결합하려면 형식화된 클래스를 사용하십시오.

```
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 =
        "{ \"type\": \"object\", \"properties\": { \"sentiment\": { \"type\": \"string\", \"enum\": [ \"positive\", \"negative\", \"neutral\" ] }, \"confidence\": { \"type\": \"number\" } }, \"required\": [ \"sentiment\", \"confidence\" ], \"additionalProperties\": false }";

    // Set effort level (low, medium, high, max)
    oRequest->Effort = "medium";

    TsgcAnthropicClass_Request_Message *oMessage = new TsgcAnthropicClass_Request_Message();
    oMessage->Role = "user";
    oMessage->Content = "Analyze the sentiment: I love this product!";
    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

도구 입력이 input_schema와 정확히 일치하도록 보장하기 위해 도구 정의에서 strict 모드를 활성화합니다.

```
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:** JSON Schema 문자열. Claude의 응답 텍스트는 이 스키마를 준수하는 유효한 JSON이 됩니다.
- **Effort:** 출력 품질 대 비용을 제어합니다. 값: 'low', 'medium', 'high'(기본값), 'max'(Opus 4.6 전용).
- **Strict:** (도구에서) True인 경우 도구 입력이 input_schema를 준수하도록 보장됩니다.

Anthropic | Files API

Files API를 사용하면 파일을 업로드, 나열, 검색, 다운로드 및 삭제할 수 있습니다. 업로드된 파일은 file source 유형의 document 콘텐츠 블록을 사용하여 메시지에서 참조할 수 있습니다.

참고: Files API에는 beta 헤더 `files-api-2025-04-14`가 필요합니다. **AnthropicOptions.BetaHeaders** 속성에서 이를 설정하십시오.

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;
}
```

메시지에서 파일 사용

SourceType을 'file'로 설정한 document 콘텐츠 블록을 사용하여 업로드된 파일을 참조합니다.

```
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.";
```

Delete a File

```
TsgcAnthropicClass_Response_File *oDeleted = Anthropic->DeleteFile("file_abc123");
try {
    ShowMessage("Deleted: " + oDeleted->Id);
} __finally {
    delete oDeleted;
}
```

API Methods

- **UploadFile:** 로컬 경로에서 파일을 업로드합니다. 파일 메타데이터(Id, Filename, MimeType, SizeBytes)를 반환합니다.
- **ListFiles:** 페이지 매김 지원(Limit, AfterId, BeforeId)으로 업로드된 모든 파일을 나열합니다.
- **GetFile:** ID로 특정 파일의 메타데이터를 검색합니다.

- **DownloadFile:** 파일의 콘텐츠를 다운로드합니다(code execution 도구로 생성된 파일만 다운로드 가능).
- **DeleteFile:** ID로 파일을 영구적으로 삭제합니다.

Anthropic | MCP Connector

MCP(Model Context Protocol) 커넥터를 사용하면 Claude가 외부 MCP 서버의 도구에 접근할 수 있습니다. 이를 통해 서드 파티 서비스 및 사용자 지정 도구 공급자와의 통합이 가능합니다.

참고: MCP connector에는 베타 헤더 `mcp-client-2025-11-20`이 필요합니다. **AnthropicOptions.BetaHeaders** 속성에 이를 설정하십시오.

간단한 예제

편의 메서드를 사용하여 MCP 서버로 메시지를 생성합니다.

```
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"));
```

고급 예제

인증을 포함하여 MCP 서버 구성을 완벽하게 제어하려면 유형이 지정된 클래스를 사용하십시오.

```
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_MCP_Server *oServer = new TsgcAnthropicClass_Request_MCP_Server();
    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;
}
```

MCP Server 속성

- **ServerType:** 현재 'url'만 지원됩니다.
- **Url:** MCP 서버의 HTTPS URL입니다.
- **Name:** 이 서버의 고유 식별자. toolset의 MCPServerName과 일치해야 합니다.
- **AuthorizationToken:** 인증된 서버를 위한 선택적 OAuth Bearer 토큰입니다.

Response Content Types

- **mcp_tool_use:** Claude가 MCP 도구를 호출했습니다. 필드: Id, Name, ServerName, Input.
- **mcp_tool_result:** MCP 도구 실행 결과입니다. 필드: ToolUsed, IsError, Data (콘텐츠).

Gemini

Google Gemini은 Google DeepMind가 개발한 멀티모달 AI 모델 제품군입니다. Gemini 모델은 텍스트 생성, 비전, 구조화된 출력, 임베딩 및 도구 사용을 지원하여 AI 기반 애플리케이션을 구축하는 강력한 기능을 제공합니다. sgcWebSockets 라이브러리는 Gemini API와 상호 작용하기 위한 Delphi 구성 요소 **TsgcHTTP_API_Gemini**를 제공합니다.

Gemini API

Gemini API는 AI 기반 애플리케이션을 구축하기 위해 Google Gemini 모델에 대한 액세스를 제공합니다. API는 콘텐츠 생성, 비전(이미지 이해), 구조화된 JSON 출력, 스트리밍, 토큰 계산, 임베딩, 도구 사용(함수 호출) 및 모델 목록을 지원합니다.

Features

- **Messages**
 - [Gemini Messages 예제](#)
- **Vision**
 - [Gemini Vision 예제](#)
- **Models**
 - [Gemini Models 예제](#)
- **Structured Outputs**
 - [Gemini Structured Outputs 예제](#)
- **Token Counting**
 - [Gemini Token Counting 예제](#)
- **Embeddings**
 - [Gemini Embeddings 예제](#)
- **Tool Use**
 - [Gemini Tool Use 예제](#)

구성

Gemini API는 인증에 API 키를 사용합니다. Google AI Studio의 [API Keys](#) 페이지를 방문하여 요청에 사용할 API 키를 검색하십시오.

API 키는 비밀임을 기억하십시오! 다른 사람과 공유하거나 클라이언트 측 코드에 노출하지 마십시오. 이 **API Key**는 구성 요소의 **GeminiOptions.ApiKey** 속성에 구성되어야 합니다.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "YOUR_API_KEY";
```

메시지

Gemini 모델에 콘텐츠를 보내고 생성된 응답을 받습니다. 모델은 제공된 입력을 기반으로 다음 메시지를 생성합니다.

- **_CreateContent:** 지정된 모델과 사용자 프롬프트로 콘텐츠를 생성합니다.
 - **Model:** 사용할 모델(예: gemini-2.0-flash).
 - **Message:** 사용자 메시지 콘텐츠.
 - **MaxOutputTokens:** 생성할 최대 토큰 수.
- **_CreateContentWithSystem:** 시스템 명령으로 콘텐츠를 생성합니다.
 - **Model:** 사용할 모델입니다.
 - **System:** 모델의 동작을 설정하는 시스템 명령입니다.
 - **Message:** 사용자 메시지 콘텐츠입니다.
 - **MaxOutputTokens:** 생성할 최대 토큰 수입니다.
- **_CreateContentStream:** 스트리밍(SSE)이 활성화된 콘텐츠를 생성합니다. 이벤트는 OnHTTPAPISSE 이벤트 핸들러를 통해 전달됩니다.

Vision

Gemini 모델은 텍스트 프롬프트와 함께 base64로 인코딩된 콘텐츠로 전달된 이미지를 이해할 수 있습니다.

- **_CreateVisionContent:** 텍스트 프롬프트와 함께 이미지를 보냅니다.
 - **Model:** 사용할 모델.
 - **Prompt:** 이미지와 함께 제공할 텍스트 프롬프트.
 - **Base64:** base64로 인코딩된 이미지 데이터.
 - **MediaType:** MIME 유형(image/jpeg, image/png, image/gif, image/webp).
 - **MaxOutputTokens:** 생성할 최대 토큰 수.

구조화된 출력(Structured Outputs)

예상되는 응답 형식을 정의하는 JSON 스키마를 제공하여 Gemini 모델에서 구조화된 JSON 출력을 생성합니다.

- **_CreateContentJSON:** 구조화된 JSON 출력으로 콘텐츠를 생성합니다.
 - **Model:** 사용할 모델입니다.
 - **Message:** 사용자 메시지 콘텐츠입니다.
 - **Schema:** 예상되는 출력 구조를 정의하는 JSON 스키마입니다.
 - **MaxOutputTokens:** 생성할 최대 토큰 수입니다.

모델

사용 가능한 Gemini 모델에 대한 세부 정보를 나열하고 검색합니다.

- **_GetModels:** 사용 가능한 모든 모델을 나열합니다.
- **_GetModel:** 특정 모델의 세부 정보를 가져옵니다.
 - **ModelId:** 검색할 모델의 식별자입니다.

Token Counting

모델에 보내기 전에 메시지의 토큰 수를 계산합니다.

- **_CountTokens:** 메시지에 대한 토큰을 계산합니다.
 - **Model:** 토큰화에 사용할 모델입니다.
 - **Message:** 토큰을 계산할 텍스트 콘텐츠입니다.

Embeddings

Gemini 모델을 사용하여 텍스트 콘텐츠에 대한 벡터 embeddings를 생성합니다.

- **_EmbedContent:** 텍스트에 대한 임베딩을 생성합니다.
 - **Model:** 임베딩 생성에 사용할 모델.
 - **Text:** 임베딩을 생성할 텍스트 콘텐츠.

TsgcHTTP_API_Gemini

콘텐츠 생성, 토큰 계산, embeddings를 위한 Google Gemini REST API 클라이언트입니다.

소개

콘텐츠 생성, 토큰 계산, embeddings를 위한 Google Gemini REST API 클라이언트입니다.

소개

콘텐츠 생성, 토큰 계산, embeddings를 위한 Google Gemini REST API 클라이언트입니다.
API key 및 선택적 HTTP/재시도/로그 설정으로 **GeminiOptions** 속성을 구성한 다음, request 객체와 함께 `CreateMessage/`
`CreateChatCompletion` 같은 타입이 지정된 메서드를 호출하고 타입이 지정된 `response` 클래스를 받으십시오.
공급자에 대한 배경 정보는 [Gemini](#) 가이드를 참조하십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 5개의 속성.
- 메서드 — 3개의 공개 메서드.
- 이벤트 — 처리할 수 있는 2개의 이벤트.

DeepSeek

DeepSeek은 강력한 오픈소스 언어 모델 구축에 중점을 둔 중국 AI 회사입니다. 이들의 모델은 코딩, 추론 및 범용 작업에서 뛰어나며, 경쟁력 있는 가격으로 강력한 성능을 제공합니다. sgcWebSockets 라이브러리는 DeepSeek API와 상호 작용하기 위한 Delphi 구성 요소 **TsgcHTTP_API_DeepSeek**을 제공합니다.

DeepSeek API

DeepSeek API는 AI 기반 애플리케이션을 구축하기 위한 DeepSeek 모델에 대한 액세스를 제공합니다. API는 텍스트 생성, vision(이미지 이해), 스트리밍 및 모델 목록을 지원합니다. API는 OpenAI 호환 형식을 따르므로 통합이 쉽습니다.

Features

- **Messages**
 - [DeepSeek Messages 예제](#)
- **Vision**
 - [DeepSeek Vision 예제](#)
- **Models**
 - [DeepSeek 모델 예제](#)

구성

DeepSeek API는 인증에 API 키를 사용합니다. 요청에 사용할 API 키를 검색하려면 DeepSeek Platform의 [API Keys](#) 페이지를 방문하십시오.

API 키는 비밀임을 기억하십시오! 다른 사람과 공유하거나 클라이언트 측 코드에 노출하지 마십시오.

이 **API Key**는 구성 요소의 **DeepSeekOptions.ApiKey** 속성에 구성되어야 합니다.

```
TsgcHTTP_API_DeepSeek *DeepSeek = new TsgcHTTP_API_DeepSeek(NULL);
DeepSeek->DeepSeekOptions->ApiKey = "YOUR_API_KEY";
```

메시지

텍스트 콘텐츠가 포함된 입력 메시지의 구조화된 목록을 보내면 모델이 대화에서 다음 메시지를 생성합니다.

- **_CreateMessage:** 지정된 모델과 사용자 프롬프트로 메시지를 생성합니다.
 - **Model:** 사용할 모델(예: deepseek-chat).
 - **Message:** 사용자 메시지 콘텐츠.
- **_CreateMessageWithSystem:** system prompt와 함께 메시지를 생성합니다.
 - **System:** assistant의 동작을 설정하는 system prompt입니다.
- **_CreateMessageStream:** 스트리밍(SSE)이 활성화된 메시지를 만듭니다. 이벤트는 OnHTTPAPISSE 이벤트 핸들러를 통해 전달됩니다.

Vision

DeepSeek은 메시지 내에서 base64로 인코딩된 콘텐츠로 전달된 이미지를 이해할 수 있습니다.

- **_CreateVisionMessage:** 텍스트 프롬프트와 함께 이미지를 보냅니다.
 - **ImageBase64:** base64로 인코딩된 이미지 데이터입니다.
 - **MediaType:** MIME 유형(image/jpeg, image/png, image/gif, image/webp)입니다.
 - **Prompt:** 이미지와 함께 제공할 텍스트 프롬프트입니다.

모델

사용 가능한 DeepSeek 모델을 나열합니다.

- **_GetModels:** 사용 가능한 모든 모델을 나열합니다.

TsgcHTTP_API_DeepSeek

Chat Completions를 위한 DeepSeek REST API 클라이언트.

소개

Chat Completions를 위한 DeepSeek REST API 클라이언트.

소개

Chat Completions를 위한 DeepSeek REST API 클라이언트.

DeepSeekOptions 속성을 API 키 및 선택적 HTTP/retry/log 설정으로 구성한 다음, `createMessage/createChatCompletion`과 같은 형식화된 메서드를 `request` 객체와 함께 호출하여 형식화된 `response` 클래스를 받으십시오. 제공자에 대한 배경은 [DeepSeek](#) 가이드를 참조하십시오.

참조

- 속성 — 구성 요소에서 사용할 수 있는 5개의 속성.
- 메서드 — 1개의 public 메서드.
- 이벤트 — 처리할 수 있는 2개의 이벤트.

DeepSeek | Messages

텍스트 콘텐츠가 포함된 입력 메시지의 구조화된 목록을 보내면 모델이 대화에서 다음 메시지를 생성합니다.

간단한 예제

DeepSeek에 Hello 메시지를 보냅니다.

```
TsgcHTTP_API_DeepSeek *DeepSeek = new TsgcHTTP_API_DeepSeek(NULL);
DeepSeek->DeepSeekOptions->ApiKey = "API_KEY";
ShowMessage(DeepSeek->_CreateMessage("deepseek-chat", "Hello!"));
```

System Prompt 예제

DeepSeek의 동작을 제어하기 위해 system prompt와 함께 메시지를 보냅니다.

```
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?"));
```

스트리밍 예제

Server-Sent Events (SSE)를 사용하여 응답을 실시간으로 스트리밍합니다. 스트리밍 이벤트를 수신하려면 OnHTTPAPISSE 이벤트 핸들러를 할당하십시오.

```
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

DeepSeek는 이미지를 이해하고 분석할 수 있습니다. 콘텐츠 블록 내에서 base64로 인코딩된 데이터로 이미지를 보낼 수 있습니다.

Vision Example

DeepSeek에게 설명을 요청하는 프롬프트와 함께 이미지를 보냅니다.

```
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 | Models

사용 가능한 DeepSeek 모델을 나열합니다.

List Models

사용 가능한 모든 DeepSeek 모델을 나열합니다.

```
TsgcHTTP_API_DeepSeek *DeepSeek = new TsgcHTTP_API_DeepSeek(NULL);
DeepSeek->DeepSeekOptions->ApiKey = "API_KEY";
ShowMessage(DeepSeek->_GetModels());
```


Gemini

Google Gemini은 Google DeepMind가 개발한 멀티모달 AI 모델 제품군입니다. Gemini 모델은 텍스트 생성, 비전, 구조화된 출력, 임베딩 및 도구 사용을 지원하여 AI 기반 애플리케이션을 구축하는 강력한 기능을 제공합니다. sgcWebSockets 라이브러리는 Gemini API와 상호 작용하기 위한 Delphi 구성 요소 **TsgcHTTP_API_Gemini**를 제공합니다.

Gemini API

Gemini API는 AI 기반 애플리케이션을 구축하기 위해 Google Gemini 모델에 대한 액세스를 제공합니다. API는 콘텐츠 생성, 비전(이미지 이해), 구조화된 JSON 출력, 스트리밍, 토큰 계산, 임베딩, 도구 사용(함수 호출) 및 모델 목록을 지원합니다.

Features

- **Messages**
 - [Gemini Messages 예제](#)
- **Vision**
 - [Gemini Vision 예제](#)
- **Models**
 - [Gemini Models 예제](#)
- **Structured Outputs**
 - [Gemini Structured Outputs 예제](#)
- **Token Counting**
 - [Gemini Token Counting 예제](#)
- **Embeddings**
 - [Gemini Embeddings 예제](#)
- **Tool Use**
 - [Gemini Tool Use 예제](#)

구성

Gemini API는 인증에 API 키를 사용합니다. Google AI Studio의 [API Keys](#) 페이지를 방문하여 요청에 사용할 API 키를 검색하십시오.

API 키는 비밀임을 기억하십시오! 다른 사람과 공유하거나 클라이언트 측 코드에 노출하지 마십시오. 이 **API Key**는 구성 요소의 **GeminiOptions.ApiKey** 속성에 구성되어야 합니다.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "YOUR_API_KEY";
```

메시지

Gemini 모델에 콘텐츠를 보내고 생성된 응답을 받습니다. 모델은 제공된 입력을 기반으로 다음 메시지를 생성합니다.

- **_CreateContent:** 지정된 모델과 사용자 프롬프트로 콘텐츠를 생성합니다.
 - **Model:** 사용할 모델(예: gemini-2.0-flash).
 - **Message:** 사용자 메시지 콘텐츠.
 - **MaxOutputTokens:** 생성할 최대 토큰 수.
- **_CreateContentWithSystem:** 시스템 명령으로 콘텐츠를 생성합니다.
 - **Model:** 사용할 모델입니다.
 - **System:** 모델의 동작을 설정하는 시스템 명령입니다.
 - **Message:** 사용자 메시지 콘텐츠입니다.
 - **MaxOutputTokens:** 생성할 최대 토큰 수입니다.
- **_CreateContentStream:** 스트리밍(SSE)이 활성화된 콘텐츠를 생성합니다. 이벤트는 OnHTTPAPISSE 이벤트 핸들러를 통해 전달됩니다.

Vision

Gemini 모델은 텍스트 프롬프트와 함께 base64로 인코딩된 콘텐츠로 전달된 이미지를 이해할 수 있습니다.

- **_CreateVisionContent:** 텍스트 프롬프트와 함께 이미지를 보냅니다.
 - **Model:** 사용할 모델.
 - **Prompt:** 이미지와 함께 제공할 텍스트 프롬프트.
 - **Base64:** base64로 인코딩된 이미지 데이터.
 - **MediaType:** MIME 유형(image/jpeg, image/png, image/gif, image/webp).
 - **MaxOutputTokens:** 생성할 최대 토큰 수.

구조화된 출력(Structured Outputs)

예상되는 응답 형식을 정의하는 JSON 스키마를 제공하여 Gemini 모델에서 구조화된 JSON 출력을 생성합니다.

- **_CreateContentJSON:** 구조화된 JSON 출력으로 콘텐츠를 생성합니다.
 - **Model:** 사용할 모델입니다.
 - **Message:** 사용자 메시지 콘텐츠입니다.
 - **Schema:** 예상되는 출력 구조를 정의하는 JSON 스키마입니다.
 - **MaxOutputTokens:** 생성할 최대 토큰 수입니다.

모델

사용 가능한 Gemini 모델에 대한 세부 정보를 나열하고 검색합니다.

- **_GetModels:** 사용 가능한 모든 모델을 나열합니다.
- **_GetModel:** 특정 모델의 세부 정보를 가져옵니다.
 - **ModelId:** 검색할 모델의 식별자입니다.

Token Counting

모델에 보내기 전에 메시지의 토큰 수를 계산합니다.

- **_CountTokens:** 메시지에 대한 토큰을 계산합니다.
 - **Model:** 토큰화에 사용할 모델입니다.
 - **Message:** 토큰을 계산할 텍스트 콘텐츠입니다.

Embeddings

Gemini 모델을 사용하여 텍스트 콘텐츠에 대한 벡터 embeddings를 생성합니다.

- **_EmbedContent:** 텍스트에 대한 임베딩을 생성합니다.
 - **Model:** 임베딩 생성에 사용할 모델.
 - **Text:** 임베딩을 생성할 텍스트 콘텐츠.

Gemini | Messages

Gemini 모델에 프롬프트를 전송하고 생성된 콘텐츠를 수신합니다. Gemini API는 텍스트 입력을 처리하고 모델 응답을 반환하기 위해 generateContent 엔드포인트를 사용합니다.

간단한 예제

Gemini에 Hello 메시지를 보냅니다.

```
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

Gemini의 동작을 제어하기 위해 system instruction과 함께 메시지를 보냅니다.

```
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?"));
```

스트리밍 예제

Server-Sent Events (SSE)를 사용하여 응답을 실시간으로 스트리밍합니다. 스트리밍 이벤트를 수신하려면 OnHTTPAPISSE 이벤트 핸들러를 할당하십시오.

```
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);
}
```

고급 예제

콘텐츠 생성 매개변수를 완전히 제어하려면 typed request/response 클래스를 사용하십시오.

```
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

Gemini는 이미지를 이해하고 분석할 수 있습니다. content 부분 내에서 base64로 인코딩된 데이터로 이미지를 보낼 수 있습니다.

Vision Example

분석을 위해 base64로 인코딩된 이미지를 Gemini에 보냅니다.

```
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 | 모델

사용 가능한 Gemini 모델에 대한 정보를 나열하고 조회합니다.

List Models

사용 가능한 모든 모델을 나열합니다.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";
ShowMessage(Gemini->_GetModels());
```

Get Model

특정 모델에 대한 세부 정보를 가져옵니다.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";
ShowMessage(Gemini->_GetModel("gemini-2.0-flash"));
```

고급 예제

모델 데이터에 대한 완전한 제어를 위해 형식화된 response 클래스를 사용하십시오.

```
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

Gemini는 사용자가 정의한 스키마를 준수하는 구조화된 JSON 형식으로 응답을 반환할 수 있습니다. 이는 스키마 정의와 일치하는 유효하고 파싱 가능한 JSON 출력을 보장합니다.

간단한 예제

JSON 스키마 출력으로 콘텐츠를 생성하려면 편의 메서드를 사용하십시오.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";

String vSchema = "{\n\"type\": \"object\", \"properties\": {\n\"name\": {\n\"type\": \"string\"}, \"\n\"age\": {\n\"type\": \"integer\"}}, \"required\": [\n\"name\", \"age\"]}";

ShowMessage(Gemini->_CreateContentJSON("gemini-2.0-flash",
    "Extract the name and age: John is 30 years old.", vSchema));
```

고급 예제

형식화된 클래스를 사용하여 ResponseMimeType과 ResponseSchema로 구조화된 출력을 구성합니다.

```
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 =
        "{\n\"type\": \"object\", \"properties\": {\n\"sentiment\": {\n\"type\": \"string\"}, \"\n\"confidence\": {\n\"type\": \"number\"}}, \"required\": [\n\"sentiment\", \"confidence\"]}";

    SetLength(oContents, 1);
    oContents[0] = new TsgcGeminiClass_Request_Content();
    oContents[0]->Role = "user";
    SetLength(oParts, 1);
    oParts[0] = new TsgcGeminiClass_Request_Part();
    oParts[0]->Text = "Analyze the sentiment: I love this product!";
    oContents[0]->Parts = oParts;
    oRequest->Contents = oContents;

    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:** 구조화된 JSON 출력을 활성화하려면 'application/json'으로 설정하십시오.
- **ResponseSchema:** 예상되는 출력 구조를 정의하는 JSON Schema 문자열.

Gemini | 토큰 카운팅

API에 보내기 전에 메시지의 토큰 수를 계산합니다. 이는 비용을 추정하고 메시지가 모델 한도 내에 있는지 확인하는 데 도움이 됩니다.

간단한 예제

메시지에 대한 토큰을 계산합니다.

```
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?"));
```

고급 예제

토큰 카운팅 매개변수를 완전히 제어하려면 형식이 지정된 요청/응답 클래스를 사용하십시오.

```
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

텍스트 콘텐츠에 대한 벡터 임베딩을 생성합니다. 임베딩은 텍스트의 의미를 포착하며 유사성 검색, 클러스터링 및 기타 NLP 작업에 사용할 수 있습니다.

간단한 예제

텍스트에 대한 임베딩을 생성합니다.

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";
ShowMessage(Gemini->EmbedContent("text-embedding-004", "Hello world"));
```

고급 예제

임베딩 데이터에 대한 전체 제어를 위해 타입 응답 클래스를 사용하십시오.

```
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 | Tool Use

Gemini는 함수 호출(도구 사용)을 지원하여 대화 중에 모델이 호출할 수 있는 함수를 정의할 수 있습니다. Gemini가 함수를 호출하기로 결정하면, 응답에서 `functionCall` 부분을 반환합니다. 그런 다음 함수를 실행하고 결과를 다시 보냅니다.

Function Calling 흐름

1. 이름, 설명 및 매개변수(JSON Schema)로 함수 선언을 정의합니다.
2. 함수 선언과 함께 content 생성 요청을 보냅니다.
3. Gemini는 function 이름과 인수를 포함하는 **functionCall** 부분으로 응답합니다.
4. 제공된 인수로 함수를 실행합니다.
5. 결과를 포함하는 **functionResponse** 부분과 함께 새 요청을 보냅니다.
6. Gemini가 최종 답변으로 응답합니다.

예제

weather 함수를 정의하고 함수 호출을 처리합니다.

```
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\": [\"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:** 사용 가능한 함수를 정의하는 TsgcGeminiClass_Request_FunctionDeclaration의 배열입니다.
 - **Name:** 함수 이름입니다.
 - **Description:** 함수가 수행하는 작업에 대한 설명입니다.
 - **Parameters:** 함수 매개변수를 정의하는 JSON Schema 문자열입니다.
- **ToolChoice:** 함수 호출 동작을 제어합니다. 모드를 설정하십시오(예: 'AUTO', 'ANY', 'NONE').

Ollama

Ollama는 대규모 언어 모델을 로컬에서 실행하기 위한 오픈소스 도구입니다. Llama, Mistral, Gemma, Phi 등 다양한 모델을 지원하여, 클라우드 API 접근 없이 로컬 AI 추론을 가능하게 합니다. sgcWebSockets 라이브러리는 Ollama API와 상호 작용하기 위한 Delphi 구성 요소 **TsgcHTTP_API_Ollama**를 제공합니다.

Ollama API

Ollama API는 AI 기반 애플리케이션을 위해 로컬에서 실행되는 모델에 대한 액세스를 제공합니다. API는 텍스트 생성, 스트리밍, 모델 관리 및 임베딩을 지원합니다. 모델이 로컬에서 실행되므로 기본적으로 API 키가 필요하지 않습니다. 호스트와 포트는 구성 가능합니다.

Features

- 메시지
 - [Ollama Messages 예제](#)
- Models
 - [Ollama Models 예제](#)
- Embeddings
 - [Ollama Embeddings 예제](#)

구성

Ollama는 로컬에서 실행되며 기본적으로 **http://localhost:11434**에서 수신 대기합니다. **OllamaOptions.Host** 속성에서 호스트를 구성하십시오. API 키는 선택 사항이며 Ollama 인스턴스에 인증을 구성한 경우에만 필요합니다.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
```

메시지

텍스트 콘텐츠가 포함된 입력 메시지의 구조화된 목록을 보내면 모델이 대화에서 다음 메시지를 생성합니다.

- **_CreateMessage:** 지정된 모델과 사용자 프롬프트로 메시지를 생성합니다.
 - **Model:** 사용할 모델(예: llama3).
 - **Message:** 사용자 메시지 내용.
- **_CreateMessageWithSystem:** system prompt와 함께 메시지를 생성합니다.
 - **System:** assistant의 동작을 설정하는 system prompt입니다.
- **_CreateMessageStream:** 스트리밍(SSE)이 활성화된 메시지를 만듭니다. 이벤트는 OnHTTPAPISSE 이벤트 핸들러를 통해 전달됩니다.

모델

로컬에서 사용 가능한 모델을 관리하고 쿼리합니다.

- **_GetModels:** 사용 가능한 모든 모델을 나열합니다.
- **_GetTags:** 로컬에서 사용 가능한 모든 모델 태그를 나열합니다.
- **_ShowModel:** 특정 모델에 대한 자세한 정보를 검색합니다.
 - **Model:** 쿼리할 모델의 이름입니다.
- **_PullModel:** Ollama 모델 라이브러리에서 모델을 다운로드합니다.
 - **Model:** pull할 모델의 이름.
- **_DeleteModel:** 로컬에서 사용 가능한 모델을 삭제합니다.
 - **Model:** 삭제할 모델의 이름입니다.

Embeddings

로컬에서 실행 중인 모델을 사용하여 텍스트 입력에서 벡터 임베딩을 생성합니다.

- **_CreateEmbeddings:** 주어진 텍스트에 대한 임베딩을 생성합니다.
 - **Model:** 임베딩에 사용할 모델입니다(예: llama3).
 - **Input:** 임베딩을 생성할 텍스트입니다.

TsgcHTTP_API_Ollama

로컬 Chat Completions 및 Embeddings를 위한 Ollama REST API 클라이언트입니다.

소개

로컬 Chat Completions 및 Embeddings를 위한 Ollama REST API 클라이언트입니다.

소개

로컬 Chat Completions 및 Embeddings를 위한 Ollama REST API 클라이언트입니다.
API 키와 선택적 HTTP/재시도/로그 설정으로 **OllamaOptions** 속성을 구성한 다음, 요청 객체와 함께 `CreateMessage/`
`CreateChatCompletion`과 같은 형식화된 메서드를 호출하고 형식화된 응답 클래스를 받으십시오.
공급자에 대한 배경 정보는 [Ollama](#) 가이드를 참조하십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 5개의 속성.
- 메서드 — 2개의 public 메서드.
- 이벤트 — 처리할 수 있는 2개의 이벤트.

Ollama | Messages

텍스트 콘텐츠가 포함된 입력 메시지의 구조화된 목록을 보내면 모델이 대화에서 다음 메시지를 생성합니다.

간단한 예제

로컬 Ollama 모델에 Hello 메시지를 보냅니다.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
ShowMessage(Ollama->_CreateMessage("llama3", "Hello!"));
```

System Prompt 예제

모델의 동작을 제어하기 위해 시스템 프롬프트와 함께 메시지를 보냅니다.

```
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?"));
```

스트리밍 예제

Server-Sent Events (SSE)를 사용하여 응답을 실시간으로 스트리밍합니다. 스트리밍 이벤트를 수신하려면 OnHTTPAPISSE 이벤트 핸들러를 할당하십시오.

```
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 | Models

로컬에서 사용 가능한 Ollama 모델을 관리하고 쿼리합니다.

List Models

사용 가능한 모든 모델을 나열합니다.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
ShowMessage(Ollama->_GetModels());
```

Get Tags

로컬에서 사용 가능한 모든 모델 태그를 나열합니다.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
ShowMessage(Ollama->_GetTags());
```

Show Model

특정 모델에 대한 상세 정보를 검색합니다.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
ShowMessage(Ollama->_ShowModel("llama3"));
```

Pull Model

Ollama 모델 라이브러리에서 모델을 다운로드합니다.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
ShowMessage(Ollama->_PullModel("llama3"));
```


Ollama | Embeddings

로컬에서 실행 중인 Ollama 모델을 사용하여 텍스트 입력에서 벡터 embeddings를 생성합니다. Embeddings는 시맨틱 검색, 클러스터링 및 기타 NLP 작업에 사용할 수 있습니다.

Create Embeddings

지정된 텍스트 입력에 대한 임베딩을 생성합니다.

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);  
Ollama->OllamaOptions->Host = "http://localhost:11434";  
ShowMessage(Ollama->_CreateEmbeddings("llama3", "Hello world"));
```

Grok

Grok는 xAI가 개발한 대화형 AI assistant로, 실시간 지식으로 유용하고 유익한 응답을 제공하도록 설계되었습니다. Grok 모델은 강력한 추론 능력과 최신 정보 액세스로 잘 알려져 있습니다. sgcWebSockets 라이브러리는 Grok API와 상호 작용하기 위한 Delphi 구성 요소 **TsgcHTTP_API_Grok**을 제공합니다.

Grok API

Grok API는 AI 기반 애플리케이션을 구축하기 위한 Grok 모델에 대한 액세스를 제공합니다. API는 텍스트 생성, vision(이미지 이해), streaming 및 모델 목록을 지원합니다.

Features

- **Messages**
 - [Grok Messages 예제](#)
- **Vision**
 - [Grok Vision Examples](#)
- **Models**
 - [Grok 모델 예제](#)

구성

Grok API는 인증에 API 키를 사용합니다. 요청에 사용할 API 키를 검색하려면 [xAI Console](#)을 방문하십시오. API 키는 비밀임을 기억하십시오! 다른 사람과 공유하거나 클라이언트 측 코드에 노출하지 마십시오. 이 **API Key**는 구성 요소의 **GrokOptions.ApiKey** 속성에 구성되어야 합니다.

```
TsgcHTTP_API_Grok *Grok = new TsgcHTTP_API_Grok(NULL);
Grok->GrokOptions->ApiKey = "YOUR_API_KEY";
```

메시지

텍스트 콘텐츠가 포함된 입력 메시지의 구조화된 목록을 보내면 모델이 대화에서 다음 메시지를 생성합니다.

- **_CreateMessage:** 지정된 모델과 사용자 프롬프트로 메시지를 생성합니다.
 - **Model:** 사용할 모델(예: grok-3).
 - **Message:** 사용자 메시지 내용.
- **_CreateMessageWithSystem:** system prompt와 함께 메시지를 생성합니다.
 - **System:** assistant의 동작을 설정하는 system prompt입니다.
- **_CreateMessageStream:** 스트리밍(SSE)이 활성화된 메시지를 만듭니다. 이벤트는 OnHTTPAPISSE 이벤트 핸들러를 통해 전달됩니다.

Vision

Grok은 메시지 내에 base64로 인코딩된 콘텐츠로 전달된 이미지를 이해할 수 있습니다.

- **_CreateVisionMessage:** 텍스트 프롬프트와 함께 이미지를 보냅니다.
 - **ImageBase64:** base64로 인코딩된 이미지 데이터입니다.
 - **MediaType:** MIME 유형(image/jpeg, image/png, image/gif, image/webp)입니다.
 - **Prompt:** 이미지와 함께 제공할 텍스트 프롬프트입니다.

모델

사용 가능한 Grok 모델을 나열합니다.

- **_GetModels:** 사용 가능한 모든 모델을 나열합니다.

TsgcHTTP_API_Grok

TsgcHTTP_API_Grok 구성 요소입니다.

소개

Chat Completions 및 모델 목록을 위한 xAI Grok REST API 클라이언트입니다.

소개

Chat Completions 및 모델 목록을 위한 xAI Grok REST API 클라이언트입니다.

GrokOptions 속성을 API 키와 선택적 HTTP/재시도/로그 설정으로 구성한 다음, 요청 객체와 함께 `CreateMessage/`
`CreateChatCompletion`과 같은 형식화된 메서드를 호출하고 형식화된 응답 클래스를 받으십시오.
제공자에 대한 배경은 [Grok](#) 가이드를 참조하십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 7개의 속성입니다.
- 메서드 — 1개의 public 메서드.
- Events — 처리할 수 있는 0개의 이벤트.

Grok | Messages

텍스트 콘텐츠가 포함된 입력 메시지의 구조화된 목록을 보내면 모델이 대화에서 다음 메시지를 생성합니다.

간단한 예제

Grok에 Hello 메시지를 보냅니다.

```
TsgcHTTP_API_Grok *Grok = new TsgcHTTP_API_Grok(NULL);
Grok->GrokOptions->ApiKey = "API_KEY";
ShowMessage(Grok->_CreateMessage("grok-3", "Hello!"));
```

System Prompt 예제

Grok의 동작을 제어하기 위해 system prompt가 있는 메시지를 보냅니다.

```
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?"));
```

스트리밍 예제

Server-Sent Events (SSE)를 사용하여 응답을 실시간으로 스트리밍합니다. 스트리밍 이벤트를 수신하려면 OnHTTPAPISSE 이벤트 핸들러를 할당하십시오.

```
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

Grok은 메시지 내에서 base64로 인코딩된 콘텐츠로 전달된 이미지를 이해할 수 있습니다. vision 엔드포인트를 사용하여 텍스트 프롬프트와 함께 이미지를 보내고 설명 또는 분석을 받으십시오.

Vision Example

분석을 위해 base64로 인코딩된 이미지를 Grok에 전송합니다.

```
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

사용 가능한 Grok 모델과 기본 정보를 나열합니다.

List Models Example

사용 가능한 모든 Grok 모델 목록을 검색합니다.

```
TsgcHTTP_API_Grok *Grok = new TsgcHTTP_API_Grok(NULL);  
Grok->GrokOptions->ApiKey = "API_KEY";  
ShowMessage(Grok->_GetModels);
```

Mistral AI

Mistral AI는 효율적이고 강력한 언어 모델을 개발하는 프랑스 인공지능 회사입니다. 이들의 모델은 경쟁력 있는 효율성을 유지하면서 추론, 코딩 및 다국어 작업 전반에서 강력한 성능으로 알려져 있습니다. sgcWebSockets 라이브러리는 Mistral AI API와 상호 작용하기 위한 Delphi 구성 요소 **TsgcHTTP_API_Mistral**을 제공합니다.

Mistral API

Mistral API는 AI 기반 애플리케이션을 구축하기 위한 Mistral 모델에 대한 액세스를 제공합니다. API는 텍스트 생성, vision(이미지 이해), 스트리밍, 임베딩, JSON 모드 및 모델 목록을 지원합니다.

Features

- **Messages**
 - [Mistral Messages 예제](#)
- **Vision**
 - [Mistral Vision 예제](#)
- **Models**
 - [Mistral Models 예제](#)
- **Embeddings**
 - [Mistral Embeddings 예제](#)

구성

Mistral API는 인증에 API 키를 사용합니다. Mistral Console의 [API Keys](#) 페이지를 방문하여 요청에 사용할 API 키를 검색하십시오.

API 키는 비밀임을 기억하십시오! 다른 사람과 공유하거나 클라이언트 측 코드에 노출하지 마십시오. 이 **API Key**는 구성 요소의 **MistralOptions.ApiKey** 속성에 구성되어야 합니다.

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);
Mistral->MistralOptions->ApiKey = "YOUR_API_KEY";
```

메시지

텍스트 콘텐츠가 포함된 입력 메시지의 구조화된 목록을 보내면 모델이 대화에서 다음 메시지를 생성합니다.

- **_CreateMessage:** 지정된 모델과 사용자 프롬프트로 메시지를 생성합니다.
 - **Model:** 사용할 모델(예: mistral-large-latest).
 - **Message:** 사용자 메시지 내용.
- **_CreateMessageWithSystem:** system prompt와 함께 메시지를 생성합니다.
 - **System:** assistant의 동작을 설정하는 system prompt입니다.
- **_CreateMessageStream:** 스트리밍(SSE)이 활성화된 메시지를 만듭니다. 이벤트는 OnHTTPAPISSE 이벤트 핸들러를 통해 전달됩니다.
- **_CreateMessageJSON:** JSON 모드가 활성화된 메시지를 생성합니다. 모델은 유효한 JSON 응답을 반환합니다.
 - **Model:** 사용할 모델입니다.
 - **Message:** 사용자 메시지 내용입니다.

Vision

Mistral은 메시지 내에서 base64로 인코딩된 콘텐츠로 전달된 이미지를 이해할 수 있습니다.

- **_CreateVisionMessage:** 텍스트 프롬프트와 함께 이미지를 보냅니다.
 - **ImageBase64:** base64로 인코딩된 이미지 데이터입니다.
 - **MediaType:** MIME 유형(image/jpeg, image/png, image/gif, image/webp)입니다.
 - **Prompt:** 이미지와 함께 제공할 텍스트 프롬프트입니다.

모델

사용 가능한 Mistral 모델을 나열합니다.

- **_GetModels:** 사용 가능한 모든 모델을 나열합니다.

Embeddings

시맨틱 검색, 클러스터링, 기타 기계 학습 작업에 사용할 수 있는 주어진 입력의 벡터 표현을 가져옵니다.

- **_CreateEmbeddings:** 입력 텍스트를 나타내는 임베딩 벡터를 생성합니다.
 - **Model:** 사용할 모델(예: mistral-embed).
 - **Input:** 임베딩을 얻으 입력 텍스트.

TsgcHTTP_API_Mistral

Chat Completions 및 Embeddings를 위한 Mistral REST API 클라이언트입니다.

소개

Chat Completions 및 Embeddings를 위한 Mistral REST API 클라이언트입니다.

소개

Chat Completions 및 Embeddings를 위한 Mistral REST API 클라이언트입니다.
API 키와 선택적 HTTP/재시도/로그 설정으로 **MistralOptions** 속성을 구성한 다음, 요청 객체와 함께 `CreateMessage/`
`CreateChatCompletion`과 같은 형식화된 메서드를 호출하여 형식화된 응답 클래스를 받으십시오.
공급자에 대한 배경 정보는 [Mistral](#) 가이드를 참조하십시오.

참조

- 속성 — 구성 요소에서 사용할 수 있는 5개의 속성.
- 메서드 — 2개의 공용 메서드입니다.
- 이벤트 — 처리할 수 있는 2개의 이벤트입니다.

Mistral | Messages

텍스트 콘텐츠가 포함된 입력 메시지의 구조화된 목록을 보내면 모델이 대화에서 다음 메시지를 생성합니다.

간단한 예제

Mistral에 Hello 메시지를 보냅니다.

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);
Mistral->MistralOptions->ApiKey = "API_KEY";
ShowMessage(Mistral->_CreateMessage("mistral-large-latest", "Hello!"));
```

System Prompt 예제

Mistral의 동작을 제어하기 위해 system prompt와 함께 메시지를 보냅니다.

```
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?"));
```

스트리밍 예제

Server-Sent Events (SSE)를 사용하여 응답을 실시간으로 스트리밍합니다. 스트리밍 이벤트를 수신하려면 OnHTTPAPISSE 이벤트 핸들러를 할당하십시오.

```
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

Mistral에서 JSON 형식의 응답을 요청합니다.

```
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

Mistral은 메시지 내에서 base64로 인코딩된 콘텐츠로 전달된 이미지를 이해할 수 있습니다. vision 엔드포인트를 사용하여 텍스트 프롬프트와 함께 이미지를 보내고 설명 또는 분석을 받으십시오.

Vision Example

분석을 위해 base64로 인코딩된 이미지를 Mistral에 보냅니다.

```
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

사용 가능한 Mistral 모델과 기본 정보를 나열합니다.

List Models Example

사용 가능한 모든 Mistral 모델 목록을 검색합니다.

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);  
Mistral->MistralOptions->ApiKey = "API_KEY";  
ShowMessage(Mistral->_GetModels);
```

Mistral | Embeddings

시맨틱 검색, 클러스터링, 기타 기계 학습 작업에 사용할 수 있는 주어진 입력의 벡터 표현을 가져옵니다.

Embeddings Example

텍스트 입력에 대한 embedding vector를 만듭니다.

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);
Mistral->MistralOptions->ApiKey = "API_KEY";
ShowMessage(Mistral->_CreateEmbeddings("mistral-embed", "Hello world"));
```

IoT

사물 인터넷(IoT)은 개인용 컴퓨터 및 모바일 디바이스와 같은 기존 컴퓨팅 플랫폼을 넘어 인터넷 연결을 전통적으로 "멍청한" 또는 인터넷이 지원되지 않는 모든 범위의 물리적 디바이스와 일상적인 사물로 확장하는 개념을 의미합니다. 전자 장치, 인터넷 연결 및 기타 형태의 하드웨어(예: 센서)를 내장한 이러한 디바이스는 인터넷을 통해 다른 디바이스와 통신하고 상호작용할 수 있으며, 원격으로 모니터링하고 제어할 수 있습니다.

sgcWebSockets 패키지는 다음 IoT 클라이언트를 구현합니다:

1. Amazon AWS IoT: AWS IoT는 센서, 액추에이터, 임베디드 마이크로컨트롤러, 스마트 가전제품 같은 인터넷 연결 장치와 AWS Cloud 간의 안전한 양방향 통신을 제공합니다. 이를 통해 여러 장치에서 텔레메트리 데이터를 수집하고, 데이터를 저장하고 분석할 수 있습니다. 또한 사용자가 휴대폰이나 태블릿에서 이 장치들을 제어할 수 있도록 하는 애플리케이션을 만들 수도 있습니다.

2. Azure IoT Hub: IoT Hub는 클라우드에서 호스팅되는 관리형 서비스로, IoT 애플리케이션과 관리하는 디바이스 간의 양방향 통신을 위한 중앙 메시지 허브 역할을 합니다. Azure IoT Hub를 사용하여 수백만 개의 IoT 디바이스와 클라우드 호스팅 솔루션 백엔드 간의 안정적이고 안전한 통신으로 IoT 솔루션을 구축할 수 있습니다.

IoT Amazon MQTT Client

AWS IoT란 무엇입니까?

AWS IoT는 센서, 액추에이터, 임베디드 마이크로 컨트롤러 또는 스마트 가전과 같은 인터넷에 연결된 디바이스와 AWS Cloud 간에 안전한 양방향 통신을 제공합니다. 이를 통해 여러 디바이스에서 텔레메트리 데이터를 수집하고, 데이터를 저장 및 분석할 수 있습니다. 또한 사용자가 휴대폰이나 태블릿에서 이러한 디바이스를 제어할 수 있는 애플리케이션을 만들 수 있습니다.

Message broker

디바이스와 AWS IoT 애플리케이션이 서로 메시지를 게시하고 수신할 수 있는 안전한 메커니즘을 제공합니다. MQTT 프로토콜을 직접 사용하거나 MQTT over WebSocket을 사용하여 게시 및 구독할 수 있습니다.

AWS IoT 메시지 브로커는 AWS IoT로/에서 메시지를 보내고 받을 수 있게 하는 게시/구독 브로커 서비스입니다. AWS IoT와 통신할 때 클라이언트는 `Sensor/temp/room1`과 같은 토픽으로 주소가 지정된 메시지를 보냅니다.

메시지 브로커는 차례로 해당 토픽에 대한 메시지를 받기 위해 등록한 모든 클라이언트에게 메시지를 보냅니다. 메시지를 보내는 행위를 게시(publishing)라고 합니다. 토픽 필터에 대한 메시지를 받기 위해 등록하는 행위를 구독(subscribing)이라고 합니다.

topic 네임스페이스는 각 AWS 계정 및 리전 쌍에 대해 격리됩니다. 예를 들어, 한 AWS 계정의 `Sensor/temp/room1` topic은 다른 AWS 계정의 `Sensor/temp/room1` topic과 독립적입니다. 이는 리전에 대해서도 마찬가지입니다. `us-east-1`의 동일한 AWS 계정에 있는 `Sensor/temp/room1` topic은 `us-east-2`의 동일한 topic과 독립적입니다. AWS IoT는 AWS 계정 및 리전 간 메시지 송수신을 지원하지 않습니다.

메시지 브로커는 모든 클라이언트 세션 목록과 각 세션의 구독을 유지합니다. 토픽에 메시지가 게시되면 브로커는 해당 토픽에 매핑되는 구독이 있는 세션을 확인합니다. 그런 다음 브로커는 현재 연결된 클라이언트가 있는 모든 세션으로 게시 메시지를 전달합니다.

MQTT Client

`TsgcIoTAmazon_MQTT_Client`는 AWS IoT에 연결하는 데 사용되는 구성 요소입니다. 하나의 클라이언트는 하나의 디바이스에만 연결할 수 있습니다. 클라이언트는 일반 MQTT 프로토콜을 사용하여 연결하고 X.509 클라이언트 인증서를 사용하여 인증합니다.

AWS IoT에 연결하려면 클라이언트에 다음 속성이 필요합니다:

Amazon.ClientId: 클라이언트의 식별, 선택적.

Amazon.Endpoint: MQTT 클라이언트가 연결할 서버 이름입니다.

Amazon.Port: 기본적으로 포트 8883을 사용합니다. 포트가 443이면 연결하기 위해 자동으로 ALPN을 사용합니다 (사용자 지정 Indy 버전 필요).

AWS IoT Core는 MQTT 및 MQTT over WebSocket Secure (WSS) 프로토콜을 사용하여 메시지를 게시하고 구독하는 장치와 클라이언트를 지원합니다. 다음 표는 AWS IoT 장치 엔드포인트가 지원하는 프로토콜과 사용하는 인증 방법 및 포트를 나열합니다.

프로토콜	Authentication	Port	ALPN Protocol Name
WebSocket을 통한 MQTT	Signature Version 4	443	
WebSocket을 통한 MQTT	Custom Authentication	443	

MQTT	X.509 client certificate	443	x-amzn-mqtt-ca
MQTT	X.509 client certificate	8883	
MQTT	Custom Authentication	443	mqtt

인증서 인증

Amazon AWS 콘솔에서 인증서를 생성하고 저장된 경로를 설정해야 합니다.

IOHandler로 **OpenSSL**을 사용하는 경우 다음 경로에 인증서를 설정해야 합니다

Certificate.Enabled: 인증서를 사용하려면 True로 설정하십시오.
Certificate.CertFile: X.509 클라이언트 인증서 경로입니다.
Certificate.KeyFile: X.509 클라이언트 키 파일의 경로입니다.

SChannel을 IOHandler로 사용할 때, 먼저 PEM 인증서 + Key를 PFX 인증서로 변환하십시오. 이를 위해서는 OpenSSL 바이너리가 필요합니다:

```
openssl pkcs12 -inkey 884ccf73ff-private.pem.key -in 884ccf73ff-certificate.pem.crt -export -out 884ccf73ff-cert.pfx
```

그런 다음 다음 경로를 설정하십시오(키 파일은 인증서에 이미 포함되어 있으므로 설정할 필요가 없습니다).

Certificate.Enabled: 인증서를 사용하려면 True로 설정하십시오.
Certificate.CertFile: PFX 인증서 경로입니다.

SignatureV4 Authentication

Amazon AWS 콘솔에서 사용자를 생성하고 WebSocket 요청에 서명하는 데 사용될 Access 및 Secret 키를 저장해야 합니다.

SignatureV4.Enabled: 이 유형의 인증을 사용하려면 True로 설정하십시오.
SignatureV4.Region: 디바이스가 위치한 리전(예: us-east-1).
SignatureV4.AccessKey: Amazon 콘솔에서 생성되거나 임시 자격 증명으로 얻은 access key입니다.
SignatureV4.SecretKey: amazon 콘솔에서 생성하거나 임시 자격 증명으로 얻은 secret key입니다.
SignatureV4.SessionToken: (조건부) 임시 보안 자격 증명을 사용하는 경우 여기에 보안 토큰을 설정하십시오.
OpenSSL_Options: openssl 라이브러리의 구성입니다.

APIVersion: 어떤 OpenSSL API를 사용할지 정의할 수 있게 합니다.

oslAPI_1_0: API 1.0 OpenSSL을 사용하며, Indy가 지원하는 최신 버전입니다

oslAPI_1_1: API 1.1 OpenSSL을 사용하며, 당사의 사용자 정의 Indy 라이브러리가 필요하고 OpenSSL 1.1.1 라이브러리(TLS 1.3 지원 포함)를 사용할 수 있게 합니다.

oslAPI_3_0: API 3.0 OpenSSL을 사용하며, 사용자 지정 Indy 라이브러리가 필요하고 OpenSSL 3.0.0 라이브러리(TLS 1.3 지원 포함)를 사용할 수 있습니다.

LibPath: 여기에서 openssl 라이브러리가 위치한 곳을 구성할 수 있습니다.

oslpNone: 이것이 기본값이며, openssl 라이브러리는 바이너리가 있는 동일한 폴더 또는 알려진 경로에 있어야 합니다.

oslpDefaultFolder: 모든 IDE 퍼스널리티에 대해 라이브러리가 위치해야 하는 openssl 경로를 자동으로 설정합니다.

oslpCustomFolder: 이 옵션이 선택된 경우, LibPathCustom 속성에 전체 경로를 정의하십시오.

LibPathCustom: LibPath = oslpCustomFolder일 때 openssl 라이브러리가 있는 전체 경로를 여기에 정의하십시오.

UnixSymLinks: Unix 시스템에서 SymLinks 로딩을 활성화하거나 비활성화합니다(기본적으로 활성화되어 있으며, OSX64에서는 예외):

oslsSymLinksDefault: 기본적으로 활성화되지만 OSX64에서는 예외입니다(MacOS Monterey 이후 버전 없이 라이브러리를 로드하려고 하면 실패함).

oslsSymLinksLoadFirst: 버전 라이브러리를 로드하기 전에 SymLink를 로드하고 먼저 시도합니다.
oslsSymLinksLoad: 버전 라이브러리를 로드하려고 시도한 후 SymLinks를 로드합니다.
oslsSymLinksDontLoad: SymLinks를 로드하지 않습니다.

*SignatureV4에는 Indy 10.5.7+가 필요합니다

Custom Authentication

사용자 정의 인증을 사용하면 authorizer 리소스를 사용하여 클라이언트를 인증하고 권한을 부여하는 방법을 정의할 수 있습니다. 장치는 요청의 헤더 필드나 쿼리 매개변수(MQTT over WebSockets 프로토콜의 경우) 또는 MQTT CONNECT 메시지의 user name 및 password 필드(MQTT 및 MQTT over WebSockets 프로토콜의 경우)에 자격 증명을 전달합니다.

CustomAuthentication.Enabled: 이 유형의 Authentication을 사용하려면 True로 설정하십시오.
CustomAuthentication.Parameters: 서버에 전달될 쿼리 매개변수를 여기에 설정하십시오(기본값은 /mqtt)
CustomAuthentication.Headers: 여기에 사용자 지정 헤더 필드를 넣을 수 있습니다.
CustomAuthentication.WebSockets: true로 설정되면 연결은 WebSocket 프로토콜을 통해 작동하고, 그렇지 않으면 일반 TCP를 통해 작동합니다.

MQTTAuthentication.Enabled: mqtt 연결에서 사용자 이름/비밀번호를 전달해야 하는 경우 이 속성을 활성화하십시오

MQTTAuthentication.Username: mqtt 연결의 사용자 이름

MQTTAuthentication.Password: mqtt 연결의 비밀번호.

클라이언트는 선택적으로 ClientId를 보내 클라이언트 연결을 식별할 수 있으며, 그러면 다른 클라이언트는 이 클라이언트가 연결, 구독, 연결 끊김... 할 때마다 알림을 받기 위해 구독할 수 있습니다.

권한 부여

포트 8883을 사용하여 연결할 수 없고 전송으로 TCP(기본값)를 사용하는 경우, amazon은 "AWS IoT Core policy"를 사용하여 클라이언트 및 구독에 권한 부여를 제공하거나 제공하지 않습니다. 아마도 클라이언트 id를 권한 부여해야 할 것입니다. Amazon AWS 콘솔에 들어가서 IoT Core로 이동하여 "Secure/Policies" 메뉴에 접근한 다음, IoT Thing에 연결된 정책을 선택하고 끝에서 연결이 어떻게 구성되어 있는지 확인하십시오. 예:

```
{
  "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-*"
  ]
}
```

이 구성은 ID가 sdk-java, basicPubSub, sdk-nodejs-*인 클라이언트만 연결을 허용함을 의미합니다. 적절히 변경하고 다시 시도하십시오.

여전히 작동하지 않으면 로그를 활성화하고 cloudwatch에서 연결할 수 없는 이유를 확인하십시오.

기타 속성

MQTTHeartBeat: 활성화하면 x초마다 ping을 보내 MQTT 연결을 유지하려고 시도합니다.

Interval: 각 ping 사이의 초 수입니다.

MQTTAuthentication: 활성화하면 MQTT 연결에 username과 password를 포함합니다.

UserName: 사용자의 이름

Password: secret string

WatchDog: 활성화되면 예기치 않은 연결 끊김이 감지될 때 서버에 자동으로 재연결을 시도합니다.

Interval: 재연결 시도 전 초.

Attempts: 최대 재연결 시도 횟수입니다. 0은 무제한을 의미합니다.

LogFile: 활성화하면 소켓 메시지를 로그 파일에 저장합니다(디버깅에 유용). 로그 파일에 대한 액세스는 여러 스레드에서 액세스되는 경우 스레드 안전하지 않습니다.

Enabled: 활성화하면 소켓으로 메시지를 수신하고 보낼 때마다 파일에 저장됩니다.

FileName: 파일 이름의 전체 경로입니다.

Implementation

Amazon MQTT 구현은 MQTT 버전 3.1.1을 기반으로 하지만 다음과 같이 사양에서 벗어납니다:

- AWS IoT에서 Quality of Service(QoS) 0으로 토픽을 구독한다는 것은 메시지가 0회 이상 전달됨을 의미합니다. 메시지가 두 번 이상 전달될 수 있습니다. 두 번 이상 전달되는 메시지는 다른 패킷 ID로 전송될 수 있습니다. 이러한 경우 DUP 플래그가 설정되지 않습니다.
- AWS IoT는 QoS 2를 사용한 게시 및 구독을 지원하지 않습니다. AWS IoT 메시지 브로커는 QoS 2가 요청될 때 PUBACK 또는 SUBACK을 보내지 않습니다.
- 연결 요청에 응답할 때 메시지 브로커는 CONNACK 메시지를 보냅니다. 이 메시지에는 연결이 이전 세션을 재개하는 지를 나타내는 플래그가 포함됩니다. 두 MQTT 클라이언트가 동일한 클라이언트 ID로 동시에 연결하면 이 플래그의 값이 올바르게 맞을 수 없습니다.
- 클라이언트가 토픽을 구독할 때, 메시지 브로커가 SUBACK을 보내는 시점과 클라이언트가 새로 일치하는 메시지를 수신하기 시작하는 시점 사이에 지연이 있을 수 있습니다.
- MQTT 사양은 게시자가 브로커에게 토픽으로 전송된 마지막 메시지를 보관하고 향후의 모든 토픽 구독자에게 전송하도록 요청하는 규정을 제공합니다. AWS IoT는 보관 메시지를 지원하지 않습니다. 메시지를 보관하도록 요청하면 연결이 끊깁니다.
- 메시지 브로커는 클라이언트 ID를 사용하여 각 클라이언트를 식별합니다. 클라이언트 ID는 MQTT 페이로드의 일부로 클라이언트에서 메시지 브로커로 전달됩니다. 동일한 클라이언트 ID를 가진 두 클라이언트는 메시지 브로커에 동시에 연결할 수 없습니다. 클라이언트가 다른 클라이언트가 사용 중인 클라이언트 ID를 사용하여 메시지 브로커에 연결하면 두 클라이언트 모두에 CONNACK 메시지가 전송되고 현재 연결된 클라이언트는 연결이 해제됩니다.
- 드문 경우에, 메시지 브로커가 동일한 논리적 PUBLISH 메시지를 다른 패킷 ID로 다시 보낼 수 있습니다.
- 메시지 브로커는 메시지와 ACK가 수신되는 순서를 보장하지 않습니다.

Connect to AWS IoT

먼저 AWS 콘솔에 로그인하여 새 디바이스를 등록하고 이 디바이스에 대한 X.509 인증서를 생성해야 합니다. 완료되면 새 TsgcIoTAmazon_MQTT_Client를 생성하고 AWS IoT Server에 연결할 수 있습니다. 예를 들어:

```
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");
}
```

Topics

메시지 브로커는 토픽을 사용하여 게시 클라이언트에서 구독 클라이언트로 메시지를 라우팅합니다. 슬래시(/)는 토픽 계층을 구분하는 데 사용됩니다. 다음 표는 구독할 때 토픽 필터에서 사용할 수 있는 와일드카드를 나열합니다. #은 구독하는 토픽의 마지막 문자여야 합니다. 현재 트리와 모든 하위 트리를 일치시켜 와일드카드로 작동합니다.

예를 들어, Sensor/#에 대한 구독은 Sensor/, Sensor/temp, Sensor/temp/room1에 게시된 메시지를 수신하지만, Sensor에 게시된 메시지는 수신하지 않습니다.

+ 토픽 계층에서 정확히 하나의 항목과 일치합니다. 예를 들어, Sensor/+/room1에 대한 구독은 Sensor/temp/room1, Sensor/moisture/room1 등으로 게시된 메시지를 수신합니다.

```
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);
}
```

예약된 토픽

다음 메서드는 예약된 토픽을 구독/게시하는 데 사용됩니다.

Subscribe_ClientConnected(const aClientId: String): 지정된 클라이언트 ID를 가진 MQTT 클라이언트가 AWS IoT에 연결될 때 AWS IoT가 이 토픽에 게시합니다

Subscribe_ClientDisconnected(const aClientId: String): 지정된 클라이언트 ID를 가진 MQTT 클라이언트가 AWS IoT에서 연결을 끊으면 AWS IoT가 이 토픽에 게시합니다

Subscribe_ClientSubscribed(const aClientId: String): 지정된 클라이언트 ID를 가진 MQTT 클라이언트가 MQTT 토픽을 구독하면 AWS IoT가 이 토픽에 게시합니다

Subscribe_ClientUnSubscribed(const aClientId: String): AWS IoT는 지정된 클라이언트 ID를 가진 MQTT 클라이언트가 MQTT 토픽 구독을 해제할 때 이 토픽에 게시합니다

Publish_Rule(const aRuleName, aText: String): 장치 또는 애플리케이션이 이 토픽에 게시하여 규칙을 직접 트리거합니다

Publish_DeleteShadow(const aThingName, aText: String): 디바이스 또는 애플리케이션이 shadow를 삭제하기 위해 이 토픽에 게시합니다

Subscribe_DeleteShadow(const aThingName: String): 장치나 애플리케이션이 shadow를 삭제하기 위해 이 토픽을 구독합니다

Subscribe_ShadowDeleted(const aThingName: String): Device Shadow 서비스는 shadow가 삭제될 때 이 topic에 메시지를 보냅니다

Subscribe_ShadowRejected(const aThingName: String): shadow 삭제 요청이 거부되면 Device Shadow 서비스가 이 토픽으로 메시지를 보냅니다.

Publish_ShadowGet(const aThingName, aText: String): 애플리케이션 또는 thing이 이 topic에 빈 메시지를 게시하여 shadow를 가져옵니다

Subscribe_ShadowGet(const aThingName: String): 애플리케이션 또는 thing이 shadow를 얻기 위해 이 topic을 구독합니다

Subscribe_ShadowGetAccepted(const aThingName: String): shadow 요청이 성공적으로 이루어지면 Device Shadow 서비스가 이 topic에 메시지를 보냅니다

Subscribe_ShadowGetRejected(const aThingName: String): shadow에 대한 요청이 거부될 때 Device Shadow 서비스가 이 토픽으로 메시지를 전송합니다

Publish_ShadowUpdate(const aThingName, aText: String): thing 또는 애플리케이션이 shadow를 업데이트하기 위해 이 토픽에 게시합니다.

Subscribe_ShadowUpdateAccepted(const aThingName: String): Device Shadow 서비스는 shadow에 대한 업데이트가 성공적으로 이루어지면 이 topic에 메시지를 보냅니다

Subscribe_ShadowUpdateRejected(const aThingName: String): Device Shadow 서비스는 shadow에 대한 업데이트가 거부될 때 이 topic으로 메시지를 보냅니다

Subscribe_ShadowUpdateDelta(const aThingName: String): Device Shadow 서비스는 shadow의 reported 및 desired 섹션 간에 차이가 감지될 때 이 토픽에 메시지를 보냅니다

Subscribe_ShadowUpdateDocuments(const aThingName: String): shadow에 대한 업데이트가 성공적으로 수행될 때마다 AWS IoT가 이 토픽에 상태 문서를 게시합니다

영속 세션

persistent 세션은 MQTT 메시지 브로커에 대한 진행 중인 연결을 나타냅니다. 클라이언트가 persistent 세션을 사용하여 AWS IoT 메시지 브로커에 연결하면 메시지 브로커는 클라이언트가 연결 중에 하는 모든 구독을 저장합니다. 클라이언트가 연결을 끊으면 메시지 브로커는 미확인 QoS 1 메시지와 클라이언트가 구독한 토픽에 게시된 새 QoS 1 메시지를 저장합니다. 클라이언트가 persistent 세션에 다시 연결하면 모든 구독이 복원되고 저장된 모든 메시지가 초당 최대 10개의 메시지 속도로 클라이언트에 전송됩니다.

OnMQTTBeforeConnect 이벤트에서 **cleanSession** 매개변수를 False로 설정하여 MQTT 영구 세션을 생성합니다. 클라이언트에 대한 세션이 없으면 새 영구 세션이 생성됩니다. 클라이언트에 대한 세션이 이미 존재하면 재개됩니다.

장치는 영속 세션이 있는지 확인하기 위해 **OnMQTTConnect** 이벤트의 **Session** 속성을 확인해야 합니다. **Session이 True**이면 영속 세션이 존재하고 저장된 메시지가 클라이언트에게 전달됩니다. **Session이 False**이면 영속 세션이 없으며 클라이언트는 자신의 topic 필터를 다시 구독해야 합니다.

영구 세션의 기본 만료 기간은 1시간입니다. 만료 기간은 메시지 브로커가 클라이언트 연결 끊김(MQTT disconnect 또는 timeout)을 감지할 때 시작됩니다. 영구 세션 만료 기간은 표준 제한 증가 프로세스를 통해 늘릴 수 있습니다. 클라이언트가 만료 기간 내에 세션을 재개하지 않으면 세션이 종료되고 관련된 저장 메시지가 모두 폐기됩니다. 만료 기간은 대략적이며, 세션은 구성된 기간보다 최대 30분 더 길게(그러나 더 짧지는 않게) 영속화될 수 있습니다.

임시 자격 증명

AWS IoT Core는 Identity Pools를 통해 얻은 Temporary Credentials와 함께 작동할 수 있습니다. 2가지 유형의 Identity가 있습니다:

- **UnAuthenticated:** IAM에서 policy type을 설정하기만 하면 됩니다
- **Authenticated:** IAM 및 AWS IoT Core 정책에서 정책 유형을 설정해야 합니다.

Unauthenticated

Unauthenticated 자격 증명을 사용하는 경우, IAM 메뉴에서 자동으로 생성된 UnAuthenticated Role에 정책을 연결하기만 하면 됩니다. 그런 다음 Cognito 서비스가 반환한 Access, Secret Key 및 Token을 설정하여 클라이언트를 구성하십시오. 인증되지 않은 자격 증명을 가져오는 .NET 코드를 아래에서 확인하십시오

```
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

인증된 자격 증명, IAM 메뉴에서 자동으로 생성된 Authenticated Role에 정책을 연결하고 AWS IoT Core 정책에서 사용자의 정책을 연결해야 합니다.

따라서 IoT Core policies 메뉴에서 새 정책을 생성하고 새 사용자가 인증할 때마다 이 정책을 해당 사용자에게 연결하십시오. 정책을 연결하거나 lambda 함수를 만들려면 다음 AWS 명령을 사용할 수 있습니다.

```
aws iot attach-policy --policy-name PolicyName --target us-east-1:XXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Device Provisioning

Fleet Provisioning 서비스는 다음 MQTT API 작업을 지원합니다:

- **CreateCertificateFromCsr:** certificate signing request(CSR)에서 인증서를 생성합니다. AWS IoT는 Amazon Root certificate authority(CA)에 의해 서명된 클라이언트 인증서를 제공합니다. 새 인증서는 PENDING_ACTIVATION 상태를 가집니다. 이 인증서로 사물을 프로비저닝하기 위해 RegisterThing을 호출하면, 템플릿에 설명된 대로 인증서 상태가 ACTIVE 또는 INACTIVE로 변경됩니다.
- **CreateKeysAndCertificate:** 새 키와 인증서를 생성합니다. AWS IoT는 Amazon Root certificate authority(CA)가 서명한 클라이언트 인증서를 제공합니다. 새 인증서는 PENDING_ACTIVATION 상태입니다. 이 인증서로 thing을 프로비저닝하기 위해 RegisterThing을 호출하면, 템플릿에 설명된 대로 인증서 상태가 ACTIVE 또는 INACTIVE로 변경됩니다.
- **RegisterThing:** 사전 정의된 템플릿을 사용하여 thing을 프로비저닝합니다.

CreateCertificateFromCsr

인증서를 생성하려면 CertificateSigningRequest를 매개변수로 전달하여 CreateCertificateFromCsr 메서드를 사용하십시오. 이 요청에 대한 응답을 받으려면 먼저 다음 메서드를 구독하십시오: SubscribeCreateCertificateFromCsrResponse 및 SubscribeCreateCertificateFromCsrError

CreateKeysAndCertificate

새 인증서와 키를 생성하려면 CreateKeysAndCertificate 메서드를 사용하십시오. 이 요청에 대한 응답을 받으려면 먼저 SubscribeCreateKeysAndCertificateResponse 및 SubscribeCreateKeysAndCertificateError 메서드를 구독하십시오

RegisterThing

SubscribeEx로 이전에 연 구독을 제거합니다.

IoT Azure MQTT Client

Azure IoT Hub란?

IoT Hub는 클라우드에서 호스팅되는 관리형 서비스로, IoT 애플리케이션과 관리하는 장치 간의 양방향 통신을 위한 중앙 메시지 허브 역할을 합니다. Azure IoT Hub를 사용하여 수백만 개의 IoT 장치와 클라우드 호스팅 솔루션 백엔드 간의 신뢰할 수 있고 안전한 통신으로 IoT 솔루션을 구축할 수 있습니다. 사실상 모든 장치를 IoT Hub에 연결할 수 있습니다.

IoT Hub는 장치에서 클라우드로, 그리고 클라우드에서 장치로의 통신을 모두 지원합니다. IoT Hub는 device-to-cloud telemetry, 장치로부터의 파일 업로드, 클라우드에서 장치를 제어하기 위한 request-reply 메서드와 같은 여러 메시징 패턴을 지원합니다. IoT Hub 모니터링은 장치 생성, 장치 실패 및 장치 연결과 같은 이벤트를 추적하여 솔루션의 상태를 유지하는 데 도움이 됩니다.

IoT Hub의 기능은 제조에 사용되는 산업 장비 관리, 의료 분야의 귀중한 자산 추적, 사무실 건물 사용량 모니터링과 같은 확장 가능하고 완전한 기능을 갖춘 IoT 솔루션을 구축하는 데 도움이 됩니다.

Message broker

IoT Hub는 기기가 데이터를 보낼 수 있는 보안 통신 채널을 제공합니다. IoT Hub와 기기 SDK는 기기 연결을 위해 다음 프로토콜을 지원합니다:

- **MQTT**
- **MQTT over WebSockets**

여러 인증 유형이 다양한 장치 기능을 지원합니다:

- IoT 솔루션을 빠르게 시작하기 위한 **SAS** 토큰 기반 인증입니다.
- 안전하고 표준 기반의 인증을 위한 개별 **X.509 인증서** 인증입니다.

MQTT Client

TsgcIoTAzure_MQTT_Client는 Azure IoT에 연결하는 데 사용되는 구성 요소입니다. 하나의 클라이언트는 단 하나의 장치에만 연결할 수 있습니다. 클라이언트는 일반 MQTT 프로토콜을 사용하여 연결하고 SAS / X.509 Client Certificate를 사용하여 인증합니다.

Azure IoT Hub에 연결하려면 클라이언트에 다음 속성이 필요합니다:

Azure.IoTHub: MQTT 클라이언트가 연결할 서버 이름입니다.
Azure.Deviceld: Azure IoT Hub의 장치 이름입니다.

Azure는 여러 인증 유형을 허용하며, 기본적으로 SAS 토큰을 사용합니다.

SAS 인증

SAS.Enabled: 인증이 SAS를 사용하는 경우 활성화하십시오.
SAS.SecretKey: Azure IoT 계정의 SAS Token입니다.
SAS.KeyName: Shared Access Key Name입니다.
SAS.Expiry: SAS Token이 만료되기 전 분 수를 설정합니다. 기본값은 1440(24시간)입니다.

연결 문자열이 있는 경우, **ReadConnectionString** 메서드를 사용하여 연결 문자열 값을 자동으로 읽을 수 있습니다. 예제:

```
ReadConnectionString('HostName=yourhub.azure-devices.net;SharedAccessKeyName=iotechowner;SharedAccessKey=Yj7RRPnkSDTv+UCFLgWIP/FrbDymZv4qVAIoTLHUFR8=');
```


X509 Certificates

IOHandler로 **OpenSSL** 사용

Certificate.Enabled: 인증이 인증서를 사용하는 경우 활성화하십시오.

Certificate.CertFile: X.509 클라이언트 인증서 경로입니다.

Certificate.KeyFile: X.509 클라이언트 키 파일의 경로입니다.

Certificate.Password: 인증서에 비밀번호가 있는 경우 여기에 설정하십시오.

Version: TLS 버전이며, 기본적으로 TLS 1.0을 사용합니다.

IOHandler로 **SChannel**을 사용

Certificate.Enabled: 인증이 인증서를 사용하는 경우 활성화하십시오.

Certificate.CertFile: PFX 인증서 경로(먼저 인증서를 PFX로 변환해야 함). [자세히 보기](#).

Certificate.Password: 인증서에 비밀번호가 있는 경우 여기에 설정하십시오.

Version: TLS 버전이며, 기본적으로 TLS 1.0을 사용합니다.

기타 속성:

MQTTHeartBeat: 활성화하면 x초마다 ping을 보내 MQTT 연결을 유지하려고 시도합니다.

Interval: 각 ping 사이의 초 수입니다.

WatchDog: 활성화되면 여기치 않은 연결 끊김이 감지될 때 서버에 자동으로 재연결을 시도합니다.

Interval: 재연결 시도 전 초.

Attempts: 최대 재연결 시도 횟수입니다. 0은 무제한을 의미합니다.

LogFile: 활성화하면 소켓 메시지를 로그 파일에 저장합니다(디버깅에 유용). 로그 파일에 대한 액세스는 여러 스레드에서 액세스되는 경우 스레드 안전하지 않습니다.

Enabled: 활성화하면 소켓으로 메시지를 수신하고 보낼 때마다 파일에 저장됩니다.

FileName: 파일 이름의 전체 경로입니다.

Azure MQTT 구현은 MQTT 버전 3.1.1을 기반으로 하지만 다음과 같이 사양에서 벗어납니다:

- IoT Hub는 QoS 2 메시지를 지원하지 않습니다. 디바이스 앱이 QoS 2로 메시지를 게시하면, IoT Hub는 네트워크 연결을 닫습니다.
- IoT Hub는 Retain 메시지를 유지하지 않습니다. 장치가 RETAIN 플래그를 1로 설정하여 메시지를 보내면 IoT Hub는 메시지에 x-opt-retain 애플리케이션 속성을 추가합니다. 이 경우 retain 메시지를 유지하는 대신 IoT Hub는 이를 백엔드 앱으로 전달합니다.
- IoT Hub는 장치당 하나의 활성 MQTT 연결만 지원합니다. 동일한 장치 ID를 대신하는 새 MQTT 연결은 IoT Hub가 기존 연결을 끊게 합니다.

Azure IoT Hub에 연결

먼저 Azure 계정에 로그인하고, 새 장치를 등록하고, 이 장치에 대한 인증 방법을 만들어야 합니다. 완료되면 새 TsgIoTAzure_MQTT_Client를 만들고 Azure IoT Hub에 연결할 수 있습니다.

예제:

```
TsgIoTAzure_MQTT_Client *oClient = new TsgIoTAzure_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(TsgWSConnection *Connection, const bool Session, const TmqttConnReturnCode *ReturnCoc
{
```

```
ShowMessage("Connected to Azure IoT Hub");
}
```

Device To Cloud

장치 앱에서 솔루션 백엔드로 정보를 보낼 때, IoT Hub는 다음 옵션을 노출합니다:

1. 시계열 텔레메트리 및 알림을 위한 **Device-to-cloud 메시지**.

```
oClient->Send_DeviceToCloud("{\"temp\": 10}", azuIoTQoS1);
```

TStringList를 사용하여 키-값 속성을 보낼 수 있습니다. TStringList를 원하는 메시지 속성으로 채운 후 이를 인수로 전달하기 만 하면 됩니다.

```
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();
}
```

메시지의 **ContentType**과 **ContentEncoding**을 설정해야 하는 경우 이러한 값을 Properties List에 추가해야 합니다. 이러한 속성의 이름은 Azure에 의해 정의됩니다.

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. 사용 가능한 기능, 조건 또는 장기 실행 워크플로의 상태와 같은 디바이스 상태 정보를 보고하기 위한 **디바이스 트윈의 보고된 속성**. 예: 구성 및 소프트웨어 업데이트.

```
oClient->Set_DeviceTwinsProperties("1", "{\"sgc\":1}");
```

Cloud To Device

IoT Hub는 디바이스 앱이 백엔드 앱에 기능을 노출하는 세 가지 옵션을 제공합니다:

1. 결과의 즉각적인 확인이 필요한 통신을 위한 **Direct 메서드**입니다. Direct 메서드는 종종 팬을 켜는 것과 같은 장치의 대화형 제어에 사용됩니다.

```
oClient->Subscribe_DirectMethod();
```


다음 메서드를 사용하여 public 메서드에 응답할 수 있습니다.

```
oClient->RespondPublicMethod(RequestId, Status, "Your Response", azuIoTQoS1);
```

2. 장기 실행 명령을 위한 **Twin의 desired 속성**은 장치를 특정 desired 상태로 만들기 위한 것입니다. 예를 들어, 텔레메트리 전송 간격을 30분으로 설정합니다. 다음 메서드를 사용하여 속성을 가져올 수 있습니다.

```
oClient->Get_DeviceTwinsProperties("1");
```

3. 장치 앱으로의 단방향 알림을 위한 **Cloud-to-device 메시지**입니다. 메시지를 받으려면 먼저 구독해야 합니다.

```
oClient->Subscribe_CloudToDevice;
```

메시지는 **OnMQTTPublish** 이벤트(텍스트 페이로드) 또는 **OnMQTTPublishEx** 이벤트(Value, Bytes 및 Stream 속성을 가진 TsgcWSMQTTPublishData로서의 페이로드)에서 수신됩니다.

```
void AzureIoTMQTTPublish(TsgcWSConnection *Connection, string aTopic, string aText)
{
    DoLog("Received Message: " + aTopic + " " + aText);
}
```

Upload Files

IoT hub는 연결된 디바이스에 공유 액세스 서명(SAS) URI 또는 X509 인증서를 제공하여 디바이스로부터의 파일 업로드를 용이하게 합니다.

SAS를 선택하면 다음 속성을 설정해야 합니다:

- **Azure.IoTHub:** 예: youriothub.azure-devices.net
- **Azure.Deviceld:** 예: myDevice
- **SAS.SecretKey:** 예: Yj7RRPnkSDTv+UCFLgwIP/FrbDymZv4qVAIoTLHUFR8=
- **SAS.KeyName:** 예: iothubowner
- **SAS.Enabled:** 값을 true로 설정해야 합니다.

X509 인증서를 선택하는 경우 다음 속성을 설정해야 합니다:

- **Certificate.CertFile:** PEM 인증서의 경로입니다.
- **Certificate.KeyFile:** PEM 키 파일의 경로입니다.
- **Certificate.Enabled:** 값을 true로 설정해야 합니다.

Azure 서버에 파일을 업로드하려면 **UploadFile** 메서드를 사용하십시오. Overwrite 매개변수가 true로 설정되면 기존 파일을 교체합니다. Overwrite 매개변수가 false로 설정되면 파일이 존재하지 않는 경우에만 업로드하고 존재하는 경우 오류를 발생시킵니다(이것이 기본 옵션입니다).

```
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

Azure IoT를 사용하면 DPS를 사용하여 코드에서 장치를 등록할 수 있습니다. 현재 라이브러리는 Scope Id와 Registration Id를 매개변수로 전달하여 장치를 등록하는 것을 지원합니다.

```
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

IoT Hub에 연결된 디바이스와 상호 작용하려면 **Azure IoT Explorer** 애플리케이션을 사용할 수 있습니다. 수신된 텔레메트리 메시지, 등록된 디바이스 등을 볼 수 있습니다. 애플리케이션은 무료이며 다음에서 다운로드할 수 있습니다:

<https://github.com/Azure/azure-iot-explorer/releases>

HTTP

HTTP 프로토콜을 사용하면 이미지와 HTML 문서와 같은 리소스를 서버에서 가져올 수 있습니다. 이는 클라이언트-서버 프로토콜로, 클라이언트가 필요한 리소스를 서버에 요청함을 의미합니다.

클라이언트가 서버에 연결하려고 할 때, 다음 단계를 따릅니다:

1. 새 TCP 연결을 엽니다
2. 요청된 데이터와 함께 서버에 메시지를 보냅니다

```
GET / HTTP/1.1
Host: server.com
Accept-Language: en-us
```

3. 서버가 보낸 응답을 읽습니다

```
HTTP/1.1 200 OK
Server: Apache
Content-Length: 120
Content-Type: text/html
...
```

구성 요소

- **HTTP/2:** HTTP/2(또는 h2)는 push, 스트림 멀티플렉싱 및 프레임 제어를 웹에 제공하는 바이너리 프로토콜입니다.
- **HTTP/1 Client:** TldHTTP 클라이언트 구성 요소에서 상속되는 비시각적 구성 요소입니다.
- **OAuth2:** OAuth2를 사용하면 서드파티 애플리케이션이 HTTP 서비스에 대한 제한된 액세스를 받을 수 있습니다.
- **JWT:** JWT를 사용하면 일부 클레임을 주장하는 JSON을 페이로드로 보유하는, 선택적 서명 및/또는 암호화가 있는 데이터 생성할 수 있습니다.
- **Amazon SQS:** 마이크로서비스, 분산 시스템 및 서버리스 애플리케이션을 위한 완전 관리형 메시지 큐 서비스입니다.
- **Google Cloud Pub/Sub:** 애플리케이션 간 메시징을 제공하며, 애플리케이션 간에 신뢰할 수 있고, 다대다이며, 비동기적인 메시징을 제공하도록 설계되었습니다.
- **Google Calendar:** Google Calendar API V3를 사용할 수 있게 합니다: 캘린더, 이벤트 가져오기, 자신의 캘린더와 동기화...
- **Google FCM:** Firebase Cloud Messaging을 사용하여 알림을 보냅니다.

HTTP/2

HTTP/2는 HTTP 1.1 프로토콜의 진화로, 기본적으로 네트워크를 사용할 때 더 효율적이라고 합니다. 의미론은 동일하므로 이전 프로토콜과 호환되도록 설계되었습니다.

HTTP 1.1 제한 사항

HTTP 1.1은 연결당 하나의 요청만 처리하도록 제한되므로, 일반적으로 클라이언트는 서버에서 파일을 요청하기 위해 둘 이상의 연결을 사용합니다. 그러나 이는 문제를 야기합니다. 열린 TCP 연결이 너무 많으면 서버 리소스를 사용하기 위한 클라이언트 간 경쟁이 발생하고, 더 많은 클라이언트가 서버에 연결할수록 성능이 점점 더 낮아지기 때문입니다.

주요 기능

- HTTP/2는 바이너리 프로토콜입니다(HTTP 1.1은 텍스트 프로토콜임을 기억하십시오).
- HTTP/2는 TLS와 ALPN을 통해 작동합니다.
- 멀티플렉싱됩니다(단일 TCP 연결을 통해 둘 이상의 요청을 보낼 수 있음).
- 서버는 클라이언트에 응답을 푸시할 수 있습니다.
- Round Trip Times를 줄여 클라이언트가 더 빠르게 로드할 수 있습니다.

HTTP/2는 다른 개선 사항을 도입합니다. 자세한 내용: [RFC7540](#).

HTTP/2는 ALPN 프로토콜을 요구하므로 우리의 사용자 지정 Indy 버전이 필요합니다.

구성 요소

- **TsgcHTTP2Client**: HTTP/2 프로토콜을 완전히 지원하는 클라이언트 구성 요소(sgcWebSockets 100% Pascal 코드, 외부 라이브러리 없음).
- **TsgcWebSocketHTTPServer**: HTTP/2 프로토콜을 완전히 지원하는 서버 구성 요소(sgcWebSockets 100% Pascal 코드, 외부 라이브러리 없음). 기본적으로 HTTP/2는 비활성화되어 있으며, HTTPOptions 속성을 사용하여 활성화하고 Enable = true로 설정할 수 있습니다.
- **TsgcWebSocketServer_HTTPAPI**: HTTP/2 프로토콜을 지원하는 서버 구성 요소(Microsoft 구현이며 Windows 2016+ 또는 Windows 10+가 필요함).
- **DataSnap Servers**: datasnap 서버도 HTTP/2 프로토콜을 지원할 수 있습니다.

APIs

- **Apple Push Notifications**: 서버 공급자로부터 사용자 대상 알림을 사용자의 디바이스로 푸시합니다.

TsgcHTTP2Client

*TsgcHTTP2Client*는 클라이언트 HTTP/2 구성 요소를 구현하며 HTTP/2 서버에 연결할 수 있습니다.

소개

TsgcHTTP2Client는 Client HTTP/2 구성 요소를 구현하며 HTTP/2 서버에 연결할 수 있습니다. 이 구성 요소를 구성하려면 아래 단계를 따르십시오:

1. **TsgcHTTP2Client** 구성 요소의 새 인스턴스를 생성하십시오.
2. 서버에 요청을 보내고 OnHTTP2Response 이벤트를 사용하여 응답을 처리하십시오. 예:

```
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);
}
```

참조

- 속성 — 구성 요소에서 사용할 수 있는 19개의 속성.
- 메서드 — 19개의 public 메서드.
- 이벤트 — 처리할 수 있는 13개의 이벤트입니다.

Guides

- **Requests**
 - [Request HTTP/2 Method](#)
 - [HTTP/2 Server Push](#)
 - [Download File](#)
 - [HTTP/2 Partial Responses](#)
 - [HTTP/2 Headers](#)
- **Connection**
 - [Client Close Connection](#)
 - [Client Keep Connection Active](#)
 - [HTTP/2 Reason Disconnection](#)
 - [Client Pending Requests](#)
- **Authentication**
 - [Client Authentication](#)
 - [HTTP/2 and OAuth2](#)
- **클래스**
 - [TsgcHTTP2ConnectionClient](#)
 - [TsgcHTTP2RequestProperty](#)

- [TsgcHTTP2ResponseProperty](#)

TsgcHTTP2Client | Request HTTP/2 Method

HTTP/2 Client는 블로킹 및 논블로킹 모드로 작동할 수 있습니다. 내부적으로 구성 요소는 보조 스레드에서 작동하고 요청은 비동기적으로 처리되지만, 요청을 호출하고 이 요청이 완료될 때까지 기다릴 수 있습니다.

클라이언트가 HTTP/2 서버에 HTML 페이지를 요청하는 방법과 두 가지 모드 모두에서 작동하는 방법에 대한 예제는 아래를 참조하십시오.

Asynchronous Mode

다음 url을 가져옵니다: <https://www.google.com> 그리고 클라이언트가 전체 응답을 수신하면 알림을 받습니다. **GETASYNC** 메서드를 호출한 후 프로세스가 계속되고 응답이 수신되면 OnHTTP2Response 이벤트가 호출됩니다.

```
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);
}
```

블로킹 모드

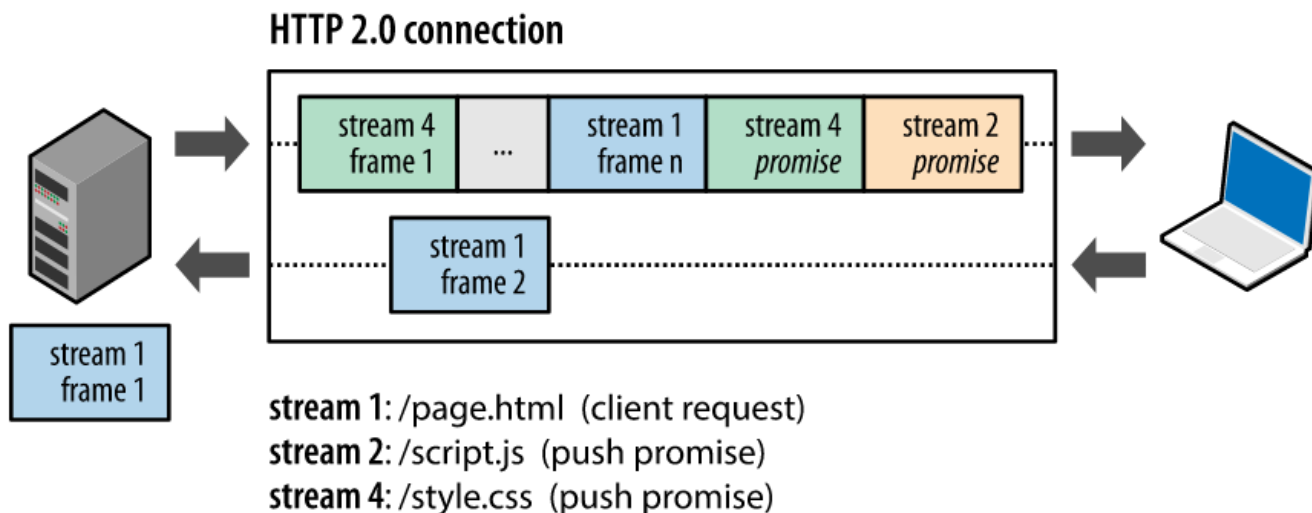
다음 url을 가져옵니다: <https://www.google.com> 그리고 클라이언트가 전체 응답을 수신할 때까지 기다립니다. **GET** 메서드를 호출한 후, 응답이 수신되거나 시간 초과에 도달할 때까지 프로세스가 대기합니다.

HTTP/2 클라이언트의 Response 속성을 사용하여 Raw Response 데이터에 액세스할 수 있습니다. 여기서 Raw Headers, Status 응답 코드, Charset 등에 액세스할 수 있습니다.

```
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));
```

요청 | HTTP/2 Server Push

Server Push는 단일 클라이언트 요청에 대해 서버가 여러 응답을 보낼 수 있는 기능입니다. 즉, 원래 요청에 대한 응답 외에도, 서버는 클라이언트가 각각을 명시적으로 요청하지 않고도 추가 리소스를 클라이언트에 푸시할 수 있습니다.



서버가 클라이언트에 PushPromise 메시지를 보낼 때마다 OnHTTP2PushPromise 이벤트가 호출됩니다. 클라이언트가 PushPromise를 수신하면 서버가 다음 패킷에서 이 리소스를 보낼 것임을 의미하므로, 클라이언트는 이를 수락하거나 거부할 수 있습니다.

```
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

클라이언트가 서버에서 파일을 요청하면 OnHTTP2Response 이벤트를 사용하여 스트림 응답을 로드하십시오.

Large File Downloads

대용량 파일(수백 MB 이상)을 다운로드할 때, 기본 60초 시간 초과로 전송이 중단되지 않고 완료되도록 **HTTP2Options.ReadTimeout**을 0(시간 초과 없음)으로 설정하십시오:

```
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

일반적으로 HTTP 요청을 전송하면 서버는 요청된 파일과 함께 응답을 전송합니다. 때때로 단일 응답을 전송하는 대신 서버는 스트림처럼 여러 응답을 전송할 수 있습니다. 이러한 경우 **OnHTTP2ResponseFragment** 이벤트를 사용하여 이러한 응답을 캡처하고 사용자에게 표시할 수 있습니다.

예제: <https://http2.golang.org/clockstream>에 요청을 보내면 서버가 매초마다 스트림 응답을 보냅니다.

```
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 | HTTP/2 Headers

TsgcHTTP2Client는 클라이언트가 연결될 때 서버로 전송되는 Headers의 사용자 지정을 허용합니다

Example: 이 HTTP 헤더 "Client: sgcWebSockets"를 추가해야 하는 경우

```
void OnHTTP2BeforeRequest(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection,
    ref TStringList *Headers)
{
    Headers->Add("Client: sgcWebSockets");
}
```

Request.CustomHeaders를 사용하여 사용자 정의 헤더를 추가할 수도 있습니다.

TsgcHTTP2Client | Client Close Connection

연결은 Active 속성을 사용하거나 Close/Disconnect 메서드를 사용하여 닫을 수 있습니다.

Active 속성

연결이 활성 상태이고 Active := False로 설정하면, 연결 끊김에 대해 서버에 어떤 메시지도 전송하지 않고 연결이 즉시 닫힙니다.

Disconnect

Disconnect 메서드(TsgcHTTP2Client 또는 TsgcHTTP2ConnectionClient에서)를 사용하여 소켓을 연결 해제할 수 있습니다.

Close

이 메서드는 연결이 닫힐 것임을 알리는 메시지를 서버에 보내며, 선택적으로 연결 해제 이유에 대한 정보를 보낼 수 있습니다. HTTP/2 연결을 깔끔하게 닫는 방법입니다.

Close 메서드는 TsgcHTTP2Client 또는 TsgcHTTP2ConnectionClient 객체에서 호출할 수 있습니다.

다음 오류 사유가 전송될 수 있습니다:

- 오류 없음
- 프로토콜 오류
- 내부 오류
- 흐름 제어 오류
- settings timeout
- stream closed
- frame size error
- refused stream
- cancel
- 압축 오류
- connect error
- enhance your calm
- 부적절한 보안
- required

TsgcHTTP2Client | Client Keep Connection Active

클라이언트가 서버에 연결되면 때때로 약한 신호, 연결 오류 등으로 인해 연결이 닫힐 수 있습니다. 연결을 활성 상태로 유지하는 데 도움이 되는 2개의 속성이 있습니다.

HeartBeat

HeartBeat 속성을 사용하면 **연결을 활성 상태로 유지**하기 위해 X 초마다 **Ping**을 보낼 수 있습니다. 일부 서버는 피어 간에 데이터가 교환되지 않으면 TCP 연결을 닫습니다. HeartBeat는 특정 간격마다 ping을 보내 이 문제를 해결합니다. 일반적으로 이것으로 연결을 활성 상태로 유지하기에 충분합니다.

HeartBeatType 속성을 사용하면 HeartBeat 작동 방식을 사용자 지정할 수 있습니다:

1. **hbtAlways**: Interval에 정의된 x초마다 ping을 보냅니다.
2. **hbtOnlyIfNoMsgRcvInterval**: Interval 속성에 정의된 최근 x초 동안 메시지가 수신되지 않은 경우에만 x초마다 ping을 보냅니다.

예제: 30초마다 ping 보내기

```
oClient = new TsgcHTTP2Client();
oClient->HeartBeat->Interval = 30;
oClient->HeartBeat->Enabled = true;
oClient->Active = true;
```

WatchDog

WatchDog이 활성화된 경우, 클라이언트가 연결 해제를 감지하면 WatchDog은 연결이 다시 활성화될 때까지 X초마다 재연결을 시도합니다.

예: 연결 해제 후 무제한 시도로 10초마다 재연결합니다.

```
oClient = new TsgcHTTP2Client();
oClient->WatchDog->Interval = 10;
oClient->WatchDog->Attempts = 0;
oClient->WatchDog->Enabled = true;
oClient->Active = true;
```

TsgcHTTP2Client | HTTP/2 Reason Disconnection

HTTP/2 서버는 여러 이유로 클라이언트를 연결 해제할 수 있으며, 서버가 연결 해제 이유를 클라이언트에게 알리려는 경우, 연결 해제에 대한 정보가 포함된 **GoAway** 메시지를 클라이언트에게 보냅니다.

OnHTTP2GoAway 이벤트를 사용하여 서버가 연결을 끊은 이유를 포착하십시오 (클라이언트가 연결을 닫으려는 경우 close 메서드를 사용하여 연결을 닫는 이유를 보낼 수 있습니다).

TsgcHTTP2GoAwayProperty 객체는 연결 해제에 대한 정보를 포함합니다

- **LastStreamId**: 서버가 처리한 마지막 스트림입니다.
- **ErrorCode**: 오류 코드를 식별하는 정수입니다.
- **ErrorDescription**: 오류에 대한 설명, 다음 중 하나:
 - 오류 없음
 - 프로토콜 오류
 - 내부 오류
 - 흐름 제어 오류
 - settings timeout
 - stream closed
 - frame size error
 - refused stream
 - cancel
 - 압축 오류
 - connect error
 - enhance your calm
 - 부적절한 보안
 - required
- **AdditionalDebugData**: 연결 해제에 대한 추가 정보를 제공하는 선택적 문자열입니다.

TsgcHTTP2Client | 클라이언트 대기 요청

클라이언트가 여러 요청을 보낼 때, 이는 보조 스레드에서 처리되는데, 때때로 어떤 이유로든 연결이 닫히고 여전히 보류 중인 요청이 있을 수 있습니다. 이러한 보류 중인 요청을 처리하려면 `OnHTTP2PendingRequests` 이벤트를 사용하십시오. 이 이벤트는 클라이언트가 연결 끊김을 감지하고 여전히 보류 중인 요청이 있을 때 호출됩니다. 이 이벤트에는 2개의 매개변수가 있습니다:

1. **Reconnect:** 기본적으로 비활성화되어 있습니다. `true`로 설정하면 클라이언트가 자동으로 재연결합니다.
2. **Clear:** 기본적으로 활성화됨, `false`로 설정하면 클라이언트가 다시 연결될 때 보류 중인 요청을 서버에 다시 전송하려고 시도합니다.

TsgcHTTP2Client | Client Authentication

HTTP/2 클라이언트는 2가지 인증 유형을 지원합니다: Basic Authentication 과 OAuth2 Authentication.

두 가지 인증 유형을 모두 처리하려면 **OnHTTP2Authorization** 이벤트를 사용하십시오.

Basic Authentication

서버가 Basic Authentication을 요청하는 헤더를 반환하면, OnHTTP2Authorization에 사용자 이름과 비밀번호를 설정하십시오.

```
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

서버가 Bearer Token Authentication을 요청하는 헤더를 반환하면 OnHTTP2Authorization에 token을 설정하십시오.

```
TsgcHTTP2Client *oClient = new TsgcHTTP2Client();
oClient->OnHTTP2Authorization = OnHTTP2AuthorizationEvent;
...
void OnHTTP2AuthorizationEvent(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection, const string AuthType)
{
    if (AuthType == "Bearer")
    {
        aToken = "bearer token";
    }
}
```

서드파티의 Bearer 값

다른 방법을 사용하여 Bearer Value를 얻어 이미 알고 있는 경우, 요청의 다음 속성을 사용하여 Bearer 값을 HTTP 헤더로 전달할 수 있습니다. HTTP 요청 메서드를 호출하기 전에 설정하기만 하면 됩니다:

```
TsgcHTTP2Client.Request.BearerAuthentication
= true

TsgcHTTP2Client.Request.BearerToken = "< value of the token >"
```

OAuth2

HTTP/2 클라이언트와 함께 [OAuth2 구성 요소](#)를 사용하려면 다음 문서를 읽으십시오.

TsgcHTTP2Client | HTTP/2 and OAuth2

OAuth2는 Google과 같은 여러 회사에서 사용하는 일반적인 권한 부여 방법입니다. Google API 중 하나를 사용하기 위해 Google 서버에 대해 인증하려면 일반적으로 OAuth2를 사용한 인증이 필요합니다.

sgcWebSockets는 HTTP/2 클라이언트에서 OAuth2를 지원합니다. [TsgcHTTP_OAuth2](#)의 인스턴스를 할당해야 하는 **Authentication.Token.OAuth**라는 속성이 있습니다.

GMail Google API에 연결하는 방법

Google API에 연결하려면, TsgcHTTP_OAuth2의 인스턴스를 생성하고 다음 데이터를 입력해야 합니다:

```
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';
```

OAuth2 클라이언트 구성 요소를 채운 후, TsgcHTTP2Client의 새 인스턴스를 생성하고 OAuth2 구성 요소를 HTTP/2 클라이언트에 Assign하십시오.

```
TsgcHTTP2Client1.Authentication.Token.OAuth := TsgcHTTP_OAuth1;
```

마지막으로 yourname@gmail.com 계정의 메시지 목록을 가져오기 위한 요청을 수행하십시오

```
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

TsgcHTTP2ConnectionClient는 클라이언트 HTTP/2 연결의 래퍼이며, 클라이언트 이벤트에서 이 객체에 접근할 수 있습니다.

Methods

- **Ping:** 연결을 유지하기 위해 서버에 ping을 보냅니다.
- **Close:** 연결을 끊는 이유에 대한 정보와 함께 서버에 메시지를 보냅니다.
- **Disconnect:** 서버에 정보 메시지를 보내지 않고 연결을 닫습니다.
- **HTTP/2 Methods:**

GET: GET 메서드는 지정된 리소스의 표현을 요청합니다. GET을 사용하는 요청은 데이터만 검색해야 합니다.

HEAD: HEAD 메서드는 GET 요청과 동일한 응답을 요청하지만, 응답 본문은 제외합니다.

POST: POST 메서드는 지정된 리소스에 엔티티를 제출하는 데 사용되며, 종종 서버의 상태 변경 또는 부작용을 일으킵니다.

PUT: PUT 메서드는 대상 리소스의 현재 모든 표현을 요청 페이로드로 대체합니다.

DELETE: HEAD 메서드는 GET 요청과 동일한 응답을 요청하지만 응답 본문은 포함하지 않습니다.

CONNECT: CONNECT 메서드는 대상 리소스로 식별되는 서버에 대한 터널을 설정합니다.

OPTIONS: OPTIONS 메서드는 대상 리소스에 대한 통신 옵션을 설명하는 데 사용됩니다.

TRACE: TRACE 메서드는 대상 리소스로의 경로를 따라 메시지 루프백 테스트를 수행합니다.

PATCH: PATCH 메서드는 리소스에 부분 수정을 적용하는 데 사용됩니다.

TsgcHTTP2RequestProperty

이 객체는 OnHTTP2Response 이벤트의 인수로 수신되며, 서버가 보낸 응답의 원래 요청을 알 수 있게 합니다.

Properties

- **Method:** HTTP/2 메서드(GET, POST...)를 식별합니다
- **URL:** 요청된 URL입니다.
- **Request:** 요청의 필드를 포함합니다.

TsgcHTTP2ResponseProperty

이 객체는 OnHTTP2Response 이벤트의 인수로 수신되며, 서버가 클라이언트에 보낸 응답을 알 수 있게 합니다.

Properties

- **Headers:** 서버로부터 수신된 원시 헤더 목록을 포함합니다.
- **Data:** 요청에 대한 응답으로 서버가 보낸 원시 본문을 포함합니다.
- **DataString:** Data를 문자열로 변환한 것입니다.
- **DataUTF8:** Data를 UTF8 문자열로 변환한 것입니다.
- **PushPromise:** 할당된 경우, 서버가 클라이언트에 보낸 PushPromise 객체를 포함합니다(이 응답 객체가 클라이언트에 의해 요청되지 않았음을 의미합니다).

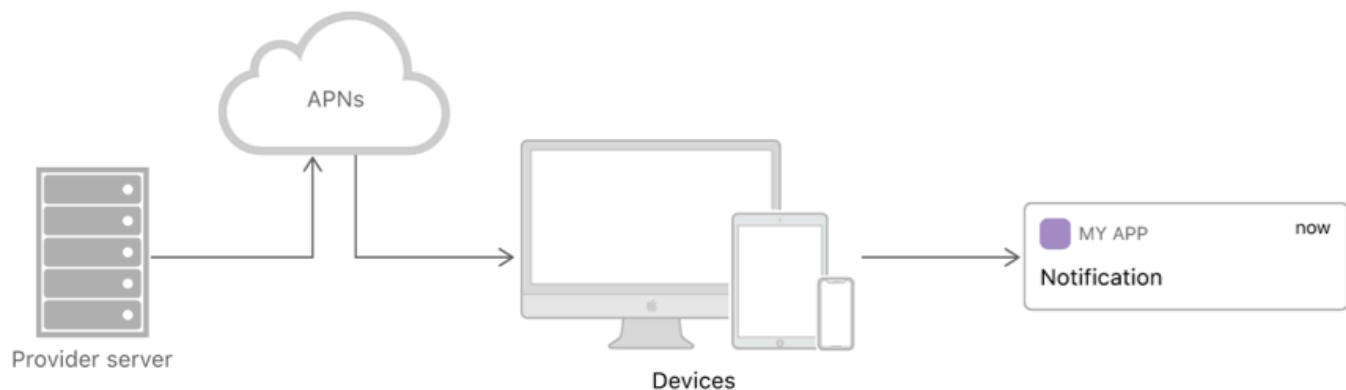
HTTP2 | Apple Push Notifications

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server

Apple은 Apple Push Notification Service(APNs)를 사용하여 Apple 디바이스에 푸시 알림을 보낼 수 있게 합니다.

장치에 알림을 전송하려면 공급자가 다음 정보를 포함하여 APNs에 HTTP/2 POST를 전송해야 합니다:

- 보내려는 정보가 담긴 JSON Payload입니다.
- 사용자의 디바이스를 식별하는 Device Token입니다.
- 알림을 전달하는 방법에 대한 일부 HTTP 헤더입니다.
- APNs에 대해 요청을 인증하기 위한 SSL 인증서 또는 JWT Token



알림을 보내는 데 필요한 것

Rad Studio를 사용하여 기기에 알림을 보내려면 다음 단계를 따라야 합니다.

- APNs로 APP 등록
- [원격 알림 생성](#)
- [Sending Notification Requests to APNs](#)
 - [APNs에 대한 토큰 기반 연결](#)
 - [APNs에 대한 인증서 기반 연결](#)

APN | Generate a Remote Notification APNs

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/generating_a_remote_notification

Apple 알림은 JSON 페이로드를 사용하여 알림 객체를 보냅니다. 페이로드의 최대 크기는 4096바이트입니다.

JSON 페이로드 샘플

간단한 알림 메시지

```
{
  "aps": {
    "alert": "Alert from sgcWebSockets!"
  }
}
```

제목과 부제목이 있는 알림입니다.

```
{
  "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

원격 알림 페이로드와 장치 토큰 정보를 Apple Push Notification service (APNs)로 보냅니다.

APNs에 연결하는 방법

서버 공급자와 다음 서버 중 하나 사이에 성공적인 연결을 설정하려면 HTTP/2 프로토콜과 최소 TLS 1.2 이상을 사용해야 합니다:

개발 서버: <https://api.sandbox.push.apple>

Production Server: <https://api.push.apple>

샘플 코드

TsgcHTTP2Client의 새 인스턴스를 생성하고 POST 메서드를 호출하여 APNs에 알림을 보냅니다.

```
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();
}
```

알림을 전송하려면 HTTP/2 프로토콜과 TLS 1.2 이상을 사용하여 APNs와 토큰 기반 또는 인증서 기반 신뢰를 설정해야 합니다.

- [Token-Based Connection to APNs](#)
- [APNs에 대한 인증서 기반 연결](#)

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

stateless 인증 토큰을 사용하여 Apple Push Notification service(APNs)로 통신을 보호하십시오.

먼저 Apple Developer Account에서 **Encryption Key**와 **Key ID**를 얻어야 합니다. 등록에 성공하면 Key ID가 담긴 10자 문자열과 인증 토큰 서명 키를 .p8 파일 확장자로 얻게 됩니다.

ES256을 알고리즘으로 사용하여 JWT를 생성하려면 sgcWebSockets **JWT Client**를 사용해야 합니다. 토큰은 모든 HTTP/2 요청에 대해 생성되어서는 안 되며, 토큰은 20분 전에 갱신되어서는 안 되고 60분 이후에 갱신되어서도 안 됩니다.

JWT 클라이언트 구성

다음 값으로 JWT 클라이언트를 구성하십시오:

- **JWTOptions.Header.Algorithm:** token을 암호화하는 데 사용한 암호화 알고리즘입니다. APNs는 ES256 알고리즘만 지원합니다.
- **JWTOptions.Header.kid:** 개발자 계정에서 얻은 10자리 Key ID입니다.
- **JWTOptions.Payload.iss:** 그 값은 회사의 앱을 개발하는 데 사용하는 10자리 Team ID입니다. 이 값은 개발자 계정에서 얻으십시오.
- **JWTOptions.Payload.iat:** "issued at" 시간으로, 그 값은 이 JSON 토큰이 생성된 시간을 나타냅니다. 값을 UTC 기준 Epoch 이후의 초 수로 지정하십시오. 값은 현재 시간으로부터 1시간을 초과해서는 안 됩니다.
- **JWTOptions.RefreshTokenAfter:** 값을 초 단위로 40분(60*40)으로 설정하십시오.

토큰 기반 연결을 사용하려면 앱의 번들 ID/앱 id 값(예: com.example.application)과 함께 **apns-topic**을 전송해야 합니다.

```
TsgcHTTP2Client *oHTTP = new TsgcHTTP2Client(NULL);
oHTTP->TLSOptions->IOHandler = iohOpenSSL;

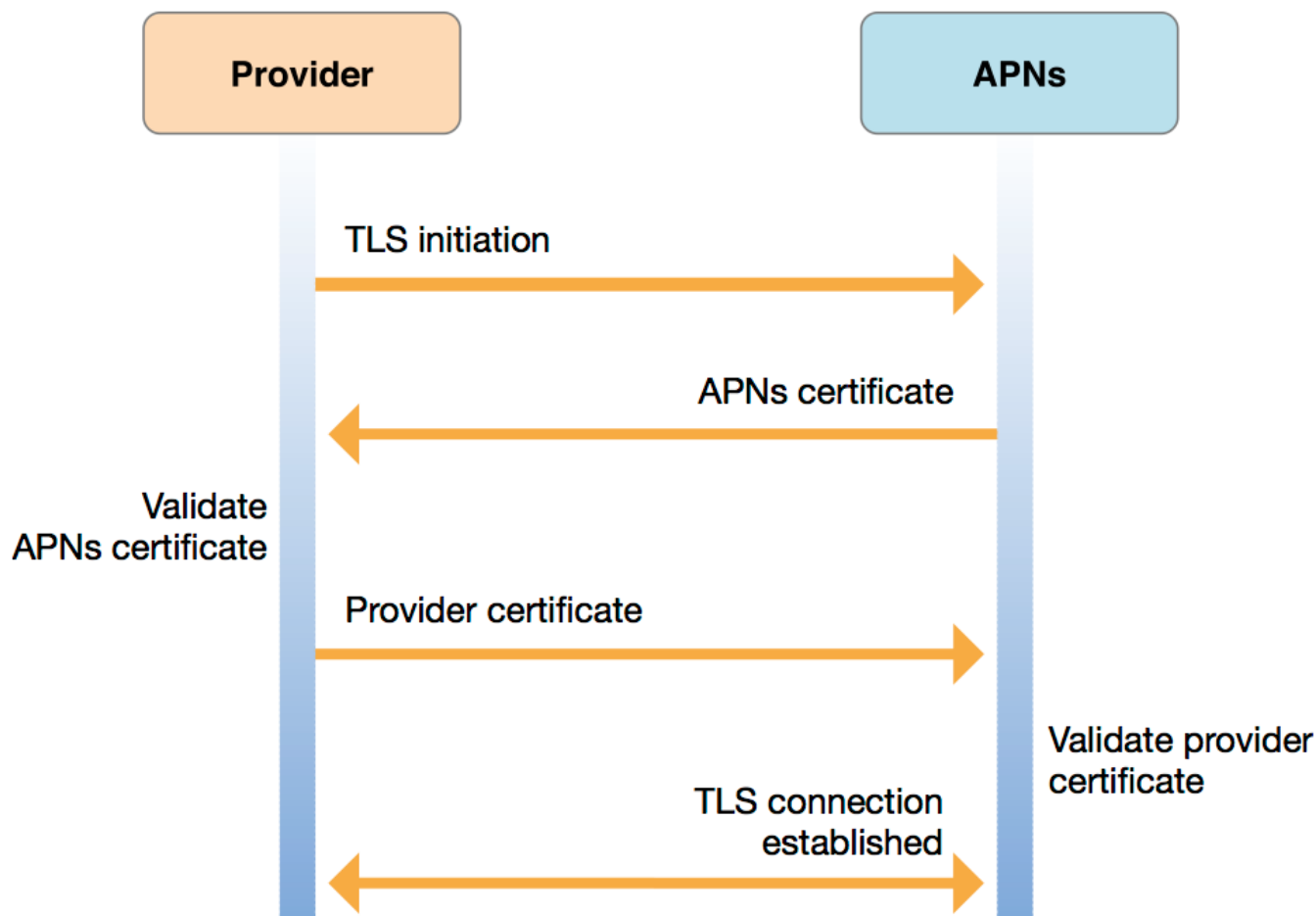
TsgcHTTP_JWT_Client *oJWT = new TsgcHTTP_JWT_Client(NULL);
oHTTP->Authentication->Token->JWT = oJWT;
oJWT->JWTOptions->Header->alg = jwtES256;
oJWT->JWTOptions->Header->kid = "apple key id";
oJWT->JWTOptions->Payload->iss = "issuer";
oJWT->JWTOptions->Payload->iat = StrToInt64(GetDateTimeUnix(Now, False));
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");
```


APNs에 대한 Certificate-Based Connection

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/establishing_a_certificate-based_connection_to_apns

Apple에서 얻은 인증서를 사용하여 Apple Push Notification service(APNs)로 통신을 보호할 수 있습니다.



먼저 개발자 계정에 들어가 Apple Push Notification 서비스에 대한 **새 인증서를 생성**하십시오

인증서를 다운로드한 후, `sgcWebSockets` HTTP/2 클라이언트는 2개의 보안 `IOHandler`를 사용할 수 있습니다(Windows 전용이며, 다른 `personality`의 경우 `openssl`만 지원됨).

- OpenSSL
- SChannel (Windows 전용)

OpenSSL

OpenSSL을 사용하는 경우 애플리케이션과 함께 OpenSSL 라이브러리를 배포해야 합니다. `TsgcHTTP2Client`로 인증서를 설정하기 전에, OpenSSL이 P12 인증서를 직접 가져오는 것을 허용하지 않으므로 이 인증서를 먼저 PEM 형식으로 변환해야 합니다.

단일 P12 인증서를 PEM 형식의 인증서와 비공개 키 파일로 변환하려면 다음 명령을 사용하십시오

PEM 인증서 파일 생성

```
openssl pkcs12 -in INFILE.p12 -out OUTFILE.crt -nokeys
```

Private Key 파일 생성

```
openssl pkcs12 -in INFILE.p12 -out OUTFILE.key -nodes -nocerts
```

PEM 형식의 인증서와 개인 키를 갖추면, [TsgcHTTP2Client](#)를 다음과 같이 구성할 수 있습니다.

```
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

SChannel을 사용하면 라이브러리를 배포할 필요가 없으며 Apple에서 다운로드한 인증서를 PEM 형식에서의 사전 변환 없이 직접 가져올 수 있습니다.

SChannel을 IOHandler로 사용할 때 UseLegacyCredentials 속성을 true로 설정하십시오.

```
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;
```

오류

"missing topic" 오류가 발생하면, 대부분 universal certificate(push notifications, voip 등에 사용할 수 있는 인증서)를 사용하고 있을 가능성이 높습니다. 이는 앱의 bundle ID/app id 값(예: com.example.application)으로 topic 이름을 설정해야 합니다. HTTP/2 클라이언트의 Request 속성에서 apns-topic 헤더를 올바른 값으로 설정하기만 하면 됩니다.

```
oHTTP->Request->CustomHeaders->Clear();
oHTTP->Request->CustomHeaders->Add("apns-topic: com.example.application");
```

HTTP/1

TsgcHTTP1Client는 TldHTTP indy 구성 요소에서 상속되고 일부 새 속성을 추가하는 비주얼이 아닌 구성 요소입니다. 이 구성 요소는 sgchttp 유닛에 있습니다.

TLSOptions

HTTP/1 프로토콜을 사용하여 보안 SSL/TLS 서버에 연결하는 방법을 구성할 수 있도록 합니다.

ALPNProtocols: 서버로 전송될 ALPN 프로토콜 목록입니다.

RootCertFile: 루트 인증서 파일의 경로입니다.

CertFile: 인증서 파일 경로입니다.

KeyFile: 인증서 키 파일의 경로입니다.

Password: 인증서가 비밀번호로 보호되어 있는 경우 여기에 설정하십시오.

VerifyCertificate: 인증서를 검증해야 하는 경우 이 속성을 활성화하십시오. SSL 검증을 사용자 지정하려면 **OnSSLVerifyPeer** 이벤트를 사용하십시오.

VerifyDepth: X.509 인증서에 대해 검증이 수행될 때 허용되는 최대 링크 수를 나타내는 Integer 속성입니다.

Version: 기본적으로 TLS 1.0을 사용합니다. 서버가 더 높은 TLS 버전을 요구하는 경우 여기에서 선택할 수 있습니다.

Proxy: 여기에서 Proxy Server를 통해 연결할지 여부를 정의할 수 있습니다. 다음 프록시 서버에 연결할 수 있습니다:

pxyHTTP: HTTP Proxy Server.

pxySocks4: SOCKS4 Proxy Server.

pxySocks4A: SOCKS4A 프록시 서버.

pxySocks5: SOCKS5 Proxy Server.

IOHandler: TLS를 사용하여 연결할 때 사용할 라이브러리를 선택합니다.

iohOpenSSL: OpenSSL 라이브러리를 사용하며 Indy 구성 요소의 기본값입니다. win32/win64 용 OpenSSL 라이브러리 배포가 필요합니다.

iohSChannel: Microsoft가 Windows용으로 구현한 보안 프로토콜인 Secure Channel을 사용합니다. OpenSSL 라이브러리를 배포할 필요가 없습니다. Windows 32/64비트에서만 작동합니다.

OpenSSL_Options: openssl 라이브러리의 구성입니다.

APIVersion: 어떤 OpenSSL API를 사용할지 정의할 수 있게 합니다.

oslAPI_1_0: API 1.0 OpenSSL을 사용하며, Indy가 지원하는 최신 버전입니다

oslAPI_1_1: API 1.1 OpenSSL을 사용하며, 당사의 사용자 정의 Indy 라이브러리가 필요하고 OpenSSL 1.1.1 라이브러리(TLS 1.3 지원 포함)를 사용할 수 있게 합니다.

oslAPI_3_0: API 3.0 OpenSSL을 사용하며, 사용자 지정 Indy 라이브러리가 필요하고 OpenSSL 3.0.0 라이브러리(TLS 1.3 지원 포함)를 사용할 수 있습니다.

LibPath: 여기에서 OpenSSL 라이브러리가 위치한 곳을 구성할 수 있습니다

oslpNone: 이것은 기본값입니다. OpenSSL 라이브러리는 바이너리와 동일한 폴더 또는 알려진 경로에 있어야 합니다.

oslpDefaultFolder: 모든 IDE personality에 대해 라이브러리가 위치해야 하는 OpenSSL 경로를 자동으로 설정합니다.

oslpCustomFolder: 이 옵션이 선택된 경우, LibPathCustom 속성에 전체 경로를 정의하십시오.

LibPathCustom: LibPath = oslpCustomFolder인 경우 OpenSSL 라이브러리가 위치한 전체 경로를 여기에 정의하십시오.

UnixSymLinks: Unix 시스템에서 SymLinks 로딩을 활성화하거나 비활성화합니다(기본적으로 활성화되어 있으며, OSX64에서는 예외):

oslsSymLinksDefault: 기본적으로 symlink는 OSX64를 제외하고 활성화됩니다(macOS Monterey 이상은 버전 없이 라이브러리를 로드하려고 할 때 실패합니다).

oslsSymLinksLoadFirst: 버전이 지정된 라이브러리를 로드하기 전에 symlink를 먼저 로드합니다.

oslsSymLinksLoad: 버전이 지정된 라이브러리를 로드하려고 시도한 후 심볼릭 링크를 로드합니다.

oslsSymLinksDontLoad: SymLinks를 로드하지 않습니다.

MinVersion: 보안 서버에 연결하기 위해 클라이언트가 사용할 최소 버전을 여기에 설정하십시오. 기본적으로 값은 `tlsUndefined`이며, 이는 최소 버전이 Version 속성에 설정된 것과 동일함을 의미합니다. 예: 클라이언트가 TLS 1.2 또는 TLS 1.3만 사용하여 연결하도록 설정하려면 다음 값을 설정하십시오.

`SSLOptions.Version := tls1_3;`

`SSLOptions.OpenSSL_Options.MinVersion := tls1_2;`

X509Checks: 추가 X509 인증서 유효성 검사를 활성화하려면 이 속성을 사용하십시오:

Mode: 어떤 옵션을 검증할지 선택합니다

oslx509chHostName: 호스트명 인증서를 검증합니다.

oslx509chIPAddress: 인증서의 ip 주소를 검증합니다.

HostName: 요청과 다른 경우 hostname을 설정합니다.

IPAddress: 요청과 다른 경우 ip 주소를 설정하십시오.

SChannel_Options: Windows Certificate Store의 인증서를 사용할 수 있게 합니다.

CertHash: 는 인증서 Hash입니다. powershell에서 dir 명령을 실행하여 인증서 Hash를 찾을 수 있습니다.

CipherList: 여 기 에 서 어 떤 Cipher 가 사 용 될 지 설 정 할 수 있 습 니 다(":"로 구 분). 예:

CALG_AES_256:CALG_AES_128

CertStoreName: 인증서가 저장되는 store 이름. 다음 중 하나를 선택하십시오:

scsnMY(기본값)

scsnCA

scsnRoot

scsnTrust

CertStorePath: 인증서가 저장된 저장소 경로입니다. 다음 중 하나를 선택하십시오:

scspStoreCurrentUser (기본값)

scspStoreLocalMachine

Log

Log 속성이 활성화되면 소켓 메시지를 지정된 로그 파일에 저장하므로 디버깅에 유용합니다.

LogOptions.FileName: 파일 이름의 전체 경로입니다.

Authentication

OAuth2 또는 **JWT**를 사용하여 인증할 수 있습니다.

Asynchronous Requests

기본적으로 HTTP1Client는 차단 요청을 사용하므로, HTTP 요청 메서드를 호출한 후 클라이언트는 서버로부터의 응답을 기다립니다. 또는 비동기 메서드를 사용하여 이러한 HTTP 요청을 보조 스레드에서 실행하여 요청이 호출되는 스레드의 차단을 방지할 수 있습니다. 다음 비동기 메서드가 구현되어 있습니다:

- GetAsync
- PostAsync
- PutAsync
- OptionsAsync
- DeleteAsync

이 메서드를 호출한 후, 응답을 기다리는 대신 코드는 다음 줄로 계속 진행되며, 응답은 **OnAsyncResponse** 이벤트를 사용하여 처리할 수 있습니다.

```
void __fastcall OnAsyncResultEvent(TObject* Sender, const TsgcHTTPAsyncRequest* aRequest, const TIdHTTPResponse*
```

비동기 요청을 처리하는 동안 오류가 있으면, **OnAsyncException** 이벤트에서 예외가 발생합니다.

Examples

HTTPs 서버에 **GET** 메서드를 요청하고 **TLS 1.2**를 사용합니다

```
TsgcHTTP1Client *oHTTP = new TsgcHTTP1Client();
try
{
    oHTTP->TLSOptions->Version = tls1_2;
    ShowMessage(oHTTP->Get("https://www.google.es"));
}
__finally
```

```
{
    oHTTP->Free();
}
```

openssl 1.1 및 **TLS 1.3**을 사용하여 HTTPs 서버에 **GET** 메서드를 요청합니다.

```
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();
}
```

비동기 POST 메서드를 요청하고 **OnAsyncResultEvent**를 사용하여 응답을 읽습니다.

```
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);
```

Windows용 SChannel을 사용하여 HTTPs 서버에 GET 메서드를 요청합니다.

```
TsgcHTTP1Client *oHTTP = new TsgcHTTP1Client();
try
{
    oHTTP->TLSOptions->IOHandler = iohSChannel;
    oHTTP->TLSOptions->Version = tls1_2;
    ShowMessage(oHTTP->Get("https://www.google.es"));
}
__finally
{
    oHTTP->Free();
}
```

데이터 이벤트를 얻으려면 **SSE** 메서드를 요청하십시오

```
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);
}
```

Events

OnSSEMessage

이 이벤트는 새 SSE 메시지가 수신될 때 호출됩니다.

OnSSLVerifyPeer

verify certificate가 활성화된 경우, 이 이벤트에서 서버 인증서를 확인하고 수락할지 여부를 결정할 수 있습니다.

OnSSLGetHandler

이 이벤트는 SSL 핸들러가 생성되기 전에 발생합니다. 여기에서 자체 SSL 핸들러를 생성하고 (TidServerIOHandlerSSLBase 또는 TidIOHandlerSSLBase에서 상속되어야 함) 필요한 속성을 설정할 수 있습니다.

OnSSLAfterCreateHandler

사용자 정의 SSL 객체가 생성되지 않은 경우, OpenSSL 핸들러를 사용하여 기본 객체가 생성됩니다. SSL 핸들러 속성에 액세스하여 필요한 경우 수정할 수 있습니다.

OnAsyncResult

이 이벤트는 비동기 메서드(GetAsync, PutAsync... 메서드 사용)를 요청한 후 호출됩니다. 요청 결과를 알기 위해 Response 매개변수를 사용하십시오.

OnAsyncException

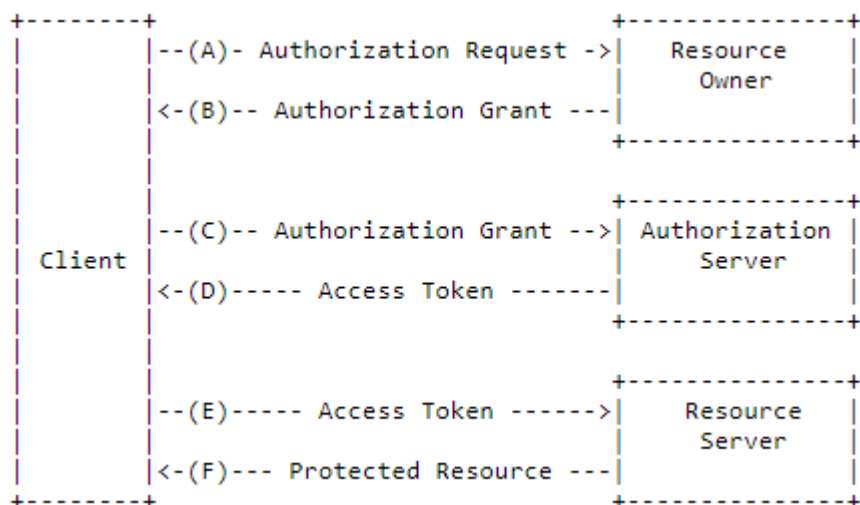
async 요청을 처리하는 동안 오류가 발생하면, 발생한 예외와 함께 이 이벤트가 호출됩니다.

HTTP | OAuth2

OAuth2를 사용하면 서드파티 애플리케이션이 리소스 소유자를 대신하거나 서드파티 애플리케이션이 자체적으로 액세스를 얻도록 허용하여 HTTP 서비스에 대한 제한된 액세스를 받을 수 있습니다. OAuth2 덕분에, 서비스 공급자와 소비자 애플리케이션은 안전한 방식으로 서로 상호 작용할 수 있습니다.

OAuth2에는 4가지 역할이 있습니다:

- **Resource Owner:** 사용자입니다.
- **Resource Server:** 보호된 리소스를 호스팅하고 access token을 기반으로 액세스를 제공하는 서버입니다.
- **Client:** 권한을 요청하는 외부 애플리케이션입니다.
- **Authorization Server:** 사용자를 인증한 후 access token을 발급합니다.



구성 요소

- **TsgcHTTP_OAuth2_Client:** OAuth2를 지원하는 클라이언트로, Google, Facebook과 같은 OAuth2 서버에 연결하여 인증을 요청할 수 있습니다...
- **TsgcHTTP_OAuth2_Server:** 는 OAuth2 프로토콜의 서버 구현으로, 서버의 리소스를 보호할 수 있게 합니다.
- **TsgcHTTP_OAuth2_Server_Provider:** 서버에서 외부 OAuth2 공급자(Azure AD, Google, Facebook...)를 구현할 수 있게 하여 사용자가 Azure, Google, Facebook... 사용자 자격 증명을 사용하여 로그인할 수 있습니다.

Server 및 **Client** OAuth2 구성 요소는 **PKCE** (Proof Key for Code Exchange)를 지원하며, 이는 CSRF 및 authorization code injection 공격을 방지하기 위한 Authorization Code 흐름의 확장입니다(RFC 7636).

TsgcHTTP_OAuth2_Client

클라이언트가 새 Access Token이 필요할 때, Authorization 서버로부터의 응답을 처리하기 위해 자동으로 HTTP 서버를 시작합니다.

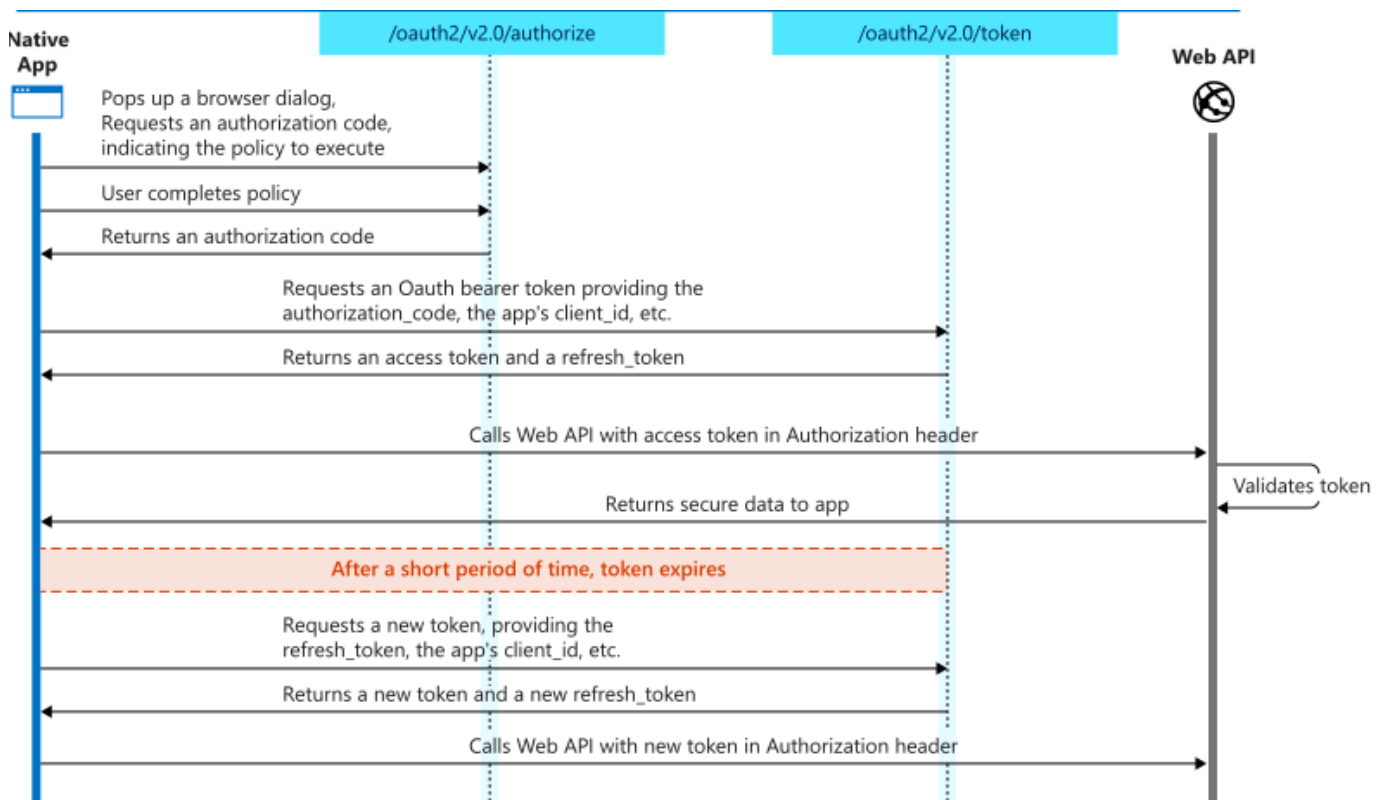
소개

이 구성 요소는 클라이언트와 다른 역할 간의 흐름을 처리할 수 있게 합니다. 기본적으로 Active := True로 설정하면 새 웹 브라우저를 열고 사용자에게 권한 부여를 요청합니다. 성공하면 권한 부여 서버가 애플리케이션에 토큰을 전송하고, 이 토큰이 처리되며, 이 토큰으로 클라이언트가 리소스 서버에 연결할 수 있습니다. 이 구성 요소는 권한 부여 서버 응답을 처리하는 간단한 HTTP 서버를 시작하고 HTTP 클라이언트를 사용하여 Access Token을 요청합니다.

GrantType

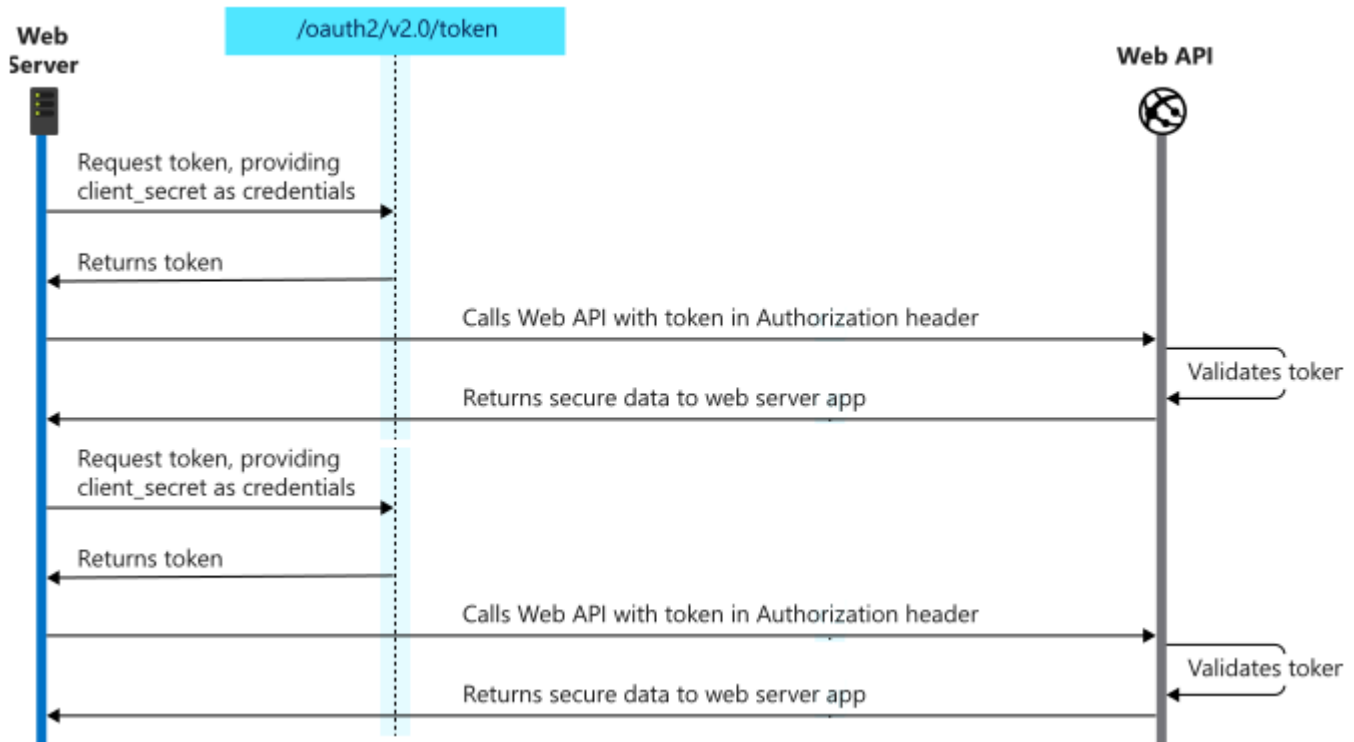
클라이언트는 다음 유형의 권한 부여를 지원합니다:

auth2Code: 단일 페이지 애플리케이션, 웹 애플리케이션, 네이티브 설치 애플리케이션을 포함한 대부분의 애플리케이션 유형에서 인증 및 권한 부여를 수행하는 데 사용됩니다. 이 흐름을 통해 앱은 보안 리소스에 액세스하는 데 사용할 수 있는 access_token, 추가 access_token을 얻기 위한 갱신 토큰, 그리고 로그인한 사용자의 ID 토큰을 안전하게 획득할 수 있습니다.



auth2CodePKCE: PKCE가 활성화된 auth2Code와 동일한 인증 플로우입니다. PKCE(Proof Key for Code Exchange)는 OAuth 2.0의 보안 확장으로, 네이티브 및 단일 페이지 애플리케이션의 권한 부여 플로우 보안을 강화하도록 설계되었습니다. 특히 권한 부여 코드가 전송 중에 가로채기에 노출될 수 있는 공개 클라이언트에서 가로채기 공격의 위험을 완화합니다. 일반적으로 이 옵션은 네이티브 및 모바일 앱에서 사용됩니다.

auth2ClientCredentials: 이 유형의 grant는 사용자와의 즉각적인 상호 작용 없이 백그라운드에서 실행되어야 하는 서버 간 상호 작용에 일반적으로 사용됩니다. 이러한 유형의 애플리케이션은 종종 데몬 또는 서비스 계정이라고 합니다.



auth2DeviceCode: 브라우저가 없거나 입력 기능이 제한된 스마트 TV, 미디어 콘솔, IoT 장치와 같은 입력 제약 장치를 위한 Device Authorization Grant(RFC 8628)입니다. 장치는 사용자 코드와 확인 URI를 표시합니다. 사용자는 보조 장치(휴대폰 또는 컴퓨터)에서 URI를 방문하고 코드를 입력하여 권한을 부여합니다.

LocalServerOptions

클라이언트가 새 Access Token이 필요할 때, Authorization 서버로부터 응답을 처리하기 위해 자동으로 HTTP 서버를 시작합니다. 이 서버는 사용자에게 투명하며 일반적으로 localhost에서 작동합니다. 기본적으로 포트 8080을 사용하지만 필요한 경우 변경할 수 있습니다.

- **IP:** 서버가 수신 대기하는 IP입니다(예: 127.0.0.1).
- **Port:** 기본값 8080. GrantType = auth2CodePKCE를 사용할 때(데스크톱 및 네이티브 애플리케이션용) 포트 값을 0으로 설정하면 서버가 임의의 포트를 선택합니다.
- **RedirectURL:** (선택 사항) redirect URL을 사용자 정의할 수 있습니다. 예: `http://localhost:8080/oauth/`.
- **SSL:** 로컬 서버가 보안 포트에서 실행되는 경우 이 속성을 활성화하십시오(*Professional 및 Enterprise 에디션에서만 지원됨).
- **SSLOptions:** 서버의 SSL 속성을 사용자 지정할 수 있습니다(*Professional 및 Enterprise 에디션에서만 지원됨).
- **LogOptions:** HTTP 내부 서버가 수신하고 전송한 요청/응답의 로그를 저장할 수 있게 합니다(*Professional 및 Enterprise Edition 전용).
 - **Enabled:** 파일 로그를 활성화하려면 True로 설정하십시오.
 - **FileName:** 로그 파일을 저장할 파일 이름을 설정하십시오.

AuthorizationServerOptions

여기서 Authorization과 Access Token의 URL을 설정해야 합니다. 일반적으로 이것들은 API 사양에 제공됩니다. Scope는 클라이언트가 요청한 모든 범위의 목록입니다. 예:

- **AuthURL:** `https://accounts.google.com/o/oauth2/auth`
- **TokenURL:** `https://accounts.google.com/o/oauth2/token`
- **Scope:** `https://mail.google.com/`
- **RevocationURL:** 토큰 취소 엔드포인트의 URL입니다(Revoke 메서드에 필요). 예: `https://accounts.google.com/o/oauth2/revoke`
- **IntrospectionURL:** 토큰 인트로스펙션 엔드포인트의 URL입니다(Introspect 메서드에 필요). 예: `https://accounts.google.com/o/oauth2/introspect`

OAuth2Options

ClientId는 서버에 클라이언트의 식별 정보를 알리는 필수 필드입니다. ClientId를 얻는 방법을 알려면 API 사양을 확인하십시오. client secret도 마찬가지입니다.

때때로 서버는 Basic Authentication을 사용하여 연결하기 위해 사용자와 비밀번호를 요구합니다. 이 경우 Username/Password 필드에서 이를 설정할 수 있습니다. 예제:

- **GrantType:** 권한 부여 흐름 유형
 - **auth2Code:** 웹 서버와 같은 신뢰할 수 있는 앱입니다.
 - **auth2CodePKCE:** 신뢰할 수 없는 네이티브 또는 모바일 앱입니다.
 - **auth2ClientCredentials:** 사용자 상호 작용이 없는 자동화된 앱입니다.
 - **auth2ResourceOwnerPassword:** 애플리케이션이 사용자의 비밀번호를 직접 처리하여 사용자를 로그인시킬 수 있게 합니다
 - **auth2DeviceCode:** 브라우저가 없거나 입력 기능이 제한된 입력 제약 장치를 위한 Device Authorization Grant(RFC 8628)입니다.
- **ClientId:** 180803918307-eqjtm20gqfhcs6gjk1bbrreng022mqqc.apps.googleusercontent.com
- **ClientSecret:** _by1iYYrvVHxC2Z8TbtNEYJN
- **Username:**
- **Password:**

HTTPClientOptions

여기서 새 토큰을 요청하기 위해 HTTP Server에 연결할 때 클라이언트 옵션을 사용자 지정할 수 있습니다.

TLSOptions: TLS가 활성화된 경우, 여기에서 일부 TLS 속성을 사용자 정의할 수 있습니다.

ALPNProtocols: 서버로 전송될 ALPN 프로토콜 목록입니다.

RootCertFile: 루트 인증서 파일의 경로입니다.

CertFile: 인증서 파일 경로입니다.

KeyFile: 인증서 키 파일의 경로입니다.

Password: 인증서가 비밀번호로 보호된 경우 여기에 설정하십시오.

VerifyCertificate: 인증서를 검증해야 하는 경우 이 속성을 활성화하십시오.

VerifyDepth: X.509 인증서에 대해 검증이 수행될 때 허용되는 최대 링크 수를 나타내는 Integer 속성입니다.

Version: 기본적으로 TLS 1.0을 사용하며, 서버가 더 높은 TLS 버전을 요구하는 경우 여기에서 선택할 수 있습니다.

IOHandler: TLS를 사용하여 연결할 때 사용할 라이브러리를 선택합니다.

iohOpenSSL: OpenSSL 라이브러리를 사용하며 Indy 구성 요소의 기본값입니다. win32/win64용 openssl 라이브러리를 배포해야 합니다.

iohSChannel: Microsoft가 Windows용으로 구현한 보안 프로토콜인 Secure Channel을 사용하며, openssl 라이브러리를 배포할 필요가 없습니다. Windows 32/64비트에서만 작동합니다.

OpenSSL_Options: 사용할 OpenSSL API를 정의할 수 있습니다.

APIVersion: 어떤 OpenSSL API를 사용할지 정의할 수 있게 합니다.

oslAPI_1_0: API 1.0 OpenSSL을 사용하며, Indy가 지원하는 최신 버전입니다

oslAPI_1_1: API 1.1 OpenSSL을 사용하며, 당사의 사용자 정의 Indy 라이브러리가 필요하고 OpenSSL 1.1.1 라이브러리(TLS 1.3 지원 포함)를 사용할 수 있게 합니다.

oslAPI_3_0: API 3.0 OpenSSL을 사용하며, 사용자 지정 Indy 라이브러리가 필요하고 OpenSSL 3.0.0 라이브러리(TLS 1.3 지원 포함)를 사용할 수 있습니다.

LibPath: 여기에서 openssl 라이브러리가 위치한 곳을 구성할 수 있습니다.

oslpNone: 이것이 기본값이며, openssl 라이브러리는 바이너리가 있는 동일한 폴더 또는 알려진 경로에 있어야 합니다.

oslpDefaultFolder: 모든 IDE 퍼스널리티에 대해 라이브러리가 위치해야 하는 openssl 경로를 자동으로 설정합니다.

oslpCustomFolder: 이 옵션이 선택된 경우, LibPathCustom 속성에 전체 경로를 정의하십시오.

LibPathCustom: LibPath = oslpCustomFolder일 때 openssl 라이브러리가 있는 전체 경로를 여기에 정의하십시오.

UnixSymLinks: Unix 시스템에서 SymLinks 로딩을 활성화하거나 비활성화합니다(기본적으로 활성화되어 있으며, OSX64에서는 예외):

oslsSymLinksDefault: 기본적으로 활성화되지만 OSX64에서는 예외입니다(MacOS Monterey 이후 버전 없이 라이브러리를 로드하려고 하면 실패함).

oslsSymLinksLoadFirst: 버전 라이브러리를 로드하기 전에 SymLink를 로드하고 먼저 시도합니다.

oslsSymLinksLoad: 버전 라이브러리를 로드하려고 시도한 후 SymLinks를 로드합니다.

oslsSymLinksDontLoad: SymLinks를 로드하지 않습니다.

SChannel_Options: Windows Certificate Store의 인증서를 사용할 수 있게 합니다.

CertHash: 는 인증서 Hash입니다. powershell에서 dir 명령을 실행하여 인증서 Hash를 찾을 수 있습니다.

CertStoreName: 인증서가 저장된 스토어 이름. 아래 중 하나를 선택하십시오:

scsnMY(기본값)

scsnCA

scsnRoot

scsnTrust

CertStorePath: 인증서가 저장된 저장소 경로입니다. 아래에서 하나를 선택하십시오:

scspStoreCurrentUser (기본값)
scspStoreLocalMachine

LogOptions: 파일 이름이 설정되면 HTTP 클라이언트의 HTTP 요청/응답 로그를 저장합니다.

참조

- 속성 — 구성 요소에서 사용할 수 있는 6개의 속성.
- 메서드 — 7개의 public 메서드.
- Events — 처리할 수 있는 19개의 이벤트.

OAuth2 Client for Web Applications

OAuth2 Client가 웹 애플리케이션에 연결하여 권한 부여된 토큰을 얻어야 할 때, (권한 부여된 토큰을 얻는 데 사용되는) Local Server를 웹 애플리케이션 매개변수로 구성해야 합니다. 다음 **LocalServerOptions** 속성을 설정하십시오:

- **IP:** OAuth2 구성에서 이전에 구성된 Address IP입니다.
- **Port:** OAuth2 구성에서 이전에 구성된 수신 포트입니다.
- **SSL:** 서버가 보안 연결을 사용하는 경우 이 옵션을 활성화하십시오.

OAuth2 Client for Desktop Applications

OAuth2 클라이언트가 Desktop 또는 네이티브 애플리케이션인 경우, Local Server(권한 부여된 토큰을 얻는 데 사용)는 로컬 IP 주소(예: 127.0.0.1)에서 수신할 수 있으며 포트는 무작위로 선택할 수 있습니다. 따라서 **LocalServerOptions** 는 다음과 같이 구성해야 합니다

- **IP:** 127.0.0.1
- **Port:** 0으로 설정하면 서버가 자동으로 무작위 포트를 선택합니다.

Desktop 또는 Native 애플리케이션용 OAuth2 Client를 생성할 때, 추가 보안을 추가하려면 **OAuth2Options.GrantType** 를 **auth2CodePKCE** 로 설정하십시오.

OAuth2 | TsgcHTTP_OAuth2_Client_Google

이 구성 요소를 사용하면 손쉽게 Google 계정으로 로그인할 수 있습니다.

구성

이 모듈은 먼저 Google 계정에서 **OAuth2 애플리케이션**을 구성 해야 하며, 구성되면 애플리케이션에 몇 줄만 추가하여 사용자가 모든 Google 계정으로 로그인할 수 있게 합니다.

Google의 응답을 읽는 데 사용되는 **로컬 서버**는 기본적으로 **IP 주소 127.0.0.1** 과 **port 0** (무작위 포트)에서 수신 대기합니다. 따라서 Google 애플리케이션에서 CallBack URL을 구성해야 합니다. 물론 IP 주소와 포트를 수정할 수 있습니다.

Google 계정에서 OAuth2 애플리케이션을 구성한 후, **TsgcHTTP_OAuth2_Client_Google**의 인스턴스를 생성하고 **Authenticate** 메서드를 호출하여 **Client_Id**와 **Client_Secret**을 매개변수로 전달하기만 하면 됩니다. 이 **메서드**는 사용자가 **성공적으로 로그인**할 때까지 (기본적으로 최대 60초) 대기합니다. 사용자가 인증되었는지 여부, 이름, Id... 및 사용자 프로필의 추가 데이터를 확인할 수 있는 객체를 반환합니다.

예제

```
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

이 구성 요소를 사용하면 쉬운 방법으로 Microsoft 계정으로 로그인할 수 있습니다.

구성

모듈은 먼저 Microsoft 계정에서 **OAuth2 애플리케이션**을 구성해야 합니다. 구성이 완료되면 애플리케이션에 몇 줄을 추가하여 사용자가 모든 Microsoft 계정으로 로그인할 수 있게 하면 됩니다.

Microsoft로부터 응답을 읽는 데 사용되는 **Local Server**는 기본적으로 **IP 주소 127.0.0.1** 및 **포트 8080**에서 수신 대기하고 SSL을 사용합니다. 따라서 Microsoft 애플리케이션에서 CallBack URL을 **https://localhost:8080**으로 구성해야 합니다 (Microsoft는 로컬 IP 주소로 localhost만 허용합니다). 물론 IP 주소와 포트를 수정할 수 있습니다.

Microsoft 계정에서 OAuth2 Application을 구성한 후, **TsgcHTTP_OAuth2_Client_Microsoft**의 인스턴스를 만들고 **Authenticate** 메서드를 호출하여 매개변수로 **TenantId**, **Client_Id** 및 **Client_Secret**를 전달하십시오. 이 메서드는 대기합니다 (기본적으로 최대 60초) 사용자가 **성공적으로 로그인**할 때까지. 사용자가 인증되었는지 여부, Name, Id 등 사용자 프로파일의 추가 데이터를 확인할 수 있는 객체를 반환합니다.

예제

```
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)

개요

Authorization Code grant은 가장 일반적인 OAuth2 흐름입니다. 클라이언트가 client secret을 안전하게 저장할 수 있는 웹 애플리케이션, 데스크톱 애플리케이션 및 모바일 애플리케이션을 위해 설계되었습니다. 이 흐름은 사용자를 권한 부여 서버로 리디렉션하는 것을 포함하며, 거기서 사용자가 인증하고 권한을 부여합니다. 그런 다음 서버는 권한 부여 코드를 redirect URI로 반환하고, 클라이언트는 해당 코드를 액세스 토큰으로 교환합니다.

흐름

- 클라이언트가 client_id, redirect_uri, response_type=code, scope 및 state 매개변수와 함께 사용자를 **AuthURL**로 리디렉션합니다.
- 사용자가 권한 부여 서버로 인증하고 클라이언트에 권한을 부여합니다.
- 권한 부여 서버가 code 매개변수와 함께 사용자를 redirect URI로 다시 리디렉션합니다.
- 클라이언트는 권한 부여 코드를 액세스 토큰으로 교환하기 위해 client_id, client_secret 및 redirect_uri와 함께 **TokenURL**로 POST합니다.
- 서버는 access_token, token_type, expires_in, 그리고 선택적으로 refresh_token을 반환합니다.

구성

Property	설명
<code>OAuth2Options.GrantType</code>	auth2Code 로 설정하십시오.
<code>OAuth2Options.ClientId</code>	권한 부여 서버가 발급한 클라이언트 식별자입니다.
<code>OAuth2Options.ClientSecret</code>	권한 부여 서버가 발급한 클라이언트 시크릿입니다.
<code>AuthorizationServerOptions.AuthURL</code>	사용자가 인증하기 위해 리디렉션되는 authorization 엔드포인트 URL입니다.
<code>AuthorizationServerOptions.TokenURL</code>	authorization code가 access token으로 교환되는 토큰 엔드포인트 URL입니다.
<code>AuthorizationServerOptions.Scope</code>	액세스 요청의 scope입니다(예: openid, profile, email).
<code>LocalServerOptions.IP</code>	로컬 리디렉션 서버의 IP 주소입니다(예: 127.0.0.1).
<code>LocalServerOptions.Port</code>	로컬 리디렉션 서버의 포트(예: 8080)입니다.

Events

Event	설명
<code>OnBeforeAuthorizeCode</code>	클라이언트가 사용자를 authorization 엔드포인트로 리디렉션하기 전에 발생합니다. authorization 요청을 사용자 정의할 수 있도록 합니다.

<code>OnAfterAuthorizeCode</code>	서버로부터 authorization code가 수신된 후에 발생합니다.
<code>OnErrorAuthorizeCode</code>	권한 부여 코드 단계 중에 오류가 발생하면 발생합니다.
<code>OnBeforeAccessToken</code>	클라이언트가 권한 부여 코드를 액세스 토큰으로 교환하기 전에 발생합니다.
<code>OnAfterAccessToken</code>	엑세스 토큰이 수신된 후 발생합니다. 이 이벤트를 사용하여 토큰 값을 검색하십시오.
<code>OnErrorAccessToken</code>	토큰 교환 단계 중에 오류가 발생할 때 발생합니다.

예제

```
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

엑세스 토큰이 만료되면 갱신 토큰을 사용하여 사용자 상호 작용 없이 새 토큰을 얻을 수 있습니다. 갱신 토큰 값으로 Refresh 메서드를 호출하십시오:

`OnAfterAccessToken` 이벤트가 새 액세스 토큰과 함께 발생합니다. 갱신이 실패하면 `OnErrorAccessToken` 이벤트가 발생합니다.

사용 시기

다음의 경우 Authorization Code 그랜트를 사용하십시오:

- 클라이언트 시크릿을 안전하게 저장할 수 있는 서버 측 백엔드가 있는 웹 애플리케이션.
- 서버 측 렌더링 애플리케이션(예: ASP.NET, PHP, Java)입니다.
- client secret을 합리적으로 보호할 수 있는 데스크톱 애플리케이션입니다.

클라이언트 시크릿을 안전하게 저장할 수 없는 public 클라이언트(네이티브 앱, SPA)의 경우 대신 [Authorization Code with PKCE](#)를 사용하는 것을 고려하십시오.

OAuth2 | Authorization Code with PKCE (RFC 7636)

개요

Authorization Code with PKCE(Proof Key for Code Exchange) 권한 부여는 표준 Authorization Code 흐름을 추가 보안 계층으로 확장합니다. 이는 클라이언트 시크릿을 안전하게 저장할 수 없는 네이티브 애플리케이션, 모바일 애플리케이션 및 단일 페이지 애플리케이션(SPA)을 위해 설계되었습니다. 클라이언트 시크릿에 의존하는 대신, 클라이언트는 권한 부여 코드를 교환하는 엔터티가 흐름을 시작한 엔터티와 동일함을 증명하는 암호화 코드 검증자(code verifier)와 챌린지를 생성합니다.

PKCE 작동 방식

1. 클라이언트가 무작위 **code_verifier**(암호학적으로 무작위인 32바이트 데이터, Base64URL 인코딩됨)를 생성합니다.
2. 클라이언트는 **code_challenge**를 code_verifier의 SHA-256 해시로 계산하여 Base64URL로 인코딩합니다.
3. 클라이언트는 권한 부여 요청과 함께 code_challenge와 code_challenge_method=S256을 보냅니다.
4. 사용자 권한 부여 후, 클라이언트는 토큰 교환 요청과 함께 원본 code_verifier를 보냅니다.
5. 권한 부여 서버는 토큰을 발급하기 전에 $\text{SHA256}(\text{code\_verifier}) == \text{code\_challenge}$ 를 검증합니다.

이는 권한 부여 코드 가로채기 공격을 방지합니다. 코드를 가로챈 공격자가 원래 code_verifier 없이는 이를 교환할 수 없기 때문입니다.

구성

Property	설명
<code>OAuth2Options.GrantType</code>	auth2CodePKCE 로 설정하십시오.
<code>OAuth2Options.ClientId</code>	권한 부여 서버가 발급한 클라이언트 식별자입니다.
<code>OAuth2Options.ClientSecret</code>	선택 사항. 일부 공급자는 PKCE를 사용하더라도 이를 요구하고, 다른 공급자는 요구하지 않습니다.
<code>AuthorizationServerOptions.AuthURL</code>	사용자가 인증하기 위해 리디렉션되는 authorization 엔드포인트 URL입니다.
<code>AuthorizationServerOptions.TokenURL</code>	authorization code가 access token으로 교환되는 토큰 엔드포인트 URL입니다.
<code>AuthorizationServerOptions.Scope</code>	접근 요청의 스코프입니다.
<code>LocalServerOptions.IP</code>	로컬 리디렉션 서버의 IP 주소입니다(예: 127.0.0.1).
<code>LocalServerOptions.Port</code>	로컬 redirect 서버의 포트. 임의의 사용 가능한 포트는 0 으로 설정하십시오.

예제

```
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

LocalServerOptions.Port가 0으로 설정되면, 구성 요소가 자동으로 사용 가능한 임의의 포트를 선택합니다. 포트 충돌을 방지 하므로 데스크톱 및 모바일 애플리케이션에 권장됩니다. 선택된 포트는 권한 부여 서버로 보내는 redirect URI에 포함됩니다.

Token Refresh

PKCE 흐름은 일반적으로 갱신 토큰을 반환합니다. 새 액세스 토큰을 얻으려면 Refresh 메서드를 사용하십시오:

사용 시기

다음의 경우 PKCE를 사용한 Authorization Code grant를 사용하십시오:

- 데스크톱 애플리케이션(네이티브 Windows, macOS, Linux).
- 모바일 애플리케이션(iOS, Android).
- 브라우저에서 실행되는 single-page applications (SPAs)입니다.
- client secret를 안전하게 저장할 수 없는 모든 public 클라이언트입니다.
- 기밀 클라이언트의 경우에도 표준 [Authorization Code](#) grant보다 더 안전한 대안으로 사용됩니다.

OAuth2 | Client Credentials Grant (RFC 6749)

개요

Client Credentials 그랜트는 사용자 상호 작용이 필요하지 않은 머신 대 머신(M2M) 인증에 사용됩니다. 클라이언트는 자체 자격 증명(client_id 및 client_secret)을 사용하여 권한 부여 서버에 직접 인증하고 액세스 토큰을 받습니다. 권한 부여 코드 단계나 사용자 로그인이 없습니다.

이 그랜트 유형은 클라이언트가 사용자를 대신해서가 아니라 자신을 대신하여 작동할 때 적합합니다.

흐름

1. 클라이언트가 토큰 엔드포인트에 client_id, client_secret, grant_type=client_credentials 및 scope를 POST합니다.
2. 권한 부여 서버가 클라이언트 자격 증명을 검증합니다.
3. 서버는 access_token, token_type, expires_in을 반환합니다.

클라이언트가 자격 증명을 사용하여 단순히 새 액세스 토큰을 요청할 수 있으므로, 이 grant 유형에서는 일반적으로 갱신 토큰이 발급되지 않는다는 점에 유의하십시오.

구성

Property	설명
<code>OAuth2Options.GrantType</code>	auth2ClientCredentials 로 설정하십시오.
<code>OAuth2Options.ClientId</code>	권한 부여 서버가 발급한 클라이언트 식별자입니다.
<code>OAuth2Options.ClientSecret</code>	권한 부여 서버가 발급한 클라이언트 시크릿입니다.
<code>AuthorizationServerOptions.TokenURL</code>	토큰 엔드포인트 URL입니다. 사용자 권한 부여 단계가 없으므로 AuthURL이 필요하지 않습니다.
<code>AuthorizationServerOptions.Scope</code>	접근 요청의 스코프입니다.

예제

```
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();
```

사용 시기

다음 용도로 Client Credentials 부여를 사용하십시오:

- 사용자 상호 작용 없이 실행되는 백엔드 서비스 및 데몬.
- 서버 간 API 통신입니다.
- 예약된 작업, 배치 프로세스 및 cron 작업.
- 마이크로서비스 간 인증입니다.

- 애플리케이션이 사용자를 대신해서가 아니라 자신을 대신하여 작동하는 모든 시나리오입니다.

OAuth2 | Resource Owner Password Credentials Grant (RFC 6749)

개요

Resource Owner Password Credentials(ROPC) 그랜트는 클라이언트가 사용자의 사용자 이름과 비밀번호를 직접 수집하고 이를 액세스 토큰으로 교환할 수 있게 합니다. 사용자의 자격 증명은 토큰 엔드포인트로 전송되고, 유효한 경우 서버가 액세스 토큰을 반환합니다.

이 부여 유형은 리소스 소유자와 클라이언트 사이에 높은 수준의 신뢰가 있을 때만 사용해야 합니다. 예를 들어 클라이언트가 권한 부여 서버를 운영하는 동일한 조직에서 개발한 자사 애플리케이션인 경우입니다.

보안 경고

이 **grant** 유형은 다른 **OAuth2** 흐름보다 덜 안전한 것으로 간주됩니다. 사용자의 자격 증명을 클라이언트 애플리케이션에 직접 노출하므로 위임된 권한 부여 원칙을 위반합니다. OAuth 2.1 사양은 이 grant 유형을 더 이상 사용하지 않습니다. 가능하면 대신 **PKCE**를 사용한 **Authorization Code**를 사용하십시오.

흐름

1. 사용자가 클라이언트 애플리케이션에 사용자 이름과 비밀번호를 직접 제공합니다.
2. 클라이언트가 username, password, client_id, client_secret, grant_type=password 및 scope를 토큰 엔드포인트에 POST합니다.
3. 권한 부여 서버가 자격 증명을 검증합니다.
4. 서버는 access_token, token_type, expires_in, 그리고 선택적으로 refresh_token을 반환합니다.

구성

Property	설명
<code>OAuth2Options.GrantType</code>	auth2ResourceOwnerPassword 로 설정하십시오.
<code>OAuth2Options.ClientId</code>	권한 부여 서버가 발급한 클라이언트 식별자입니다.
<code>OAuth2Options.ClientSecret</code>	권한 부여 서버가 발급한 클라이언트 시크릿입니다.
<code>OAuth2Options.UserName</code>	resource owner의 사용자 이름.
<code>OAuth2Options.Password</code>	리소스 소유자의 비밀번호.
<code>AuthorizationServerOptions.TokenURL</code>	토큰 엔드포인트 URL. 사용자 자격 증명에 직접 제공되므로 AuthURL이 필요하지 않습니다.
<code>AuthorizationServerOptions.Scope</code>	접근 요청의 스코프입니다.

예제

```
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

서버가 refresh token을 반환하면, 이를 사용하여 사용자의 자격 증명을 다시 입력하지 않고 새 액세스 토큰을 얻을 수 있습니다:

사용 시기

다음의 경우에만 Resource Owner Password Credentials grant를 사용하십시오:

- 더 안전한 흐름으로 마이그레이션할 수 없는 레거시 애플리케이션입니다.
- 클라이언트와 권한 부여 서버를 모두 제어하는 자사 애플리케이션.
- 직접 username/password 인증에서 OAuth2로 전환하는 마이그레이션 시나리오.

다른 모든 경우에는 대신 [PKCE를 사용한 Authorization Code](#) 또는 [Authorization Code](#)를 사용하십시오.

OAuth2 | Device Authorization Grant (RFC 8628)

개요

Device Authorization 부여(Device Code 흐름이라고도 함)는 브라우저를 쉽게 표시하거나 키보드 입력을 받을 수 없는 입력 제약 장치를 위해 설계되었습니다. 예로는 스마트 TV, IoT 장치, 미디어 플레이어, CLI 도구, 키오스크 등이 있습니다. 장치는 짧은 코드를 표시하고, 사용자는 다른 장치(예: 휴대폰이나 컴퓨터)에 이를 입력하여 권한 부여를 완료합니다.

흐름

- 클라이언트는 `client_id` 및 `scope`를 **DeviceAuthorizationURL**로 POST합니다.
- 서버는 `device_code`, `user_code`, `verification_uri`, `expires_in`, `interval`을 반환합니다.
- 클라이언트는 사용자 코드와 `verification URI`와 함께 **OnDeviceCode** 이벤트를 발생시킵니다. 애플리케이션은 이를 사용자에게 표시합니다(예: "https://provider.com/device로 이동하여 코드를 입력하십시오: ABCD-1234").
- 사용자가 다른 장치에서 `verification URI`로 이동하여 `user code`를 입력합니다.
- 사용자가 보조 기기에서 인증하고 권한을 부여합니다.
- 한편, 클라이언트는 `device code`를 사용하여 `interval`초마다 `token` 엔드포인트를 자동으로 폴링합니다.
- 사용자가 권한 부여를 완료하면 다음 폴이 `access_token`을 반환합니다.
- 사용자가 권한을 부여하기 전에 디바이스 코드가 만료되면 **OnDeviceCodeExpired** 이벤트가 발생합니다.

구성

Property	설명
<code>OAuth2Options.GrantType</code>	auth2DeviceCode 로 설정하십시오.
<code>OAuth2Options.ClientId</code>	권한 부여 서버가 발급한 클라이언트 식별자입니다.
<code>OAuth2Options.ClientSecret</code>	선택 사항입니다. 일부 제공자는 이를 요구하고, 다른 제공자는 요구하지 않습니다.
<code>AuthorizationServerOptions.TokenURL</code>	폴링에 사용되는 토큰 엔드포인트 URL입니다.
<code>AuthorizationServerOptions.DeviceAuthorizationURL</code>	클라이언트가 <code>device code</code> 를 요청하는 <code>device authorization</code> 엔드포인트.
<code>AuthorizationServerOptions.Scope</code>	접근 요청의 스코프입니다.

Events

Event	설명
<code>OnDeviceCode</code>	서버로부터 <code>device code</code> 와 <code>user code</code> 가 수신될 때 발생합니다. 사용자에게 <code>user code</code> 와 <code>verification URI</code> 를 표시하십시오.
<code>OnDeviceCodeExpired</code>	사용자가 권한 부여를 완료하기 전에 장치 코드가 만료될 때 발생합니다. 애플리케이션은 사용자에게 흐름을 다시 시작하도록 안내해야 합니다.

<code>OnAfterAccessToken</code>	사용자가 권한 부여를 완료하고 액세스 토큰이 수신되면 발생합니다.
<code>OnErrorAccessToken</code>	device flow 중에 복구 불가능한 오류가 발생할 때 발생합니다.

예제

Device Code 이벤트 처리

```
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();
```

폴링 동작

`start`를 호출한 후, 구성 요소는 서버가 지정한 간격(일반적으로 5초)으로 토큰 엔드포인트를 자동으로 폴링합니다. 서버가 `slow_down` 오류로 응답하면 구성 요소는 폴링 간격을 5초 늘립니다. 폴링은 다음 중 하나가 발생할 때까지 계속됩니다:

- 사용자가 권한 부여를 완료하고 토큰이 반환됩니다.
- 장치 코드가 만료됩니다(`OnDeviceCodeExpired`가 발생함).
- 복구할 수 없는 오류가 발생합니다(`OnErrorAccessToken`이 발생함).

사용 시기

다음의 경우 Device Authorization grant를 사용하십시오:

- 스마트 TV 및 스트리밍 디바이스.
- 명령줄 인터페이스(CLI) 도구.
- 입력 기능이 제한된 IoT 장치.
- 키오스크 및 디지털 사이니지.
- 전체 URL이나 자격 증명을 입력하기가 비현실적인 모든 장치입니다.

OAuth2 | DPoP - Demonstrating Proof of Possession (RFC 9449)

개요

DPoP(Demonstrating Proof of Possession)는 개인 키의 소유 증명을 요구하여 액세스 토큰을 특정 클라이언트에 바인딩하는 보안 메커니즘입니다. 획득한 누구나 사용할 수 있는 bearer 토큰과 달리, DPoP에 바인딩된 토큰은 해당 개인 키 없이는 쓸모가 없습니다. 이는 토큰 탈취 및 재전송 공격에 대한 보호를 제공합니다.

DPoP은 grant type 자체가 아니라 모든 OAuth2 grant type(Authorization Code, Client Credentials 등)과 함께 사용할 수 있는 보안 향상 기능입니다.

작동 방식

- 클라이언트가 비대칭 키 쌍(ES256 또는 RS256)을 생성합니다.
- 클라이언트는 다음 구조로 **DPoP proof JWT**를 생성합니다:
 - Header: typ: dpop+jwt, alg: ES256, jwk: {public key}
 - Payload: htm (HTTP 메서드), htu (HTTP URI), iat (발급 시각), jti (고유 식별자)
- 클라이언트는 토큰 요청과 함께 DPoP 증명을 HTTP 헤더(DPoP: <proof>)로 보냅니다.
- Authorization 서버는 발행된 토큰을 클라이언트의 public key의 JWK thumbprint에 바인딩합니다.
- 서버는 token_type: Bearer 대신 token_type: DPoP를 반환합니다.
- 이후 API 호출에서 클라이언트는 Authorization: DPoP <token> 헤더와 해당 요청에 특정한 새 DPoP: <proof> 헤더를 모두 보냅니다.

Key Generation

GenerateDPoPKeyPair 메서드를 사용하여 Delphi에서 직접 ES256 키 쌍을 생성할 수 있습니다. 이는 내부적으로 OpenSSL을 사용하여 키 쌍을 생성하고 PrivateKey와 PublicKeyJWK를 모두 자동으로 설정합니다.

```
OAuth2->DPoPOptions->Enabled = true;
OAuth2->DPoPOptions->Algorithm = dpopES256;
OAuth2->GenerateDPoPKeyPair(); // Generates PrivateKey + PublicKeyJWK automatically
OAuth2->Start();
```

PublicKeyToJWK — 모든 키 또는 인증서를 JWK로 변환

이미 공개 키, 비공개 키 또는 X.509 인증서(PEM 형식)가 있는 경우 **PublicKeyToJWK** 클래스 함수를 사용하여 JWK로 변환할 수 있습니다. 이러한 입력 중 하나를 받아 올바른 JWK JSON을 반환합니다:

Input	PEM Header
EC 또는 RSA 공개 키	-----BEGIN PUBLIC KEY-----
EC private key	-----BEGIN EC PRIVATE KEY-----
RSA private key	-----BEGIN RSA PRIVATE KEY-----
X.509 인증서	-----BEGIN CERTIFICATE-----

PEM 키 파일로부터

```
// 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"));
```

스마트 카드 인증서에서

스마트 카드와 HSM은 개인 키를 보호합니다. 읽거나 내보낼 수 없습니다. 그러나 (공개 키를 포함하는) **인증서**는 항상 내보낼 수 있습니다. 인증서를 PEM으로 내보내고 PublicKeyToJWK에 전달하십시오:

```
// 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;
```

이 접근 방식은 인증서를 내보낼 수 있는 모든 카드 또는 HSM에서 작동합니다: PKCS#11 토큰, Windows CNG 스마트 카드, USB 보안 키(YubiKey, Feitian), 클라우드 HSM(Azure Key Vault, AWS CloudHSM).

OpenSSL 명령줄 도구 사용

키 쌍을 외부에서 생성할 수도 있습니다:

DPoP Options

Property	설명
<code>DPoPOptions.Enabled</code>	DPoP proof 생성을 활성화하려면 True 로 설정하십시오.
<code>DPoPOptions.Algorithm</code>	서명 알고리즘. 지원되는 값: ES256 (권장), RS256 .
<code>DPoPOptions.PrivateKey</code>	DPoP 증명 JWT에 서명하는 데 사용되는 PEM 인코딩된 개인 키입니다.
<code>DPoPOptions.PublicKeyJWK</code>	DPoP proof 헤더에 포함된 JWK(JSON Web Key) 형식의 공개 키.

Methods

메서드	설명
<code>GenerateDPoPKeyPair</code>	OpenSSL을 사용하여 ES256 키 쌍을 생성하고 <code>DPoPOptions.PrivateKey</code> (PEM)와 <code>DPoPOptions.PublicKeyJWK</code> (JSON)를 모두 자동으로 설정합니다.
<code>GetDPoPProof(Method, URL, AccessToken)</code>	주어진 HTTP 메서드, URL, access token에 대한 DPoP proof JWT를 생성합니다. 서명된 JWT 문자열을 반환합니다. proof에는 proof를 토큰에 바인딩하기 위한 <code>ath</code> claim(access token 해시)이 포함됩니다.
<code>GetDPoPJWKThumbprint</code>	서버가 토큰을 바인딩하는 데 사용하는 공개 키의 RFC 7638 JWK thumbprint(SHA-256)를 반환합니다.

Nonce 처리

일부 authorization 서버는 DPOP proof에 서버 제공 nonce를 요구합니다. 서버가 DPOP-Nonce 헤더와 400 또는 401 상태로 응답하면, 구성 요소는 자동으로:

1. DPOP-Nonce 응답 헤더에서 nonce를 추출합니다.
2. 다음 DPOP 증명 JWT의 nonce 클레임에 nonce를 포함합니다.
3. 업데이트된 증명으로 요청을 재시도합니다.

OnDPoPSign을 사용한 사용자 지정 서명(HSM / Smart Card)

일부 환경에서는 비공개 키가 하드웨어 보안 모듈(HSM) 또는 스마트 카드(예: PKCS#11 또는 Windows CNG를 통해)에 저장되어 추출할 수 없습니다. OnDPoPSign 이벤트를 사용하면 구성 요소가 나머지 모든 것(proof 구조, nonce, 헤더)을 처리하는 동안 서명을 외부에서 수행할 수 있습니다.

OnDPoPSign이 할당되면 구성 요소는 내부 서명을 시도하기 전에 이를 호출합니다. Handled := True로 설정하고 Signature를 제공하면 구성 요소는 사용자의 서명을 사용하고 내부 서명을 건너뜁니다. 이 경우 DPoPOptions.PrivateKey 속성은 필요하지 않습니다.

이벤트 시그니처

매개변수	Direction	설명
SigningInput	in	서명할 데이터: Base64URL(header) + "." + Base64URL(payload) . 이것은 표준 JWS 서명 입력입니다.
Algorithm	in	알고리즘 이름: "ES256" 또는 "RS256" .
Signature	out	서명 기기가 생성한 Base64URL 인코딩된 서명 바이트로 이를 설정하십시오.
Handled	out	사용자 지정 서명을 사용하려면 True 로 설정하십시오. False 인 경우, 구성 요소는 PrivateKey 를 사용한 내부 서명으로 대체됩니다.

Smart Card Example

다음 예제는 서명 카드와 함께 DPOP을 사용하는 방법을 보여 줍니다. 개인 키는 카드를 떠나지 않습니다. 해시만 서명을 위해 카드로 전송됩니다.

```
// 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;
}
```

이 접근 방식은 모든 서명 백엔드와 작동합니다: PKCS#11 토큰, Windows CNG(CryptoAPI Next Generation), Azure Key Vault, AWS CloudHSM, 또는 ECDSA/RSA 서명을 생성할 수 있는 모든 사용자 지정 하드웨어.

예제

```
// 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

이 구성 요소는 서버 측 구성 요소에서 OAuth2 프로토콜 구현을 제공합니다.

소개

이 구성 요소는 서버 측 구성 요소에서 OAuth2 프로토콜 구현을 제공합니다.

서버 구성 요소에는 TsgcHTTP_OAuth2_Server의 인스턴스를 할당할 수 있는 Authorization.OAuth.OAuth2라는 속성이 있습니다. 따라서 Authentication이 활성화되고 OAuth2 속성이 OAuth2 Server Component에 연결되면, WebSocket 및 HTTP Request는 처리되기 위해 Bearer Token이 필요하며, 그렇지 않으면 연결이 자동으로 닫힙니다.

```
TsgcHTTP_OAuth2_Server *OAuth2 = new TsgcHTTP_OAuth2_Server(this);
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2 = OAuth2;
```

서버는 다음 권한 부여 유형을 지원합니다:

- **auth2Code:** 단일 페이지 애플리케이션, 웹 애플리케이션, 네이티브 설치 애플리케이션을 포함한 대부분의 애플리케이션 유형에서 인증 및 권한 부여를 수행하는 데 사용됩니다. 이 흐름을 통해 앱은 보안 리소스에 액세스하는 데 사용할 수 있는 access_token, 추가 access_token을 얻기 위한 갱신 토큰, 그리고 로그인한 사용자의 ID 토큰을 안전하게 획득할 수 있습니다.
- **auth2ClientCredentials:** 이 유형의 그랜트는 일반적으로 사용자와의 즉각적인 상호 작용 없이 백그라운드에서 실행되어야 하는 서버 대 서버 상호 작용에 사용됩니다. 이러한 유형의 애플리케이션은 종종 데몬 또는 서비스 계정이라고 합니다.
- **password** (Resource Owner Password Credentials): 애플리케이션이 사용자의 자격 증명을 직접 처리하여 사용자를 로그인시킬 수 있도록 합니다. 클라이언트는 사용자의 사용자 이름과 비밀번호를 토큰 엔드포인트로 보냅니다.
- **urn:ietf:params:oauth:grant-type:device_code** (Device Code): RFC 8628에 따른 Device Authorization Grant. 사용자가 보조 디바이스에서 권한을 부여하도록 하여 입력이 제한된 디바이스(스마트 TV, IoT 디바이스)가 사용자 권한 부여를 얻을 수 있도록 합니다.

Authorization 유형은 앱을 등록할 때 사용자 정의할 수 있으며, 기본적으로 모든 권한 부여 유형이 지원됩니다.

Endpoints

기본적으로, 구성 요소는 Authorization 및 Token 요청을 처리하기 위해 다음 엔드포인트로 구성됩니다

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

따라서 서버가 포트 443에서 수신 대기하고 도메인이 www.esegece.com인 경우 EndPoints는 다음과 같습니다:

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>

엔드포인트는 OAuth2Options 속성에서 구성할 수 있습니다.

기본적으로 **PKCE** (CSRF 및 authorization code 주입 공격을 방지하기 위한 Authorization Code 흐름의 확장)가 활성화되어 있습니다.

구성

OAuth2 프로세스를 시작하기 전에 어떤 앱을 사용할 수 있는지 등록해야 하며, 이는 OAuth2 서버 구성 요소의 Apps 속성을 사용하여 수행됩니다.

Register App

Apps.AddApp을 사용하여 OAuth2 서버에 새 Application을 추가하며, 다음 매개변수를 설정해야 합니다:

- **App Name:** 애플리케이션의 이름입니다. 예: MyApp
- **RedirectURI:** 응답이 리디렉션될 위치입니다. 예: http://127.0.0.1:8080
- **ClientId:** 는 public 정보이며 클라이언트의 ID입니다.
- **ClientSecret:** 기밀로 유지되어야 합니다.

선택적으로 다음 매개변수를 설정할 수 있습니다:

- **ExpiresIn:** 기본값은 3600초이므로 토큰은 1시간 후에 만료됩니다. 필요한 경우 더 큰 값을 설정할 수 있습니다.
- **RefreshToken:** 기본적으로 갱신 토큰이 지원되며, 그렇지 않은 경우 이 매개변수를 false로 설정하십시오.
- **AllowedGrantTypes:** 기본적으로 모든 부여 유형이 지원됩니다(auth2Code 및 auth2ClientCredentials). 그러나 서버는 Code Authorization만 허용하거나 Client Credentials만 허용하도록 구성할 수 있습니다.

앱 삭제

기존 App을 삭제하려면 Apps.RemoveApp을 사용하십시오.

AddToken

일부 token이 발급된 상태에서 서버가 다시 시작된 경우, OAuth2 서버를 시작하기 전이자 App을 등록한 후에 AddToken 메서드를 사용하여 이러한 token을 복구할 수 있습니다

- **AppName:** 애플리케이션의 이름입니다.
- **Token:** 액세스 토큰입니다.
- **Expires:** 토큰이 만료되는 시점입니다.
- **RefreshToken:** 갱신 토큰.

RemoveToken

이미 발행된 Token을 제거합니다.

OAuth2Options

OAuth2Options 속성을 사용하면 서버 엔드포인트와 선택적 기능을 구성할 수 있습니다.

취소(Revocation)

[RFC 7009](#)에 따른 토큰 취소입니다. 활성화하면 클라이언트가 이전에 발급된 access 또는 refresh 토큰을 취소할 수 있습니다.

- **OAuth2Options.Revocation.Enabled:** revocation 엔드포인트를 활성화하려면 True로 설정하십시오.
- **OAuth2Options.Revocation.URL:** 엔드포인트 URL 경로입니다. 기본값: /sgc/oauth2/revoke

Introspection

[RFC 7662](#)에 따른 토큰 introspection. 활성화되면 리소스 서버는 권한 부여 서버에 쿼리하여 토큰의 활성 상태와 메타데이터를 확인할 수 있습니다.

- **OAuth2Options.Introspection.Enabled:** introspection 엔드포인트를 활성화하려면 True로 설정하십시오.
- **OAuth2Options.Introspection.URL:** 엔드포인트 URL 경로입니다. 기본값: /sgc/oauth2/introspect

DeviceAuthorization

[RFC 8628](#)에 따른 Device Authorization Grant. 활성화되면 입력 제약 장치가 보조 장치에서 사용자가 권한을 부여하도록 하여 권한 부여를 요청할 수 있습니다.

- **OAuth2Options.DeviceAuthorization.Enabled:** device authorization 엔드포인트를 활성화하려면 True로 설정하십시오.
- **OAuth2Options.DeviceAuthorization.URL:** device code 요청을 위한 엔드포인트 URL 경로. 기본값: /sgc/oauth2/device
- **OAuth2Options.DeviceAuthorization.VerificationURL:** 사용자 검증 페이지의 엔드포인트 URL 경로입니다. 기본값: /sgc/oauth2/device/verify

- **OAuth2Options.DeviceAuthorization.ExpiresIn**: device code의 수명(초)입니다. 기본값: 600 (10분).
- **OAuth2Options.DeviceAuthorization.Interval**: 클라이언트가 token 엔드포인트를 폴링할 때 사용해야 하는 최소 폴링 간격(초). 기본값: 5.

참조

- 속성 — 구성 요소에서 사용할 수 있는 2개의 속성.
- 메서드 — 11개의 공개 메서드.
- 이벤트 — 처리할 수 있는 13개의 이벤트.

Guides

- **QuickStart**
 - [OAuth2 Server Example](#)
 - [OAuth2 Customize Sign-in HTML](#)
 - [OAuth2 Server Endpoints](#)
 - [OAuth2 Register Apps](#)
 - [OAuth2 Recover Access Tokens](#)
- **인증**
 - [OAuth2 서버 인증](#)
 - [일부 URL에 대한 OAuth2 None 인증](#)

OAuth2 | Server Example

[TsgcWebSocketHTTPServer](#)를 사용하여 간단한 OAuth2 서버 예제를 만들어 보겠습니다.

먼저, 포트 443에서 수신하고 sgc.pem 파일의 자체 서명된 인증서를 사용하는 새 TsgcWebSocketHTTPServer를 생성하십시오.

```
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;
```

그런 다음 TsgcHTTP_OAuth2_Server의 새 인스턴스를 생성하고 이전에 생성한 서버에 할당하십시오.
다음 값으로 새 Application을 등록하십시오:

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;
```

그런 다음 OAuth2 server 구성 요소의 OnOAuth2Authentication 이벤트를 처리하고 사용자를 로그인하는 자체 메서드를 구현하십시오. 로그인을 수락하기 위해 "user/secret" 쌍을 사용하겠습니다.

```
void OnOAuth2Authentication(TsgcWSConnection *Connection, TsgcHTTPOAuth2Request *OAuth2, string aUser,
    string aPassword, ref bool Authenticated)
{
    if ((aUser == "user") and (aPassword == "secret"))
    {
        Authenticated = true;
    }
}
```

마지막으로 서버를 시작하고 OAuth2 클라이언트를 사용하여 로그인하십시오. 예를 들어 sgcWebSockets 라이브러리에 포함된 [TsgcHTTP_OAuth2_Client](#)를 사용할 수 있습니다.

 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

새 Access Token을 요청하면 새 웹 브라우저 세션이 표시되며, 사용자는 연결을 허용한 다음 로그인해야 합니다.

OAuth2 Authorization

Sign in

Introduce your username and password.

GO BACK

Sign in

SIGN IN

로그인이 성공하면 새 토큰이 클라이언트에 반환됩니다. 그러면 모든 요청에 이 bearer 토큰이 HTTP 헤더에 포함되어야 합니다.

 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 | Customize Sign-In HTML

OAuth2 클라이언트가 새 Access Token을 가져오는 요청을 하면 이 연결을 허용하고 User와 Password로 로그인하기 위해 웹 브라우저에 Web-Page가 표시됩니다.

HTML 페이지는 Server 구성 요소에 기본적으로 포함되어 있지만, 이 코드는 **OnAuth2BeforeDispatchPage** 이벤트를 사용하여 사용자 정의할 수 있습니다.

```
void OnAuth2BeforeDispatchPage(TObject *Sender; TsgcHTTPOAuth2Request *OAuth2; ref string HTML)
{
    HTML = "your custom html";
}
```

완전히 새로운 HTML 코드로 HTML을 사용자 지정하는 경우, 최소한 사용자 이름과 비밀번호가 전송되는 form은 유지해야 합니다.

```
<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>
```

숨겨진 id 매개변수는 서버가 요청을 식별할 수 있도록 원본 폼의 동일한 값을 유지해야 합니다.

OAuth2 | Server Endpoints

기본적으로 OAuth2 Server는 다음 엔드포인트를 사용합니다:

Authorization: /sgc/oauth2/auth

Token: /sgc/oauth2/token

Revocation: /sgc/oauth2/revoke (POST) - RFC 7009에 따라 토큰을 취소합니다

Introspection: /sgc/oauth2/introspect (POST) - RFC 7662에 따라 토큰 메타데이터를 반환합니다

Device Authorization: /sgc/oauth2/device (POST) - RFC 8628에 따라 디바이스 코드를 발급합니다

Device Verification: /sgc/oauth2/device/verify (GET/POST) - 사용자 verification 페이지

즉, 서버가 IP 80.54.41.30과 포트 8443에서 수신 대기하는 경우 전체 OAuth2 엔드포인트는 다음과 같습니다:

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

이 엔드포인트는 쉽게 수정할 수 있습니다. 구성 요소의 OAuth2Options 속성에 액세스하여 Authorization 및 Token URL을 수정하기만 하면 됩니다.

예: 엔드포인트가 다음과 같아야 하는 경우

Authorization: https://80.54.41.30:8443/authentication/auth

Token: https://80.54.41.30:8443/authentication/token

다음 값으로 OAuth2Options 속성을 설정하십시오:

OAuth2Options.Authorization.URL = /authentication/auth

OAuth2Options.Token.URL = /authentication/token

동일한 접근 방식이 다른 엔드포인트에도 적용됩니다:

OAuth2Options.Revocation.URL = /authentication/revoke

OAuth2Options.Introspection.URL = /authentication/introspect

OAuth2Options.DeviceAuthorization.URL = /authentication/device

OAuth2Options.DeviceAuthorization.VerificationURL = /authentication/device/verify

OAuth2 | Register Apps

클라이언트가 새 OAuth2를 요청하기 전에 App이 서버에 등록되어야 합니다.
새 App을 등록하려면 다음 정보가 필요합니다:

- **App Name:** 애플리케이션의 이름입니다. 예: MyApp
- **RedirectURI:** 응답이 리디렉션될 위치입니다. 예: http://127.0.0.1:8080
- **ClientId:** 는 public 정보이며 클라이언트의 ID입니다.
- **ClientSecret:** 기밀로 유지되어야 합니다.

선택적으로 다음 매개변수를 설정할 수 있습니다:

- **ExpiresIn:** 기본값은 3600초이므로 토큰은 1시간 후에 만료됩니다. 필요한 경우 더 큰 값을 설정할 수 있습니다.
- **RefreshToken:** 기본적으로 갱신 토큰이 지원되며, 그렇지 않은 경우 이 매개변수를 false로 설정하십시오.

새 클라이언트가 OAuth2를 사용하여 인증하려면 먼저 App이 서버에 등록되어야 하며, 다음을 사용할 수 있습니다:

1. RegisterApp

이 메서드는 앱 이름과 RedirectURI가 필요하며, ClientId와 ClientSecret을 반환합니다.

2. Apps.AddApp

이 메서드에는 AppName, RedirectURI, ClientId 및 ClientSecret이 필요합니다. 일반적으로 서버에 이미 생성된 App이 있고 시작하기 전에 이를 로드하려는 경우 이 메서드를 사용할 수 있습니다.

두 메서드 모두 동일한 작업, 즉 서버에 애플리케이션을 등록하지만, 첫 번째는 App을 처음 등록할 때 가장 유용하고, 두 번째 메서드는 서버를 시작하기 전에 등록된 모든 앱을 로드하려는 경우(예: 데이터베이스에 저장되어 있기 때문에)에 유용합니다.

OAuth2 | Recover Access Tokens

OAuth2 Server가 소멸되고(재시작되기 때문에) 발행된 유효한 Access Token이 있는 경우, 기본적으로 이들은 손실됩니다. **AddToken** 메서드를 사용하여 이러한 Access Token을 복구할 수 있습니다. 이 메서드는 토큰을 OAuth2 Server에 저장합니다.

Token 추가에는 다음 정보가 필요합니다:

- **AppName:** 앱의 이름입니다.
- **Token:** 액세스 토큰입니다.
- **Expires:** 토큰이 만료되는 시점입니다.
- **RefreshToken:** 갱신 토큰.

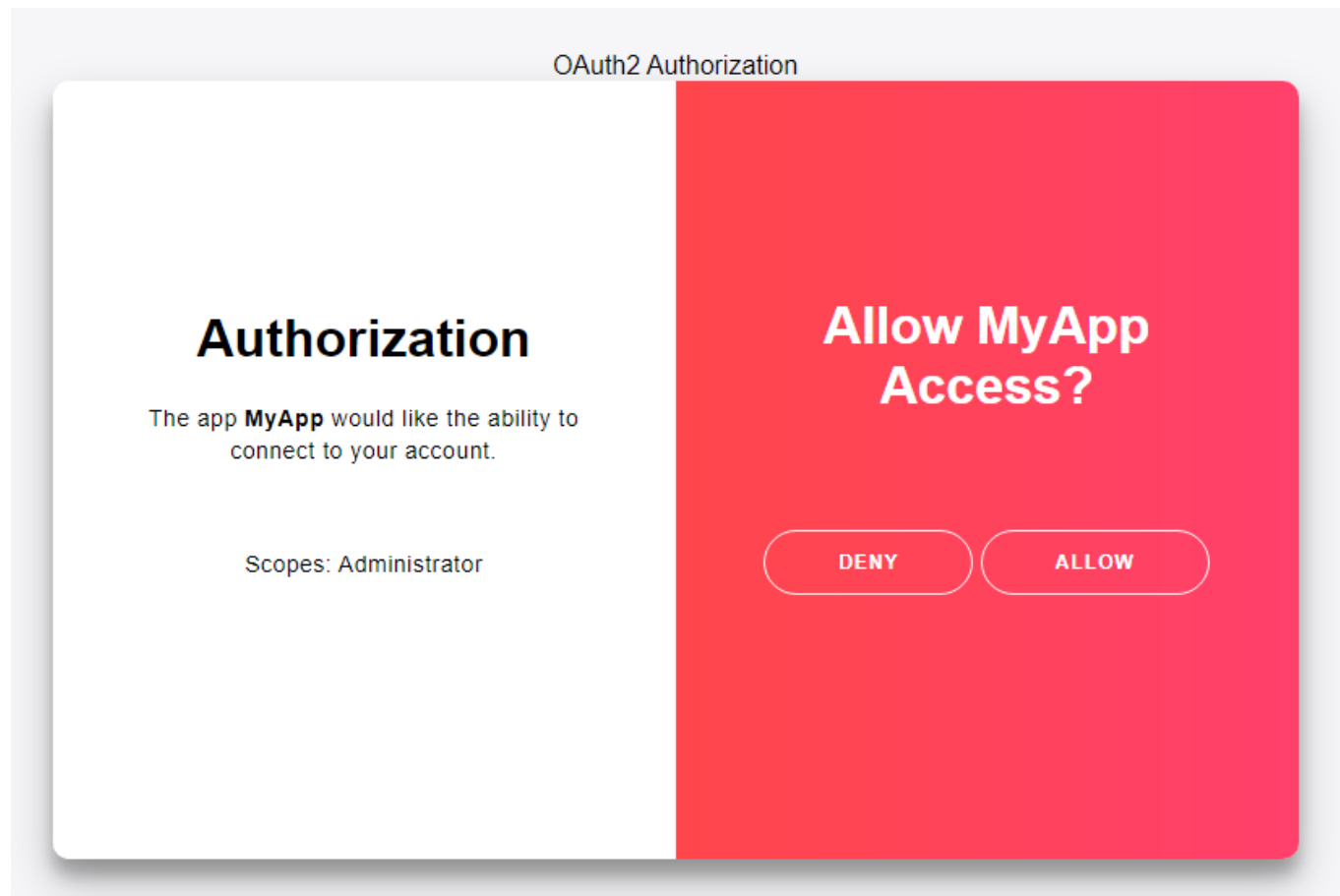
OAuth2AfterAccessToken 이벤트를 처리하여 발급된 토큰을 저장할 수 있습니다.

```
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->OAuth2 = OAuth2;
```


OAuth2 | Server Authentication

OAuth2 클라이언트가 새 Authorization을 요청하면, 서버는 사용자가 연결을 허용한 다음 로그인해야 하는 웹 페이지를 표시합니다. 이 페이지는 sgcWebSockets 라이브러리에서 제공하며 클라이언트가 Authorization을 요청할 때 자동으로 디스패치됩니다.



사용자가 접근을 허용하면, 사용자가 사용자 이름과 비밀번호를 설정해야 하는 로그인 양식이 표시됩니다. 이 데이터는 OnOAuth2Authentication 이벤트에서 수신되므로, 사용자/비밀번호가 올바른지 검증해야 하며, 올바른 경우 Authenticated 매개변수를 true로 설정하십시오.

```
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

기본적으로 서버 측에서 OAuth2가 활성화되면 모든 HTTP 요청은 Bearer Token을 사용한 인증이 필요합니다. Bearer Token을 사용할 필요 없이 일부 URL에 액세스할 수 있도록 허용하려면 **OnOAuth2BeforeRequest** 이벤트를 사용할 수 있습니다

```
procedure OnOAuth2BeforeRequest(TObject *Sender; TsgcWSConnection *aConnection; TStringList *aHeaders;  
    ref bool Cancel)  
{  
    if (DecodeGETFullPath(aHeaders) == "/Public.html")  
    {  
        Cancel = true;  
    }  
}
```

OAuth2 Server | Authorization Code Grant

개요

Authorization Code grant 는 서버 측 애플리케이션을 위한 가장 일반적인 OAuth2 흐름입니다. [TsgcHTTP_OAuth2_Server](#)는 이 흐름을 처리하기 위해 두 개의 엔드포인트를 제공합니다: 사용자에게 로그인 페이지를 표시하는 Authorization 엔드포인트와 authorization code를 access token으로 교환하는 Token 엔드포인트입니다. 이 흐름은 클라이언트가 브라우저와 상호 작용할 수 있는 웹 애플리케이션, 데스크톱 애플리케이션 및 모바일 애플리케이션에 적합합니다.

흐름

- 클라이언트는 사용자를 `/sgc/oauth2/auth?response_type=code&client_id=...&redirect_uri=...&scope=...&state=...`로 리디렉션합니다
- 서버는 HTML 로그인 페이지의 사용자 정의를 허용하기 위해 **OnOAuth2BeforeDispatchPage**를 발생시킵니다.
- 사용자가 자격 증명을 제출합니다. 서버는 사용자 이름과 비밀번호를 검증하기 위해 **OnOAuth2Authentication**을 발생시킵니다.
- 인증되면 서버는 `?code=...&state=...`가 추가된 `redirect_uri`로 사용자를 리디렉션합니다.
- 클라이언트는 `grant_type=authorization_code&code=...&redirect_uri=...&client_id=...&client_secret=...`로 `/sgc/oauth2/token`에 POST합니다
- 서버가 권한 부여 코드와 클라이언트 자격 증명을 검증하고, **OnOAuth2AfterAccessToken**을 발생시키고, `access_token`, `token_type`, `expires_in` 및 선택적으로 `refresh_token`을 포함하는 JSON 응답을 반환합니다.

구성

Property	설명
<code>OAuth2Options.Authorization.Enabled</code>	권한 부여 엔드포인트를 활성화하려면 True로 설정하십시오. 기본값: True.
<code>OAuth2Options.Authorization.URL</code>	권한 부여 엔드포인트 URL 경로. 기본값: <code>/sgc/oauth2/auth</code>
<code>OAuth2Options.Token.Enabled</code>	토큰 엔드포인트를 활성화하려면 True로 설정하십시오. 기본값: True.
<code>OAuth2Options.Token.URL</code>	token 엔드포인트 URL 경로. 기본값: <code>/sgc/oauth2/token</code>
<code>OAuth2Options.PKCE</code>	True인 경우 서버는 권한 부여 코드 요청에 PKCE(<code>code_challenge</code> 및 <code>code_verifier</code>)를 요구합니다. 기본값: True.

App Registration

클라이언트가 Authorization Code 흐름을 사용할 수 있으려면 먼저 서버에 애플리케이션을 등록해야 합니다. `Apps.AddApp`을 사용하고 `AllowedGrantTypes`에 `auth2Code`가 포함되어 있는지 확인하십시오.

```
OAuth2Server->Apps->AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2Code);
```

Events

Event	설명
<code>OnOAuth2BeforeDispatchPage</code>	로그인 HTML 페이지가 사용자에게 전송되기 전에 발생합니다. HTML 내용의 사용자 정의를 허용합니다.
<code>OnOAuth2Authentication</code>	사용자가 자격 증명을 제출할 때 발생합니다. 여기에서 사용자 이름과 비밀번호를 검증하고 <code>Authenticated</code> 를 True 또는 False로 설정하십시오.
<code>OnOAuth2AfterAccessToken</code>	서버가 액세스 토큰을 성공적으로 발급한 후 발생합니다. 로깅에 유용합니다.
<code>OnOAuth2AfterValidateAccessToken</code>	클라이언트가 유효하고 만료되지 않은 토큰으로 요청할 때 발생합니다. 로깅 및 감사에 유용합니다.

예제

다음 예제는 앱 등록 및 이벤트 핸들러를 포함하여 Authorization Code grant를 사용한 완전한 서버 설정을 보여줍니다.

```
// 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

`OAuth2Options.PKCE`가 `True`(기본값)로 설정되면, 서버는 Proof Key for Code Exchange를 적용합니다. 클라이언트는 권한 부여 요청에 `code_challenge` 매개변수를, 토큰 요청에 해당하는 `code_verifier`를 포함해야 합니다. 서버는 토큰을 발급하기 전에 `code_verifier`가 원래 `code_challenge`와 일치하는지 검증합니다.

PKCE는 모든 클라이언트, 특히 클라이언트 시크릿을 안전하게 저장할 수 없는 공개 클라이언트(네이티브 앱 및 SPA)에 권장됩니다.

OAuth2 Server | Client Credentials Grant

개요

Client Credentials grant은 사용자 상호작용이 필요하지 않은 기계 대 기계(M2M) 통신에 사용됩니다. 클라이언트는 자체 자격 증명(client_id 및 client_secret)을 사용하여 [TsgcHTTP_OAuth2_Server](#)에 직접 인증하고 액세스 토큰을 받습니다. authorization code 단계, 리디렉션 또는 사용자 로그인 페이지가 없습니다.

이 grant type은 클라이언트 애플리케이션이 특정 사용자를 대신하지 않고 자체적으로 작동할 때 적합합니다.

흐름

1. 클라이언트가 grant_type=client_credentials&client_id=...&client_secret=...&scope=...로 토큰 엔드포인트에 POST합니다.
2. 서버는 등록된 애플리케이션과 대조하여 클라이언트 자격 증명을 검증합니다.
3. 서버가 **OnOAuth2AfterAccessToken**을 발생시키고 access_token, token_type 및 expires_in이 포함된 JSON 응답을 반환합니다.

이 그랜트 유형에서는 인증할 사용자가 없으므로 onOAuth2Authentication 이벤트가 발생하지 않습니다. 서버는 등록된 앱에 대해 client_id와 client_secret을 자동으로 검증합니다.

구성

auth2ClientCredentials를 포함하는 AllowedGrantTypes로 애플리케이션을 등록하십시오. 이 흐름에는 토큰 엔드포인트만 필요합니다.

Property	설명
<code>OAuth2Options.Token.Enabled</code>	토큰 엔드포인트를 활성화하려면 True로 설정하십시오. 기본값: True.
<code>OAuth2Options.Token.URL</code>	token 엔드포인트 URL 경로. 기본값: /sgc/oauth2/token

App Registration

```
OAuth2Server->Apps->AddApp("MyServiceApp", "",
    "service-client-id", "service-client-secret", 3600, false,
    TsgcOAuth2GrantTypes() << auth2ClientCredentials);
```

리디렉션이 관련되지 않으므로 RedirectURI는 비워 둘 수 있습니다. 클라이언트가 자격 증명을 사용하여 언제든지 새 토큰을 요청할 수 있으므로 RefreshToken은 일반적으로 False로 설정됩니다.

Events

Event	설명
<code>OnOAuth2AfterAccessToken</code>	서버가 액세스 토큰을 성공적으로 발급한 후 발생합니다. 로깅에 유용합니다.
<code>OnOAuth2AfterValidateAccessToken</code>	클라이언트가 유효한 토큰으로 요청할 때 발생합니다. 감사에 유용합니다.

예제

```
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);
}
```

사용 시기

서버에서 Client Credentials grant를 다음 용도로 사용하십시오:

- 사용자 상호작용 없이 인증하는 백엔드 서비스 및 데몬.
- 서버 간 API 통신입니다.
- 마이크로서비스 간 인증입니다.
- 자동화된 프로세스, 예약된 작업 및 배치 작업입니다.

OAuth2 Server | Resource Owner Password Credentials Grant

개요

Resource Owner Password Credentials(ROPC) 권한 부여는 클라이언트가 사용자의 사용자 이름과 비밀번호를 토큰 엔드포인트로 직접 보낼 수 있게 합니다. [TsgcHTTP_OAuth2_Server](#)는 **OnOAuth2Authentication** 이벤트를 통해 자격 증명을 검증하고 인증이 성공하면 액세스 토큰을 발급합니다.

이 흐름은 브라우저 리디렉션이나 로그인 페이지를 포함하지 않습니다. 클라이언트는 사용자로부터 직접 자격 증명을 수집하여 서버에 제출합니다.

보안 경고

이 **grant** 유형은 **OAuth 2.1**에서 더 이상 사용되지 않습니다. 다른 흐름(예: PKCE가 있는 Authorization Code)이 실행 가능하지 않은 경우(예: 레거시 애플리케이션 또는 매우 신뢰할 수 있는 1차 클라이언트)에만 사용해야 합니다. PKCE가 있는 Authorization Code grant가 권장되는 대안입니다.

흐름

- 클라이언트가 `grant_type=password&username=...&password=...&client_id=...&client_secret=...&scope=...`로 토큰 엔드포인트에 POST합니다
- 서버는 사용자 이름과 비밀번호를 검증하기 위해 **OnOAuth2Authentication**을 발생시킵니다.
- 인증되면 서버는 **OnOAuth2AfterAccessToken**을 발생시키고 `access_token`, `token_type`, `expires_in` 및 선택적으로 `refresh_token`이 포함된 JSON 응답을 반환합니다.
- 인증이 실패하면 서버가 오류 응답을 반환합니다.

구성

`auth2ResourceOwnerPassword`를 포함하는 `AllowedGrantTypes`로 애플리케이션을 등록합니다.

Property	설명
<code>OAuth2Options.Token.Enabled</code>	토큰 엔드포인트를 활성화하려면 <code>True</code> 로 설정하십시오. 기본값: <code>True</code> .
<code>OAuth2Options.Token.URL</code>	token 엔드포인트 URL 경로. 기본값: <code>/sgc/oauth2/token</code>

App Registration

```
OAuth2Server->Apps->AddApp("MyApp", "",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2ResourceOwnerPassword);
```

Events

Event	설명

<code>OnOAuth2Authentication</code>	서버가 사용자 이름과 비밀번호를 수신할 때 발생합니다. 여기에서 자격 증명을 검증하고 <code>Authenticated</code> 를 True 또는 False로 설정하십시오.
<code>OnOAuth2AfterAccessToken</code>	서버가 액세스 토큰을 성공적으로 발급한 후 발생합니다. 로깅에 유용합니다.
<code>OnOAuth2AfterValidateAccessToken</code>	클라이언트가 유효한 토큰으로 요청을 할 때 발생합니다.

예제

```
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->OAuth->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);
}
```

사용 시기

다음의 경우에만 Resource Owner Password Credentials grant를 사용하십시오:

- 클라이언트는 고도로 신뢰할 수 있는 1차 애플리케이션입니다.
- 다른 그랜트 유형(Authorization Code with PKCE)은 실행 가능하지 않습니다.
- 사용자 자격 증명을 이미 직접 수집하는 레거시 애플리케이션을 마이그레이션합니다.

새 애플리케이션의 경우 항상 PKCE가 있는 [Authorization Code](#) grant를 선호하십시오.

OAuth2 Server | Refresh Token Grant

개요

Refresh Token grant는 클라이언트가 사용자 상호작용 없이 이전에 발급된 refresh token을 새 access token으로 교환할 수 있도록 합니다. [TsgcHTTP_OAuth2_Server](#)는 refresh token을 유효성 검사하고 새 access token(및 선택적으로 새 refresh token)을 발급합니다.

이는 액세스 토큰의 만료가 짧지만 클라이언트가 보호된 리소스에 대한 지속적인 접근이 필요한 장기 실행 세션에 유용합니다.

흐름

1. 클라이언트가 `grant_type=refresh_token&refresh_token=...&client_id=...&client_secret=...&scope=...`로 토큰 엔드포인트에 POST합니다
2. 서버가 저장된 토큰에 대해 갱신 토큰을 검증합니다.
3. 서버는 **OnOAuth2AfterRefreshToken**을 발생시키고 새 `access_token`, `token_type`, `expires_in` 및 선택적으로 새 `refresh_token`이 포함된 JSON 응답을 반환합니다.

구성

Refresh token은 앱을 등록할 때 애플리케이션별로 활성화됩니다. `Apps.AddApp`에서 `RefreshToken` 매개변수를 `True`로 설정하십시오.

Property	설명
<code>OAuth2Options.Token.Enabled</code>	토큰 엔드포인트를 활성화하려면 <code>True</code> 로 설정하십시오. 기본값: <code>True</code> .
<code>OAuth2Options.Token.URL</code>	token 엔드포인트 URL 경로. 기본값: <code>/sgc/oauth2/token</code>
<code>RefreshToken (AddApp parameter)</code>	이 애플리케이션에 대해 갱신 토큰을 발급하려면 <code>True</code> 로 설정하십시오. 기본값: <code>True</code> .

App Registration

앱을 등록할 때 서버가 액세스 토큰과 함께 갱신 토큰을 발급하도록 `RefreshToken` 매개변수를 `True`로 설정하십시오.

```
// RefreshToken = true (6th parameter)
OAuth2Server->Apps->AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2Code);
```

Events

Event	설명
<code>OnOAuth2AfterRefreshToken</code>	서버가 refresh token에서 새 액세스 토큰을 성공적으로 발행한 후 발생합니다. 토큰 교체를 기록하는 데 유용합니다.
<code>OnOAuth2AfterAccessToken</code>	새 액세스 토큰이 발급된 후 발생합니다.

예제

```
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);
}
```

토큰 순환

갱신 토큰이 사용되면, 서버는 새 액세스 토큰과 함께 새 갱신 토큰을 발급할 수 있습니다. 이전 갱신 토큰은 무효화됩니다. 이를 갱신 토큰 회전이라고 하며 토큰 재생 공격을 방지하는 데 도움이 됩니다.

재시작 후 토큰 복구

토큰이 여전히 유효한 동안 서버가 다시 시작되면 서버를 시작하기 전에 Apps.AddToken을 사용하여 복구할 수 있습니다. 자세한 내용은 [OAuth2 Recover Access Tokens](#)를 참조하십시오.

OAuth2 Server | Device Authorization Grant (RFC 8628)

개요

Device Authorization Grant는 브라우저를 쉽게 표시하거나 키보드 입력을 받을 수 없는 입력 제약 장치(스마트 TV, IoT 장치, CLI 도구)를 위해 설계되었습니다. [TsgcHTTP_OAuth2_Server](#)는 장치 권한 부여 엔드포인트와 확인 페이지를 제공하여 사용자가 휴대폰이나 컴퓨터와 같은 보조 장치에서 장치를 인증할 수 있도록 합니다.

흐름

1. 장치는 `client_id=...&scope=...`와 함께 `/sgc/oauth2/device`로 POST합니다
2. 서버는 **OnOAuth2DeviceAuthorization**을 발생시키고 `device_code`, `user_code`, `verification_uri`, `expires_in` 및 `interval`이 포함된 JSON 응답을 반환합니다.
3. 장치는 `user_code`와 `verification_uri`를 사용자에게 표시합니다(예: "https://example.com/sgc/oauth2/device/verify로 이동하여 코드 입력: ABCD-1234").
4. 사용자는 보조 장치에서 인증 URI로 이동하여 `user_code`를 입력합니다.
5. 서버는 `user_code`를 검증하고 장치에 권한을 부여하기 위해 **OnOAuth2DeviceCodeVerification**을 발생시킵니다.
6. 한편, 디바이스는 `grant_type=urn:ietf:params:oauth:grant-type:device_code&device_code=...&client_id=...`로 토큰 엔드포인트를 폴링합니다
7. 서버는 사용자가 권한을 부여할 때까지 `authorization_pending`을 반환합니다. 권한이 부여되면 액세스 토큰을 반환합니다.

구성

Property	설명
<code>OAuth2Options.DeviceAuthorization.Enabled</code>	장치 권한 부여 엔드포인트를 활성화하려면 True로 설정하십시오. 기본값: False.
<code>OAuth2Options.DeviceAuthorization.URL</code>	장치 권한 부여 엔드포인트 URL 경로입니다. 기본값: <code>/sgc/oauth2/device</code>
<code>OAuth2Options.DeviceAuthorization.VerificationURL</code>	사용자가 코드를 입력하는 verification 페이지 URL 경로입니다. 기본값: <code>/sgc/oauth2/device/verify</code>
<code>OAuth2Options.DeviceAuthorization.ExpiresIn</code>	디바이스 코드의 수명(초)입니다. 기본값: 600(10분).
<code>OAuth2Options.DeviceAuthorization.Interval</code>	장치가 사용해야 하는 최소 폴링 간격(초)입니다. 기본값: 5.

Events

Event	설명
<code>OnOAuth2DeviceAuthorization</code>	장치가 device code를 요청할 때 발생합니다. 응답을 사용자 지정하거나 요청을 로깅할 수 있습니다.

<code>OnOAuth2DeviceCodeVerification</code>	사용자가 장치 검증 페이지에서 검증 코드를 제출할 때 발생합니다. <code>user_code</code> 를 유효성 검사하고 장치를 승인하거나 거부하십시오.
<code>OnOAuth2AfterAccessToken</code>	서버가 장치에 액세스 토큰을 발급한 후 발생합니다.

예제

```
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);
}
```

폴링 동작

사용자가 아직 장치를 인증하지 않은 동안, 장치는 구성된 간격으로 토큰 엔드포인트를 폴링합니다. 서버는 다음으로 응답합니다:

- `authorization_pending` -- 사용자가 아직 권한 부여를 완료하지 않았습니다.
- `slow_down` -- 디바이스가 너무 자주 폴링하고 있습니다. 디바이스는 간격을 5초 늘려야 합니다.
- `expired_token` -- 디바이스 코드가 만료되었습니다. 디바이스는 흐름을 다시 시작해야 합니다.
- `access_denied` -- 사용자가 권한 부여를 거부했습니다.

사용자가 디바이스를 승인하면 다음 poll에서 access token이 반환됩니다.

OAuth2 Server | Token Revocation (RFC 7009)

개요

토큰 취소를 사용하면 클라이언트가 이전에 발급된 액세스 토큰 또는 갱신 토큰이 더 이상 필요하지 않음을 [TsgcHTTP_OAuth2_Server](#)에 알릴 수 있습니다. 서버는 토큰을 무효화하여 보호된 리소스에 액세스하는 데 더 이상 사용할 수 없도록 합니다.

이는 사용자가 로그아웃하거나, 애플리케이션이 제거되거나, 토큰이 손상되었을 수 있을 때 유용합니다.

Endpoint

다음 매개변수와 함께 `/sgc/oauth2/revoke`에 POST합니다:

매개변수	설명
<code>token</code>	취소할 토큰입니다(필수).
<code>token_type_hint</code>	토큰 유형에 대한 선택적 hint입니다: <code>access_token</code> 또는 <code>refresh_token</code> .

Behavior

[RFC 7009](#)에 따라, 서버는 토큰을 찾았는지 또는 성공적으로 취소했는지 여부와 관계없이 항상 HTTP 200 OK를 반환합니다. 이는 토큰 존재 누출을 방지합니다. 즉, 클라이언트는 응답 상태를 관찰하여 토큰이 유효한지 여부를 판단할 수 없습니다. 갱신 토큰이 취소되면 연결된 액세스 토큰도 무효화될 수 있습니다(구현에 따라 다름). 액세스 토큰이 취소되면 명시적으로 취소되지 않는 한 연결된 갱신 토큰은 유효한 상태로 유지됩니다.

구성

Property	설명
<code>OAuth2Options.Revocation.Enabled</code>	폐기 엔드포인트를 활성화하려면 True로 설정하십시오. 기본값: False.
<code>OAuth2Options.Revocation.URL</code>	revocation 엔드포인트 URL 경로입니다. 기본값: <code>/sgc/oauth2/revoke</code>

Events

Event	설명
<code>OnOAuth2AfterRevokeToken</code>	토큰 취소 시도 후 발생합니다. 토큰 값, <code>token_type_hint</code> 및 토큰이 성공적으로 무효화되었는지 나타내는 <code>Revoked</code> 매개변수를 제공합니다. 취소 활동을 기록하는 데 유용합니다.

예제

```
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;
}
```

Client Request 예제

클라이언트는 revocation 엔드포인트로 POST 요청을 보내 토큰을 취소합니다:

```
// 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)

개요

Token introspection을 사용하면 리소스 서버가 [TsgcHTTP_OAuth2_Server](#)를 쿼리하여 액세스 토큰 또는 refresh 토큰의 활성 상태 및 메타데이터를 확인할 수 있습니다. 이는 리소스 서버가 토큰을 로컬에서 디코딩하지 않고 검증해야 할 때 유용합니다.

Endpoint

다음 매개변수와 함께 `/sgc/oauth2/introspect`로 POST하십시오:

매개변수	설명
<code>token</code>	introspect할 토큰입니다(필수).
<code>token_type_hint</code>	토큰 유형에 대한 선택적 hint입니다: <code>access_token</code> 또는 <code>refresh_token</code> .

응답

introspection 엔드포인트는 다음 필드가 있는 JSON 객체를 반환합니다:

필드	설명
<code>active</code>	Boolean입니다. 토큰이 현재 활성 상태(유효하며 만료되거나 폐기되지 않음)이면 True입니다.
<code>scope</code>	토큰과 연결된 범위.
<code>client_id</code>	토큰을 요청한 애플리케이션의 클라이언트 식별자입니다.
<code>token_type</code>	토큰의 유형입니다(예: Bearer).
<code>exp</code>	토큰의 만료 시간(Unix 타임스탬프)입니다.
<code>username</code>	토큰과 연결된 사용자 이름(해당하는 경우)입니다.

토큰이 유효하지 않거나, 만료되었거나, 폐기된 경우 응답에는 `{"active": false}`만 포함됩니다.

구성

Property	설명
<code>OAuth2Options.Introspection.Enabled</code>	introspection 엔드포인트를 활성화하려면 True로 설정하십시오. 기본값: False.

`OAuth2Options.Introspection.URL`

introspection 엔드포인트 URL 경로입니다. 기본값: /sgc/oauth2/introspect

Events

Event	설명
<code>OnOAuth2AfterIntrospectToken</code>	token introspection 요청 후 발생합니다. 토큰 값과, 기본 검증 결과를 재정의하도록 수정할 수 있는 <code>IsActive</code> 매개변수를 제공합니다. 사용자 정의 검증 로직 및 감사에 유용합니다.

예제

```
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));
}
```

Resource Server Request 예제

리소스 서버는 introspection 엔드포인트에 POST 요청을 보내 토큰을 검증합니다:

```
// 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)

개요

RFC 9449에 따른 DPoP(Demonstrating Proof of Possession)는 [TsgcHTTP_OAuth2_Server](#)가 발신자 제약 토큰을 요구할 수 있게 합니다. DPoP가 활성화되면 서버는 DPoP proof JWT를 검증하고, 토큰을 클라이언트의 JWK thumbprint에 바인딩하며, Bearer 대신 `token_type: DPoP`를 반환합니다.

이는 액세스 토큰이 원래 이를 획득한 클라이언트에서만 사용될 수 있도록 보장하여 토큰 도용 및 재생 공격을 방지합니다.

작동 방식

- 클라이언트는 토큰 요청에 서명된 JWT 증명이 포함된 DPoP 헤더를 포함합니다.
- 서버가 proof JWT를 검증합니다:
 - `typ`은 반드시 `dpop+jwt`여야 합니다
 - 알고리즘(`alg`)은 비대칭이어야 합니다(예: ES256, RS256)
 - `htm`(HTTP 메서드)과 `htu`(HTTP URI)는 요청과 일치해야 합니다
 - `iat`(발급 시간)는 최근이어야 합니다(신선도 검사)
 - 서명은 proof 헤더에 임베디드된 JWK에 대해 검증됩니다
- 서버는 JWK thumbprint를 추출하여 발급된 토큰에 바인딩합니다.
- 토큰 응답에 Bearer 대신 `token_type: DPoP`가 포함됩니다.
- 후속 리소스 요청 시, 서버는 증명이 올바른 토큰에 바인딩되어 있는지 확인하기 위해 `ath` 클레임(액세스 토큰 해시)을 포함한 DPoP 증명을 검증합니다.

구성

Property	설명
<code>OAuth2Options.DPoP</code>	Boolean. DPoP에 바인딩된 토큰 검증을 활성화하려면 True로 설정하십시오. 기본값: False.

Signature Verification

서버는 DPoP 증명 헤더에 포함된 JWK에서 EC 또는 RSA 공개 키를 재구성하고 JWT 서명을 검증합니다. 비대칭 알고리즘(ES256, ES384, ES512, RS256, RS384, RS512, PS256, PS384, PS512)만 수락됩니다. 대칭 알고리즘(HS256 등)은 거부됩니다.

Nonce 지원

서버는 토큰 응답에서 `DPoP-Nonce` 헤더를 발행할 수 있습니다. 존재하는 경우, 클라이언트는 후속 DPoP 증명 JWT의 `nonce` 클레임에 `nonce` 값을 포함해야 합니다. 이는 추가적인 재생 보호 계층을 제공합니다.

Events

Event	설명
<code>OnOAuth2ValidatedDPoP</code>	표준 DPoP 검증이 완료된 후에 발생합니다. 추가 클레임 확인이나 정책 시행 같은 사용자 정의 검증 로직을 허용합니다. proof를 거부하려면 <code>valid</code> 를 False로 설정하십시오.

예제

```
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::OAuth2ServerOAuth2ValidatedDPoP(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

DPoP 증명 검증이 실패하면 서버는 오류 응답과 함께 HTTP 400을 반환합니다:

```
{"error": "invalid_dpop_proof", "error_description": "..."}

```

거부의 일반적인 이유는 다음과 같습니다:

- DPoP 헤더가 없거나 잘못 형성됨.
- 잘못된 JWT 서명.
- 사용된 대칭 알고리즘입니다(비대칭 알고리즘만 허용됨).
- htm 또는 htu 클레임 불일치.
- 만료되었거나 미래의 iat 클레임.
- 리소스 요청에서 ath(액세스 토큰 해시)가 일치하지 않습니다.
- 서버가 요구할 때 nonce가 잘못되었거나 누락되었습니다.

TsgcHTTP_OAuth2_Server_Provider

이 구성 요소를 사용하면 외부 OAuth2 공급자(Azure AD, Google, Facebook 등)를 통합할 수 있습니다.

소개

이 구성 요소를 사용하면 서버 구성 요소(예: HTTP 서버)에 외부 OAuth2 공급자(Azure AD, Google, Facebook 등)를 통합할 수 있으므로, 사용자가 Azure AD 자격 증명을 사용하여 로그인할 수 있고 인증이 성공하면 HTTP 서버가 보호된 리소스에 대한 액세스를 제공할 수 있습니다.

서버 구성 요소에는 Authorization.OAuth.OAuth2Provider라는 속성이 있어 TsgcHTTP_OAuth2_Server_Provider의 인스턴스를 할당할 수 있습니다. 따라서 Authentication이 활성화되어 있고 OAuth2Provider 속성이 OAuth2 Provider Server 구성 요소에 연결되어 있으면, WebSocket 및 HTTP 요청은 처리되기 위해 Cookie / Bearer Token이 필요하며, 그렇지 않으면 연결이 자동으로 닫힙니다.

```
TsgcHTTP_OAuth2_Server_Provider *OAuth2Provider = new TsgcHTTP_OAuth2_Server_Provider(this);
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2Provider = OAuth2Provider;
```

OAuth2 Provider 등록

서버를 시작하기 전에 서버가 인증에 사용할 OAuth2 Provider를 구성해야 합니다. OAuth2 Provider를 구성하려면 RegisterProvider 메서드를 사용하십시오. 이 메서드에는 다음 매개변수가 있습니다.

- **Name:** 공급자의 이름이며, 임의의 이름일 수 있고, 나중에 공급자를 식별하기 위한 것일 뿐입니다.
- **ClientId:** 공개 client Id이며, 이 값은 OAuth2 Provider가 제공합니다.
- **ClientSecret:** 비공개 클라이언트 시크릿(기밀로 유지해야 함). 이 값은 OAuth2 공급자가 제공합니다.
- **AuthorizeURL:** OAuth2 클라이언트가 로그인을 위해 리디렉션할 URL입니다(연결은 웹 브라우저를 사용함).
- **TokenURL:** 성공적인 권한 부여 후 제공된 토큰을 유효성 검사하기 위해 서버가 사용할 URL입니다(연결은 서버 대 서버입니다).
- **Scope:** 범위의 값입니다.
- **URL:** Authorization URL로 리디렉션하는 데 사용될 HTTP Server의 URL입니다.
- **CallbackURL:** 성공적인 Authorization 후 OAuth2 서버에서 전송된 응답을 처리하는 OAuth2 Provider에 구성된 URL입니다.

예제: Azure AD를 구성하려면 OAuth2 URL에 추가되는 tenant-id와 ClientId, ClientSecret, Scope 및 CallbackURL이 필요합니다.

```
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'
);
```

기존 Provider를 삭제하려면 UnRegisterProvider 메서드를 사용하십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 2개의 속성입니다.
- 메서드 — 15개의 public 메서드.
- 이벤트 — 처리할 수 있는 4개의 이벤트.

Guides

- **QuickStart**
 - [OAuth2 Provider Azure AD](#)
- **Authenticate**
 - [OAuth2 Provider Private Endpoints](#)
 - [OAuth2 Provider Authentication](#)
 - [OAuth2 Provider Requests](#)

OAuth2 Provider | Azure AD

Azure AD는 다음 OAuth2 Authorization URL을 사용합니다

Authorization: <https://login.microsoftonline.com/<tenant-id>/oauth2/v2.0/authorize>

Token: <https://login.microsoftonline.com/<tenant-id>/oauth2/v2.0/token>

<tenant-id>는 사용자 고유의 값으로 교체해야 합니다.

OAuth2 구성을 생성할 때, 서버 callback url을 구성해야 합니다. 이 url은 성공적인 권한 부여 후 Azure가 서버에 응답을 보내는 데 사용됩니다.

예: Azure AD 공급자를 등록하는 방법에 대한 간단한 예제를 아래에서 확인하십시오.

Azure AD가 제공하는 값

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

Azure AD를 등록하는 방법

```
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

서버가 HTTP 요청을 수신할 때마다 **OnOAuth2IsPrivateEndpoint** 이벤트가 호출되어 Endpoint가 비공개인지 여부를 묻습니다. 기본적으로 비공개가 아닙니다.

```
void OnOAuth2IsPrivateEndpoint(TObject *Sender, const string aEndpoint, ref bool IsPrivate)
{
    if (aEndpoint == "/private")
    {
        IsPrivate = True;
    }
}
```

OAuth2 Provider | Authentication

OAuth2 Provider Server 구성 요소를 사용하면 외부 OAuth2 공급자(예: Azure AD 또는 Google)를 사용하여 인증하여 서버의 보호된 리소스에 접근할 수 있습니다. **예제:** 사용자가 Azure 자격 증명을 사용하여 로그인할 수 있도록 HTTP 서버를 구성할 수 있습니다. 로그인이 성공하면 해당 사용자는 서버의 보호된 리소스에 접근할 수 있습니다.

인증 프로세스는 서버 측에서 수행되며 OAuth2 토큰은 클라이언트와 공유되지 않습니다. 즉, 사용자가 예를 들어 Azure를 사용하여 로그인할 때 인증이 성공하면 Azure는 사용자 프로필, 이메일 등에 대한 일부 정보(scope에 따라 다름)를 얻기 위해 Azure 서버에 요청을 보낼 수 있는 Access Token을 반환합니다. 이 Access Token은 클라이언트(예: 웹 브라우저)와 **공유되지 않습니다**. 서버는 Access Token을 클라이언트에 반환하는 대신 Access Token과 내부적으로 연결된 임의의 ID를 생성합니다. 따라서 클라이언트(웹 브라우저)가 OAuth2 서버에 호출을 하려고 할 때마다 public ID를 사용하고, 서버는 이 ID를 사용하여 HTTP 요청을 프록시하기 위한 OAuth2 Access Token을 얻습니다.

아래에서 OAuth2 Authentication이 작동하는 방식의 예를 확인하십시오. 이 예제는 다음 링크 [OAuth2 Provider Azure AD](#)에 설명된 Azure AD 구성을 사용합니다.

서버 시작

서버는 localhost와 포트 443에서 수신을 시작합니다. sgcWebSockets HTTP Server는 OAuth2 Server Provider Component에 연결되어 있고 Authentication 속성이 활성화되어 있습니다.

서버가 시작되기 전에 다음 메서드 호출을 사용하여 Azure OAuth2 Provider가 등록됩니다.

```
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'
);
```

사용자 로그인

사용자가 새 웹 브라우저를 열고 '/login' 엔드포인트로 이동합니다.

서버는 '/login' 엔드포인트가 Azure 공급자를 사용하여 로그인하는 데 사용됨을 감지하므로 다음으로 리디렉션합니다

<https://login.microsoftonline.com/a0ca2055-5dd1-467f-bf13-291f6fd715c6/oauth2/v2.0/authorize>

그리고 OAuth2 인증 흐름이 시작됩니다.

OAuth2 Authentication

사용자는 OAuth2 Server Authentication Endpoint로 리디렉션되며, 이제 자격 증명을 사용하여 로그인하고 OAuth2 애플리케이션의 약관을 수락해야 합니다.

권한 부여가 성공하면 Azure AD가 url로 Code를 보냅니다

<https://localhost/callback>

OAuth2 Code 검증

이제 서버는 Azure로부터 코드를 수신했으며, 이 토큰이 올바른지 검증하기 위해 Azure에 내부 연결(서버에서 서버로)을 수행합니다(누군가 서버를 해킹하려는 것을 방지하기 위해).

서버가 연결하는 대상:

<https://login.microsoftonline.com/a0ca2055-5dd1-467f-bf13-291f6fd715c6/oauth2/v2.0/token>

수신된 code 및 clientsecret과 같은 일부 매개변수를 전달하여 검증이 성공하면 Azure는 프로필 읽기, 이메일 등 Azure 보호 리소스에 액세스하는 데 사용할 수 있는 Access Token을 반환합니다.

성공적인 Access Token

서버가 성공적인 AccessToken을 수신하면, **OnOAuth2ProviderTokenValid** 이벤트가 호출되므로, 여기서 매개변수 클래스 `TsgcHTTPOAuth2ProviderToken`에 액세스하여 AccessToken이 저장되는 방법(저장되는 경우)을 구성할 수 있습니다

AccessToken: Azure가 반환한 OAuth2 Token입니다.

ID: 쿠키로 저장되는 공개 식별자입니다.

이 이벤트에서 성공적인 인증 후 수행할 작업을 구성할 수 있습니다. 예: 사용자를 비공개 url로 리디렉션하려면 다음을 사용하십시오

```
Response.Redirect.URL := 'https://localhost/private';
```

Azure에 요청 보내기

이제 쿠키로 저장된 Public ID를 사용하여 Azure 서버에 요청을 보낼 수 있습니다.

예: 프로필 데이터를 읽으려면 다음 메서드를 사용하십시오.

```
Get('ID', 'https://graph.microsoft.com/v1.0/me');
```

여기서 ID는 public ID 식별자입니다.

OAuth2 Provider | Requests

인증이 성공하면 쿠키로 저장된 Public ID Token을 사용하여 OAuth2 Protected Server에 요청을 보낼 수 있습니다.

OAuth2 Provider Server Component에는 GET, POST, DELETE... 등 HTTP 요청을 보내는 여러 메서드가 있습니다.

Token을 매개변수로 전달하거나, Indy Server 구성 요소를 사용하는 경우 RequestInfo 클래스를 전달할 수 있습니다.

```
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

JWT(JSON Web Token)는 일반적으로 header + payload + signature로 구성됩니다.

Header

JWT에 대한 메타데이터 정보를 포함합니다.

- **alg**: 토큰에 서명하는 데 사용되는 알고리즘입니다.
- **typ**: 토큰의 유형이며, 항상 JWT입니다.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

더 많은 헤더를 찾을 수 있지만, 이전 헤더들은 항상 존재합니다.

Payload

페이로드에는 JWT의 클레임이 포함됩니다. 표준 헤더는 다음과 같습니다:

- **iss**: 발급자, JWT를 생성하고 발급하는 엔터티입니다.
- **sub**: 주체, 즉 이 토큰으로 식별되는 엔터티입니다.
- **aud**: audience, 이 JWT의 대상 청중입니다.
- **exp**: 만료입니다. 이 타임스탬프(UNIX 형식) 이후에는 토큰이 수락되어서는 안 됩니다.
- **iat**: issued at. 토큰이 발급된 날짜를 지정합니다.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

서명

서명은 Encoded Header, Encoded Payload, Secret, 그리고 Cryptographic Algorithm을 사용하여 생성됩니다.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4qRG9lIiwiaWF0IjoxNTE2MjM5M

DIyf0.Sf1KxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_ad0ssw5c

지원되는 알고리즘

다음 알고리즘은 Client 및 Server JWT 구성 요소 모두에서 지원됩니다.

- HS256
- HS384
- HS512
- RS256
- RS384
- RS512
- ES256
- ES384
- ES512

JWT를 서명하고 검증하려면 **OpenSSL** 라이브러리가 필요합니다.

구성 요소

- **TsgcHTTP_JWT_Client**: JWT를 인코딩 및 서명하고 **HTTP** 및 **WebSocket** 프로토콜에서 Authorization 헤더로 전송할 수 있는 JWT 클라이언트입니다.
- **TsgcHTTP_JWT_Server**: **HTTP** 및 **WebSocket** 프로토콜에서 Authorization Header로 수신된 JWT를 디코딩하고 검증할 수 있는 JWT 서버입니다.

** JWT 구성 요소는 최소 Indy 버전 10.6.0.5169 또는 sgcWebSockets Enterprise Edition을 필요로 합니다.*

TsgcHTTP_JWT_Client

JWT 클라이언트를 사용할 때 사용할 openssl 라이브러리를 구성하십시오.

소개

TsgcHTTP_JWT_Client 구성 요소를 사용하면 JWT Tokens를 인코딩하고 서명하여, [WebSocket Client](#) 또는 [HTTP/2 client](#)에 첨부할 수 있습니다. 토큰은 Authorization Bearer Token Header로 자동 전송됩니다.

구성

JWTOptions 속성에서 JWT 값을 구성할 수 있습니다. 2개의 주요 속성이 있습니다: **Header** 와 **Payload**. 필요한 각 속성에 대해 값을 설정하기만 하면 됩니다.

Signature가 Private Key(RS 및 ES 알고리즘)를 사용하여 암호화된 경우, Algorithm의 **PrivateKey** 속성에 값을 설정하십시오.

Signature가 Secret(HS 알고리즘)을 사용하여 암호화된 경우, Algorithm의 **Secret** 속성에 값을 설정하십시오.

OpenSSL Options

JWT 클라이언트를 사용할 때 사용할 openssl 라이브러리를 구성하십시오.

OpenSSL_Options: openssl 라이브러리의 구성입니다.

APIVersion: 어떤 OpenSSL API를 사용할지 정의할 수 있게 합니다.

oslAPI_1_0: API 1.0 OpenSSL을 사용하며, Indy가 지원하는 최신 버전입니다

oslAPI_1_1: API 1.1 OpenSSL을 사용하며, 사용자 정의 Indy 라이브러리가 필요하고 OpenSSL 1.1.1 라이브러리(TLS 1.3 지원 포함)를 사용할 수 있게 합니다.

oslAPI_3_0: API 3.0 OpenSSL을 사용하며, 사용자 정의 Indy 라이브러리가 필요하고 OpenSSL 3.0.0 라이브러리(TLS 1.3 지원 포함)를 사용할 수 있게 합니다.

LibPath: 여기에서 openssl 라이브러리가 위치한 곳을 구성할 수 있습니다.

oslpNone: 이것이 기본값이며, openssl 라이브러리는 바이너리가 있는 동일한 폴더 또는 알려진 경로에 있어야 합니다.

oslpDefaultFolder: 모든 IDE 퍼스널리티에 대해 라이브러리가 위치해야 하는 openssl 경로를 자동으로 설정합니다.

oslpCustomFolder: 이 옵션이 선택된 경우, LibPathCustom 속성에 전체 경로를 정의하십시오.

LibPathCustom: LibPath = oslpCustomFolder일 때 openssl 라이브러리가 있는 전체 경로를 여기에 정의하십시오.

UnixSymLinks: Unix 시스템에서 SymLinks 로딩을 활성화하거나 비활성화합니다(기본적으로 활성화되어 있으며, OSX64에서는 예외):

oslsSymLinksDefault: 기본적으로 활성화되지만 OSX64에서는 예외입니다(MacOS Monterey 이후 버전 없이 라이브러리를 로드하려고 하면 실패함).

oslsSymLinksLoadFirst: 버전 라이브러리를 로드하기 전에 SymLink를 로드하고 먼저 시도합니다.

oslsSymLinksLoad: 버전 라이브러리를 로드하려고 시도한 후 SymLinks를 로드합니다.

oslsSymLinksDontLoad: SymLinks를 로드하지 않습니다.

Custom Headers

Header 및 Payload 속성에는 JWT를 생성하는 데 사용되는 가장 일반적인 헤더가 포함되어 있지만, **AddKeyValue** 메서드를 호출하고 Key와 Value를 매개변수로 전달하여 더 많은 헤더를 추가할 수 있습니다.

예제: JWT Header에 이름이 있는 새 레코드를 추가하려면 다음 메서드를 사용하십시오

```
Header->AddKeyValue("name", "John Smith");
```

속성을 구성한 후 JWT를 생성하려면 **Sign** 메서드를 호출하기만 하면 JWT의 값을 반환합니다.

WebSocket Client and JWT

[TsgcWebSocketClient](#)는 WebSocket 서버에 연결할 때 JWT 사용을 허용합니다. 새 JWT 클라이언트를 생성하여 **Authentication.Token.JWT** 속성에 할당하기만 하면 됩니다.

```
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;
```

HTTP Clients and JWT

[TsgcHTTP2Client](#) 및 [TsgcHTTP1Client](#)는 HTTP/2 서버에 연결할 때 JWT를 사용할 수 있습니다. 새 JWT 클라이언트를 만들고 **Authentication.Token.JWT** 속성에 할당하기만 하면 됩니다.

```
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");
```

만료

Authorization Token은 HTTP 클라이언트를 사용하여 HTTP 요청을 보낼 때마다 **매번 다시 생성되거나 만료될 때까지 여러 번 재사용**될 수 있습니다.

예제: 토큰을 사용하여 Apple APNs를 호출하려면 토큰이 최소 20분 동안 최대 1시간 동안 재사용되어야 합니다. **RefreshTokenAfter** 속성을 사용하여 토큰이 만료되는 초를 설정하십시오. 예를 들어 30분 후입니다.

```
RefreshTokenAfter = 60 * 40.
```

JWT 서명 생성

WebSocket 또는 HTTP Protocol을 사용하지 않는 애플리케이션에서 사용하거나, 서드파티 애플리케이션의 구성 요소를 사용하고 JWT Token만 필요한 경우 **JWT Signature**를 수동으로 만들 수 있습니다.

JWT Signature를 얻으려면 JWT 클라이언트의 새 인스턴스를 만들고 속성을 수동으로 채운 다음, 모든 속성이 설정되면 **Sign** 메서드를 호출하면 JWT Token을 반환합니다.

```
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());
```

참조

- 속성 — 구성 요소에서 사용 가능한 2개의 속성.
- 메서드 — 2개의 public 메서드.
- 이벤트 — 처리할 수 있는 이벤트 0개.

TsgcHTTP_JWT_Server

TsgcHTTP_JWT_Server 구성 요소.

소개

TsgcHTTP_JWT_Server 구성 요소를 사용하면 WebSocket 프로토콜을 사용할 때 **WebSocket Handshake**에서 또는 HTTP 프로토콜을 사용할 때 **HTTP Header**로 수신된 JWT 토큰을 디코딩하고 검증할 수 있습니다.

구성

구성 요소의 **JWTOptions** 속성에서 다음 속성을 구성할 수 있습니다:

서명이 공개 키(RS 및 ES 알고리즘)를 사용하여 검증되는 경우, 알고리즘의 **PublicKey** 속성에 값을 설정하십시오.
서명이 Secret(HS 알고리즘)을 사용하여 검증되는 경우 Algorithm의 **Secret** 속성에 값을 설정하십시오.

JWT 토큰을 검증하려면 **TsgcHTTP_JWT_Server** 인스턴스를 WebSocket/HTTP Server의 **Authentication.JWT.JWT** 속성에 첨부하기만 하면 됩니다.

```
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 속성을 사용하면 JWT의 Payload에서 일부 검사를 활성화할 수 있으며, 기본적으로 발급 날짜가 유효한지 확인합니다.

참조

- 속성 — 구성 요소에서 사용 가능한 2개의 속성.
- 메서드 — 3개의 public 메서드입니다.
- 이벤트 — 처리할 수 있는 7개의 이벤트.

WebAuthn

WebAuthn (Web Authentication)은 웹에서 **안전한, 비밀번호 없는 인증**을 가능하게 하기 위해 World Wide Web Consortium(W3C)과 FIDO Alliance가 개발한 웹 표준입니다. 이는 더 넓은 FIDO2 프레임워크의 일부이며, 피싱, credential stuffing 및 기타 공격에 종종 취약한 **전통적인 비밀번호에 대한 의존도를 줄이는 것**을 목표로 합니다.

본질적으로 **WebAuthn**은 사용자가 **공개 키 암호화를 사용하여 인증할 수 있게 합니다**. 사용자 이름과 비밀번호 대신 사용자는 **웹 애플리케이션(Relying Party)**에 고유한 공개-개인 키 쌍을 등록합니다. 개인 키는 하드웨어 보안 키, 스마트폰 또는 내장 생체 인식 디바이스와 같은 authenticator에 안전하게 저장되고, 공개 키는 서버에 저장됩니다.

인증 중에, 서버는 사용자의 private key로 서명되어야 하는 challenge를 발급합니다. 서명된 challenge가 반환되고 저장된 public key를 사용하여 검증되어, 응답의 무결성과 출처를 모두 보장합니다. 이 접근 방식은 자격 증명이 가로채이거나 재사용되는 것을 방지합니다.

WebAuthn은 다양한 authenticator와 디바이스를 지원하므로 개발자와 사용자 모두에게 유연합니다. 또한 PIN 또는 생체 인식과 같은 다른 요소와 결합할 때 multi-factor authentication(MFA)을 가능하게 하여 사용성을 희생하지 않고 보안을 크게 향상시킵니다.

Server

- **TsgcWSAPIServer_WebAuthn**: 이 구성 요소는 WebAuthn Relying Party 서버를 구현하기 위한 간단하지만 강력한 솔루션을 제공하여 웹 애플리케이션에서 비밀번호 없는 인증을 가능하게 합니다. 서버는 Conformance Self-Validation Testing 도구를 사용하여 전체 **FIDO Conformance Test**를 통과했습니다.

클라이언트

- **Javascript Client**: 등록 및 인증 중에 사용자, 브라우저, 인증자 간의 브리지 역할을 합니다

TsgcWSAPIServer_WebAuthn

OpenSSL 3만 해당.

소개

TsgcWSAPIServer_WebAuthn 구성 요소는 WebAuthn Relying Party 서버를 구현하기 위한 간단하지만 강력한 솔루션을 제공하여 웹 애플리케이션에서 비밀번호 없는 인증을 가능하게 합니다. WebAuthn 애플리케이션은 서버 측 등록 및 인증을 처리하는 WebAuthn 서버와 일반적으로 javascript 애플리케이션인 클라이언트 측 애플리케이션으로 구성됩니다.

WebAuthn은 보안 연결(SSL/TLS)의 사용을 요구하므로 [OpenSSL](#) 라이브러리를 배포하고 서버와 함께 구성해야 합니다.

OpenSSL 3.0.0+ API만 지원됩니다. 따라서 이전 OpenSSL 버전은 작동하지 않을 수 있습니다.

구성

TsgcWSAPIServer_WebAuthn은 Server 속성을 사용하여 HTTP 서버, [TsgcWebSocketHTTP_Server](#) 또는 [TsgcWebSocketServer_HTTPAPI](#)에 연결되어야 합니다. 등록 및 인증 옵션을 처리할 서버 엔드포인트와, 지원되는 알고리즘, origin 등의 WebAuthn 옵션을 구성할 수 있습니다.

Endpoints Options

여기에서 WebAuthn을 인증자로 사용하기 위한 HTTP/JavaScript 요청을 처리할 서버 엔드포인트를 구성할 수 있습니다. 구성 요소는 이미 기본 엔드포인트로 구성되어 있지만, 필요에 맞게 모두 변경할 수 있습니다.

- **AuthenticationOptions:** 기본값은 /sgcWebAuthn/Authentication/Options입니다.
- **AuthenticationVerify:** 기본값은 /sgcWebAuthn/Authentication/Verify입니다.
- **RegistrationOptions:** 기본값은 /sgcWebAuthn/Registration/Options입니다.
- **RegistrationVerify:** 기본값은 /sgcWebAuthn/Registration/Verify입니다.
- **Webauthn:** 기본적으로 사용되는 javascript 라이브러리를 포함합니다. 이 속성을 비활성화하고 자체 webauthn 라이브러리를 사용할 수 있습니다.
- **Test:** 기본적으로 비활성화되어 있으며, WebAuthn 기능을 테스트하는 데만 사용하십시오.

예: 서버가 도메인 [www.test.com](#)에서 수신 중인 경우, 기본적으로 인증 옵션에 대한 요청은 [http://www.test.com/sgcWebAuthn/Authentication/Options](#)가 됩니다.

WebAuthn Options

이 속성에서 WebAuthn 서버 구성 요소의 주요 옵션을 구성할 수 있습니다.

- **RelyingParty:** 서버의 DNS 이름을 정의해야 하는 필수 속성입니다. 예: 서버가 [www.test.com](#) 도메인에서 실행 중인 경우 이 속성을 "www.test.com"으로 설정하십시오.

WebAuthn은 origin을 사용하여 same-origin 정책 제약 조건을 적용하며, 이는 피싱 및 cross-site 공격을 방지하는 데 필수적입니다. WebAuthn 등록 및 인증 프로세스 중에 origin은 브라우저와 authenticator에 의해 엄격하게 검증됩니다.

- **Origins:** 요청이 다른 origin에서 올 수 있는 경우, Origin 속성을 사용하여 추가 origin을 설정하십시오. 예제: 요청이 [login.test.com](#)과 [www.test.co.uk](#)에서 올 수 있는 경우, Origins 속성을 [https://login.test.com](#) 및 [https://www.test.co.uk](#) 값으로 구성하십시오.
- **TopOrigins:** 일반적으로 WebAuthn은 호출 프레임(즉, `navigator.credentials.create()` 또는 `navigator.credentials.get()`을 호출하는 프레임)의 origin에 의존합니다. 그러나 웹 페이지는 최상위 페이지와 다른

origin에서 올 수 있는 iframe에 포함될 수 있습니다. 이는 악용이나 클릭재킹 스타일 공격의 가능성을 열어줍니다. 이를 완화하기 위해 WebAuthn Level 2 사양은 TopOrigins를 정의할 수 있는 TopOrigin을 도입합니다.

WebAuthn에서 crossOrigin은 WebAuthn 작업이 최상위 브라우징 컨텍스트와 다른 origin에서 임베드된 iframe과 같은 교차 origin 컨텍스트에서 수행되는지 여부를 나타내는 boolean 매개변수입니다.

이 매개변수는 브라우저와 인증자가 임베디드 환경에서 인증 요청을 안전하게 처리하도록 돕기 위해 도입되었으며, 이는 최신 웹 애플리케이션에서 일반적인 시나리오입니다.

- **AllowCrossOrigins:** true인 경우, 크로스 오리진 iframe(예: `https://app.example.org` 페이지에 포함된 `https://auth.example.com`의 iframe)에서 만들어진 요청이 허용됨을 나타냅니다. 기본적으로 비활성화되어 있습니다.

WebAuthn은 공개 키 자격 증명 생성 및 검증을 위한 다양한 **암호화 알고리즘**을 지원합니다. 이 알고리즘들은 자격 증명 등록(`navigator.credentials.create()` 사용) 및 인증(`navigator.credentials.get()` 사용) 중에 사용되며, 비대칭 키 쌍을 사용하여 챌린지의 안전한 서명 및 검증을 보장합니다. 서버는 기본적으로 가장 일반적인 알고리즘인 **ES256** 및 **RS256**로 구성되어 있습니다. **Algorithms** 속성에서 어떤 알고리즘을 지원할지 언제든지 변경할 수 있습니다. 다음 알고리즘이 지원됩니다:

- ES256
- ES384
- ES512
- RS256
- RS384
- RS512
- PS256
- PS384
- PS512
- RS1
- EdDSA

WebAuthn에서 **attestation**은 **선택적 메커니즘**으로, authenticator(예: 디바이스 또는 보안 키)가 자격 증명 생성 중에 제조업체, 모델 및 보안 특성에 대한 정보를 제공할 수 있도록 합니다. 이 정보는 Relying Party(RP)가 authenticator를 신뢰할지 여부를 결정하는 데 도움이 됩니다.

서로 다른 attestation 형식은 이 데이터가 어떻게 구조화되고 검증되는지를 정의합니다. 일반적으로 사용되는 세 가지 형식은 android-key, packed 및 fido-u2f, apple 또는 none과 같은 기타 형식입니다. 기본적으로 모든 attestation 형식이 활성화되어 있습니다. 지원되는 attestation 형식 목록은 아래에서 확인할 수 있습니다.

- **NoneAttestation:** 이 경우 attestation 데이터가 반환되지 않습니다. 디바이스 식별자 노출을 방지하여 사용자 개인 정보를 우선시합니다. 디바이스 출처에 신경 쓰지 않는 애플리케이션에서 일반적입니다.
- **PackedAttestation:** 많은 authenticator가 사용하는 유연하고 간결한 형식입니다. authenticator는 attestation 인증서와 서명을 반환합니다. 다음일 수 있습니다: **Full attestation:** 공급업체가 제공한 키와 cert로 서명된 또는 **Self attestation:** credential private key를 사용하여 서명됨. 다양한 플랫폼에서 가장 널리 사용됩니다(예: YubiKey, Windows Hello).
- **TPMAttestation:** Trusted Platform Module(TPM)이 있는 장치에서 사용됩니다. 증명은 TPM의 키를 사용하여 서명되며 인증서 체인을 포함합니다. TPM 칩이 있는 Enterprise 데스크톱/노트북(예: Windows 머신)에서 사용됩니다.
- **AndroidKeyAttestation:** Android Keystore를 사용하는 Android 디바이스에서 사용됩니다. 키는 하드웨어에서 생성되며, attestation은 디바이스 제조업체가 발급한 인증서 체인으로 서명된 정보를 포함합니다. 하드웨어 지원 keystore(TEE 또는 StrongBox)가 있는 Android 폰에서 사용됩니다.
- **AppleAttestation:** Touch ID 및 Face ID와 같은 Apple 플랫폼 인증자가 사용합니다. Attestation은 Apple의 내부 API에 의해 생성되며 특수 인증서 형식을 포함합니다. Apple 생체 인식을 사용하는 Safari에서 사용됩니다.
- **FidoU2FAttestation:** FIDO U2F 인증자가 사용하는 레거시 증명 형식. U2F 호환 인증서와 서명을 반환합니다. FIDO U2F를 지원하는 구형 보안 키(예: 초기 YubiKey)에서 사용됩니다.

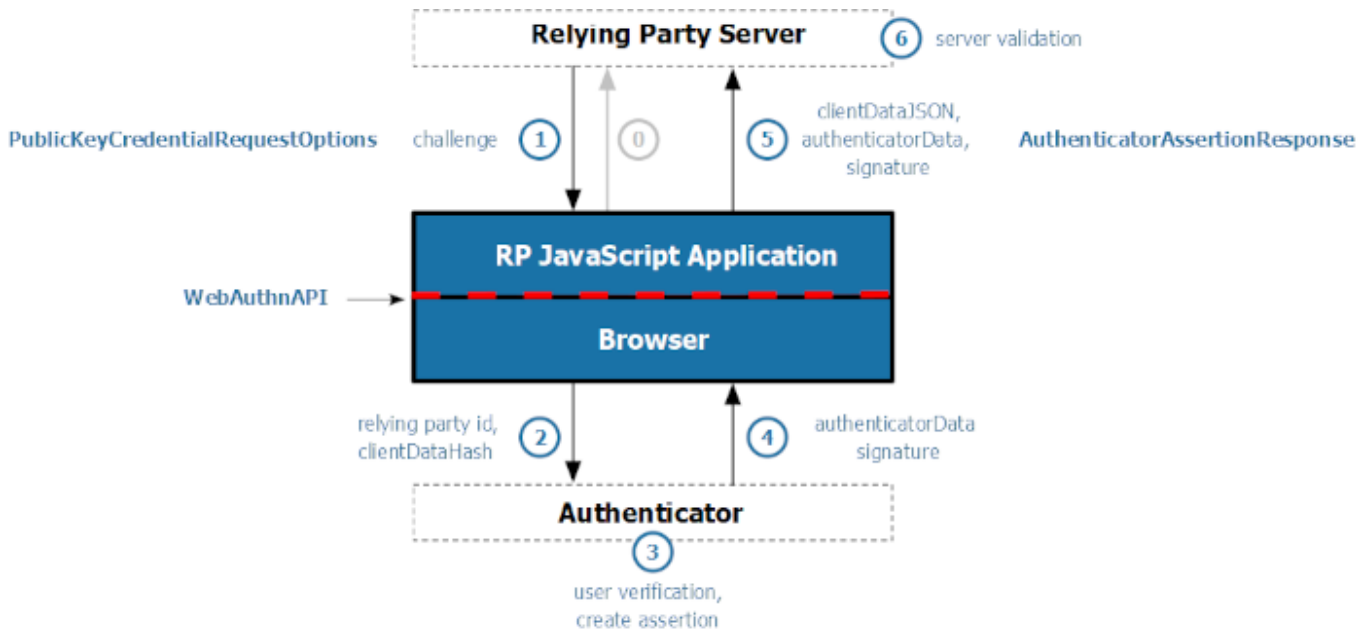
WebAuthn API에서 **AllowCredentials**는 인증 프로세스(`navigator.credentials.get()`을 통한) 중에 사용되는 **선택적 필드**입니다. 이는 특정 Relying Party(RP)에 대해 **사용자를 인증할 수 있도록 허용된 자격 증명 ID 목록을 지정**합니다. 이 메커니즘을 통해 RP는 로그인 시도에 어떤 자격 증명이 유효한 것으로 간주되는지 제어할 수 있습니다. `credentials` 속성에는 다음 필드가 있습니다.

- **AllowCredentials:** 활성화되면(기본적으로 false) 사용자를 인증할 수 있는 자격 증명 ID 목록을 지정합니다
- **ExcludeCredentials:** 사용자 이름이 주어지면 서버 구성 요소에 이미 저장된 모든 기존 자격 증명을 표시합니다.

- **Limit:** ExcludeCredentials가 true일 때 전송될 최대 자격 증명 수입니다.

WebAuthn Protocol

- **WebAuthn Registration:** 서버가 챌린지를 생성하여 클라이언트에 보내면, 클라이언트는 인증자(예: 보안 키 또는 생체 인식 디바이스)를 사용하여 키 쌍을 생성합니다. 공개 키는 다시 전송되어 향후 인증을 위해 서버에 저장됩니다. 아래에서 등록 흐름 이벤트에 대한 자세한 정보를 찾을 수 있습니다:
 - **WebAuthn Registration Request**
 - **WebAuthn Registration Response**
 - **WebAuthn Registration Result**
- **WebAuthn Authentication:** 서버가 클라이언트에 챌린지를 보내면, 클라이언트는 인증자에 저장된 이전에 등록된 개인 키를 사용하여 서명합니다. 서명된 응답은 서버가 저장된 공개 키를 사용하여 검증하여 사용자의 신원을 확인합니다. 아래에서 인증 흐름 이벤트에 대한 자세한 정보를 찾을 수 있습니다:
 - **WebAuthn Authentication Request**
 - **WebAuthn Authentication Response**
 - **WebAuthn Authentication Result**
- **MDS:** FIDO Alliance Metadata Service(MDS)는 relying party가 authenticator attestation을 검증하고 디바이스 모델의 진정성을 증명하는 데 사용하는 Metadata Statement의 중앙 집중식 저장소입니다.



- **Authorization:** 클라이언트는 인증 플로우 중에 서버로부터 bearer 토큰을 요청할 수 있습니다. 이 토큰은 나중에 passkey를 사용하여 로그인할 필요 없이 새 WebSocket 또는 HTTP 연결을 여는 데 사용할 수 있습니다.
 - **WebAuthn Authorization WebSocket**
 - **WebAuthn Authorization HTTP**

참조

- 속성 — 구성 요소에서 사용 가능한 4개의 속성.
- 메서드 — 8개의 public 메서드.
- 이벤트 — 처리할 수 있는 15개의 이벤트.

WebAuthn | Registration

WebAuthn 등록에는 클라이언트 (브라우저), 인증자 (보안 디바이스), **relying party**(RP; TsgcWSAPIServer_WebAuthn delphi/cbuilder 서버)가 관련됩니다.

등록 옵션

새 사용자 자격 증명을 생성하려면 일반적으로 클라이언트가 TsgcWSAPIServer_WebAuthn 서버에 구성된 **Registration Options Endpoint**에 새 HTTP 요청을 사용하여 등록 흐름을 시작해야 합니다. 기본적으로 엔드포인트는 /sgcWebAuthn/Registration/Options 이므로, 서버가 https://www.test.com 도메인에서 수신 대기 중인 경우 클라이언트는 https://www.test.com/sgcWebAuthn/Registration/Options url에 새 요청을 해야 합니다.

클라이언트는 다음 json을 페이로드로 전달하여 새 요청을 보냅니다(사용자 이름으로 test 사용)

```
{"username": "test", "algorithms": []}
```

서버는 요청을 읽고 새로운 챌린지가 포함된 응답을 반환합니다.

```
{
  "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
  }
}
```

응답은 다음 데이터를 반환합니다:

- **challenge**: base64로 인코딩된 무작위 바이너리 데이터.
- **rp**: Relying Party 정보(귀하의 웹 앱/사이트)입니다.
- **user**: 사용자 식별자 및 표시 이름입니다.
- **pubKeyCredParams**: 허용되는 알고리즘(예: ES256, RS256).

- **authenticatorSelection**: 인증자 유형에 대한 환경 설정(플랫폼 또는 크로스 플랫폼, 사용자 검증 옵션)입니다.
- **timeout**: 제안된 시간 초과 값입니다.
- **attestation**: Attestation 전달 환경 설정("none", "direct", "indirect")입니다.

Registration Verify

이제 클라이언트가 서버로부터 응답을 받았고, 응답을 읽으며 인증자가 암호화 데이터를 클라이언트 웹 브라우저에 반환합니다. 이제 클라이언트는 TsgcWSAPIServer_WebAuthn 서버에 구성된 Registration Verify Endpoint(기본 값은 /sgcWebAuthn/Registration/Verify)로 다음 데이터와 함께 새 HTTP 요청을 보냅니다, 따라서 서버가 https://www.test.com 도메인에서 수신 대기 중인 경우 클라이언트는 https://www.test.com/sgcWebAuthn/Registration/Verify url에 새 요청을 해야 합니다.

authenticator가 브라우저의 JavaScript에 다음과 함께 응답합니다.

- **id**: Credential ID(authenticator가 생성한 키 쌍의 고유 ID)입니다.
- **rawId**: Credential ID의 바이너리 형식입니다.
- **type**: 자격 증명 유형("public-key")입니다.
- **response**: attestationObject 및 clientDataJSON을 포함합니다.

아래에서 예제를 확인하십시오:

```
{
  "id": "yeA4BVRlrAfLG-KzqsL_rll4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
  "rawId": "yeA4BVRlrAfLG-KzqsL_rll4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
  "response": {
    "attestationObject": "o2NmbXRmcGFja2VkZ2F0dFN0....",
    "clientDataJSON": "eyJ0eXBlljoid2ViYXV0aG4uY3Jl....",
    "transports": [
      "nfc",
      "usb"
    ],
    "publicKeyAlgorithm": -7,
    "publicKey": "MFkwEwYHKoZIzj0CAQYIKoZIzj....",
    "authenticatorData": "SZYN5YgOjGh0NBcPZHZgW4_k...."
  },
  "type": "public-key",
  "clientExtensionResults": {
    "credProps": {
      "rk": true
    }
  },
  "authenticatorAttachment": "cross-platform"
}
```

서버는 클라이언트로부터 JSON 요청을 읽고 공개 키와 credential ID를 디코딩, 검증 및 저장합니다.

attestationObject와 clientDataJSON을 디코딩합니다:

- attestationObject에서 공개 키를 추출합니다.
- clientDataJSON의 챌린지가 서버가 보낸 원본과 일치하는지 검증합니다.
- 적절한 origin 검증(myapp.example.com)을 보장하십시오.

Store Credential:

- 검증이 성공하면 **OnWebAuthnRegistrationSuccessful** 이벤트가 호출되므로 향후 인증을 위해 credential ID와 추출된 public key를 데이터베이스에 안전하게 저장할 수 있습니다.
- 오류가 있으면 **OnWebAuthnRegistrationError** 이벤트가 호출되며 오류의 세부 정보를 얻을 수 있습니다.

등록 흐름

다음 링크를 사용하여 Registration Flow 프로세스에 대한 자세한 정보를 얻으십시오:

- [WebAuthn Registration Request](#)
- [WebAuthn Registration Response](#)
- [WebAuthn Registration Result](#)

WebAuthn Registration | Request

이 등록 옵션 요청은 WebAuthn 등록을 안전하게 부트스트랩하는 데 필수적입니다. 이 요청은 서버에 다음을 요청합니다:

- challenge를 생성하고,
- 사용자 ID를 조회하거나 프로비저닝하고,
- 유효한 PublicKeyCredentialCreationOptions 객체를 반환합니다(RP, user, algorithms 등 포함).

브라우저는 보안 WebAuthn 자격 증명 등록을 시작하기 위해 서버로부터 자격 증명 생성 옵션을 얻어야 합니다. 이는 일반적으로 브라우저에서 서버 엔드포인트(예: /sgcWebAuthn/Registration/Options)로의 HTTPS POST 요청을 통해 수행되며, challenge, RP ID 및 사용자 정보와 같은 매개변수를 준비합니다.

요청 본문에는 사용자를 식별하고 가능하면 구성 기본 설정을 제공하는 데이터가 포함됩니다. 종종 다음과 같이 보입니다:

```
{
  "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
}
```

WebAuthn Server 구성 요소는 **EndpointOptions.RegistrationOptions** 속성에 구성된 엔드포인트에서 PublicKeyCredentialCreationOptions를 얻기 위한 초기 브라우저 요청을 수신 대기합니다. 서버가 새 HTTP Request를 수신하면 **OnWebAuthnRegistrationOptionsRequest** 이벤트가 호출되며, 클라이언트가 보낸 요청에 액세스하고 Accept 매개변수를 False로 설정하여 요청을 취소할 수 있습니다.

```
void __fastcall TForm1::OnWebAuthnRegistrationOptionsRequest(TObject *Sender, TsgcWebAuthn_RegistrationOptions_Re
{
    if (aRequest->Username == "anonymous")
        Accept = false;
}
```

WebAuthn Registration | Response

서버는 클라이언트의 등록 옵션 요청(예: POST /sgcWebAuthn/Registration/Options)에 다음과 같은 JSON 페이로드로 응답합니다(바이너리 필드를 base64url로 인코딩한 후):

```
{
  "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"
    }
  }
}
```

아래에서 필드에 대한 설명을 확인하십시오:

- **rp**: Relying Party 정보:
 - **name**: 친숙한 표시 이름(예: "Example Inc.).
 - **id**: Relying Party ID(사이트 origin과 일치하거나 등록 가능한 접미사여야 함).
- **user**: 사용자 계정을 식별하는 정보:
 - **id**: 고유하고 안정적인 바이트 배열(예: 사용자 UUID).
 - **name**: 고유한 사용자 식별자(사용자 이름 또는 이메일).
 - **displayName**: 사용자에게 표시되는 표시 이름.
- **challenge**: 암호학적으로 무작위인 nonce(16–64바이트)로, base64url로 인코딩됩니다.
- **pubKeyCredParams**: RP가 수락할 의향이 있는 알고리즘:
 - 일반: -7 (ES256), -257 (RS256)
- **timeout**: 생성 작업에 대한 선택적 시간 초과(밀리초)입니다.
- **attestation**: "none", "indirect", "direct" 중 하나입니다.
- **authenticatorSelection**:
 - **authenticatorAttachment**: "platform"(내장) 또는 "cross-platform"(예: YubiKey).
 - **userVerification**: "required", "preferred" 또는 "discouraged".
 - **residentKey**: "required", "preferred" 또는 "discouraged".

응답이 클라이언트로 전송되기 전에 **OnWebAuthnRegistrationOptionsResponse** 이벤트가 호출되어 응답을 사용자 정의할 수 있습니다.

```
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");  
}  
}
```

WebAuthn Registration | Result

목표는 클라이언트가 반환한 데이터의 진위성과 무결성을 검증하고, 자격 증명이 예상 사용자에게 바인딩되어 있는지 확인하며, 향후 인증을 위해 공개 키 자격 증명을 안전하게 등록하는 것입니다.

서버는 다음 단계에 따라 클라이언트 응답을 검증해야 합니다:

- 클라이언트 응답 파싱: 클라이언트가 서버에 다음과 같은 응답을 전송합니다.

```
{
  "id": "base64url-encoded credential ID",
  "rawId": "base64url-encoded ID bytes",
  "response": {
    "clientDataJSON": "base64url",
    "attestationObject": "base64url"
  },
  "type": "public-key"
}
```

- 서버는 clientDataJSON 및 attestationObject를 검증합니다
- Attestation Statement를 검증합니다(정의된 경우)
- 모든 것이 유효하면 자격 증명을 저장하고 향후 로그인을 위해 공개 키를 신뢰할 수 있습니다.

등록 성공

클라이언트가 전송한 응답이 유효한 경우 **OnWebAuthnRegistrationSuccessful** 이벤트가 호출되며, 향후 로그인 검증을 위해 Credential Record를 데이터베이스에 안전하게 저장할 수 있습니다.

```
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

클라이언트 응답을 검증하는 동안 오류가 발생하면 **OnWebAuthnRegistrationError** 이벤트가 호출되며, aError 매개변수에서 오류 이유에 액세스할 수 있습니다.

```
void __fastcall OnWebAuthnRegistrationError(TObject *Sender,
  TsgcWebAuthn_RegistrationVerify_Request* aRequest,
  TsgcWebAuthn_Registration* aRegistration,
  const String aError)
{
  Log("#webauthn_registration_error: " + aError);
}
```

WebAuthn | Authentication

WebAuthn Authentication을 사용하면 사용자가 이전에 등록한 공개 키 자격 증명을 사용하여 로그인할 수 있습니다. 등록에서 저장된 사용자의 공개 키를 사용하여 서명된 챌린지를 검증하는 것이 포함됩니다.

인증 옵션

인증을 위해서는 클라이언트가 TsgcWSAPIServer_WebAuthn 서버에 구성된 **Authentication Options Endpoint**에 새 HTTP Request를 사용하여 인증 흐름을 시작해야 합니다. 기본적으로 엔드포인트는 /sgcWebAuthn/Authentication/Options이므로, 서버가 https://www.test.com 도메인에서 수신 대기하는 경우 클라이언트는 https://www.test.com/sgcWebAuthn/Authentication/Options URL로 새 요청을 해야 합니다.

클라이언트가 POST를 통해 assertion(인증 응답)을 서버로 보냅니다

```
{"username": "test", "user_verification": "preferred"}
```

서버가 PublicKeyCredentialRequestOptions를 생성합니다

```
{
  "challenge": "9d0d61edf30b45f8b88aef7087f9117716e2b7d8b0ee4460b06142f39dd0ec9f",
  "timeout": 60000,
  "rpId": "localhost",
  "allowCredentials": [
    {
      "id": "yeA4BVRlrAfLG-KzqsL_r1I4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
      "type": "public-key",
      "transports": [
        "nfc",
        "usb"
      ]
    }
  ],
  "userVerification": "preferred",
  "hints": [],
  "attestation": "none",
  "attestationFormats": [],
  "extensions": {}
}
```

Authentication Verify

인증을 위해서는 클라이언트가 TsgcWSAPIServer_WebAuthn 서버에 구성된 **Authentication Verify Endpoint**에 새 HTTP Request를 사용하여 인증 흐름을 시작해야 합니다. 기본적으로 엔드포인트는 /sgcWebAuthn/Authentication/Verify이므로, 서버가 https://www.test.com 도메인에서 수신 대기하는 경우 클라이언트는 https://www.test.com/sgcWebAuthn/Authentication/Verify URL로 새 요청을 해야 합니다.

브라우저는 사용자에게 인증자(예: 지문, YubiKey)를 사용하라는 메시지를 표시합니다. 인증자는 credential ID에 연결된 개인 키로 챌린지에 서명합니다. 반환된 자격 증명에는 다음이 포함됩니다:

- **rawId**: 자격 증명 ID(base64url)입니다.
- **response.authenticatorData**: 인증자의 메타데이터(바이너리).
- **response.clientDataJSON**: 원래 챌린지, 오리진 및 유형을 포함합니다.
- **response.signature**: authenticatorData + hash(clientDataJSON)에 대한 서명.

아래에서 클라이언트 요청의 json 예제를 찾으십시오:

```
{
  "id": "yeA4BVRlrAfLG-KzqsL_r1I4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
```

```

"rawId": "yeA4BVRlrAfLG-KzqsL_rlI4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
"response": {
  "authenticatorData": "SZYN5YgOjGh0NBcPZHZgW4_krrmihjLHmVzzuoMdl2MFAAAABw",
  "clientDataJSON": "eyJ0eXB1Ijoid2ViYXV0aG4uZ.....",
  "signature": "MEQCIARqvvys8....",
  "userHandle": "36b9d6a84204487382fee62e7e67a80d"
},
"type": "public-key",
"clientExtensionResults": {},
"authenticatorAttachment": "cross-platform"
}

```

서버는 클라이언트로부터 요청을 읽고, 자격 증명이 저장되어 있는지 검증하고, 서명을 확인합니다. 서명이 유효하면 **OnWebAuthnAuthenticationSuccessful** 이벤트가 호출됩니다.

Authentication Flow

다음 링크를 사용하여 인증 플로우 프로세스에 대한 자세한 정보를 확인하십시오.

- [WebAuthn Authentication Request](#)
- [WebAuthn Authentication Response](#)
- [WebAuthn 인증 결과](#)

WebAuthn Authentication | Request

사용자가 로그인을 시도하면 브라우저는 서버에 인증 옵션("assertion options"라고도 함)을 요청하는 요청을 보냅니다.

```
{
  "username": "alice@example.com"
}
```

이 요청을 통해 서버는 다음을 수행할 수 있습니다:

- 챌린지(안전한 임의 값)를 생성합니다.
- 이 사용자에게 어떤 자격 증명(공개 키 ID)이 유효한지 조회합니다.
- 브라우저가 navigator.credentials.get()에서 사용할 암호화 매개변수로 응답합니다.

WebAuthn Server Component가 이 요청을 수신하면(일반적으로 브라우저에서 서버 엔드포인트로의 HTTPS POST 요청을 통해 수행됨, 예: /sgcWebAuthn/Authentication/Options), **OnWebAuthnAuthenticationOptionsRequest** 이벤트가 호출되므로, 사용자 이름과 연결된 자격 증명(있는 경우)을 추가할 수 있습니다.

```
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

브라우저(클라이언트)가 사용자의 사용자 이름(또는 유사한 식별자)과 함께 서버에 요청을 보내면, 서버는 클라이언트가 브라우저의 `navigator.credentials.get()` API를 통해 인증 프로세스를 시작하는 데 사용할 인증 옵션으로 응답합니다.

이 응답은 사용자의 인증자(예: 보안 키, 생체 인식 장치)를 사용하여 WebAuthn 인증 assertion을 생성하는 데 브라우저가 필요로 하는 매개변수와 제약 조건을 제공합니다.

JSON 응답 예제:

```
{
  "challenge": "Z3lVbWV5YXBpbmdvZG90IQ",
  "timeout": 60000,
  "rpId": "example.com",
  "allowCredentials": [
    {
      "type": "public-key",
      "id": "dXNlckNyZWJZA",
      "transports": ["usb", "nfc", "ble"]
    }
  ],
  "userVerification": "preferred"
}
```

아래에서 필드에 대한 설명을 확인하십시오:

- **challenge:** 재생 공격을 방지하기 위한 암호화 챌린지(무작위로 생성됨)입니다. 요청마다 고유해야 합니다.
- **timeout:** 선택적. 작업을 취소하기 전에 브라우저가 대기해야 하는 시간(ms)입니다.
- **rpId:** Relying Party Identifier — 일반적으로 도메인 이름입니다.
- **allowCredentials:** 사용자가 이전에 등록한 허용 가능한 자격 증명(자격 증명 ID)의 목록입니다.
- **userVerification:** 사용자를 검증하는 방법을 브라우저에 알립니다: "required", "preferred", "discouraged".
- **extensions (선택 사항):** 인증자에게 요청된 추가 기능 또는 데이터입니다.

응답이 클라이언트로 전송되기 전에 **OnWebAuthnAuthenticationOptionsResponse** 이벤트가 호출되어, 응답을 사용자 정의할 수 있게 합니다.

WebAuthn Authentication | Result

목표는 브라우저가 제공한 서명된 assertion을 유효성 검사하는 것으로, 이는 사용자가 원래 등록된 개인 키를 소유하고 있음을 증명합니다. 이것이 사용자를 안전하게 로그인시키는 방법입니다. 사용자가 인증자(예: 지문, 보안 키)와 상호작용한 후, 브라우저는 인증 결과와 함께 서버로 POST 요청을 다시 보냅니다. 아래에서 json 예제를 확인하십시오:

```
{
  "id": "credential-id",
  "rawId": "base64url-encoded-credential-id",
  "type": "public-key",
  "response": {
    "clientDataJSON": "base64url",
    "authenticatorData": "base64url",
    "signature": "base64url",
    "userHandle": "optional"
  }
}
```

서버가 구성된 엔드포인트(예: /sgcWebAuthn/Authentication/Verify)에서 이 요청을 받으면 다음 단계를 검증해야 합니다.

- DB에서 자격 증명 검색: credential-id를 사용하여 사용자의 등록된 공개 키와 메타데이터(예: 서명 카운터)를 조회합니다.
- clientDataJSON 디코딩 & 파싱
- Validate Authenticator Data
- 서명 검증

모든 검증이 올바르게, 인증이 성공하고 **OnWebAuthnAuthenticationSuccessful** 이벤트가 호출됩니다.

검사 중 하나라도 실패하면 오류 이유와 함께 **OnWebAuthnAuthenticationError** 이벤트가 호출됩니다.

WebAuthn | MDS

Metadata Service (MDS)는 FIDO 인증 프로그램을 통해 인증된 authenticator에 대한 **Metadata Statement**를 집계하고 게시하는 **FIDO Alliance**에서 제공하는 중앙 집중식 서비스입니다. 이러한 statement에는 authenticator에 대한 자세한 보안, 규정 준수 및 운영 정보가 포함됩니다.

- 서비스 엔드포인트는 종종 **MDS3**(Metadata Service 프로토콜의 세 번째 버전)이라고 합니다.
- Relying Party(예: WebAuthn을 구현하는 웹사이트 또는 애플리케이션)는 authenticator에 대한 정보에 입각한 신뢰 결정을 내리기 위해 MDS에서 **메타데이터 명세서를 검색**합니다.

MDS는 WebAuthn을 사용하는 relying party에 중요한 신뢰 및 보안 검증 계층을 추가합니다:

- **Authenticator Validation:** 인증기의 FIDO 표준 준수 여부 검증을 활성화하고 WebAuthn attestation에 제시된 AAGUID 검증에 도움을 줍니다.
- **손상 및 취소 탐지:** 손상되거나 취소된 인증자에 대한 최신 정보를 제공하여, 신뢰 당사자가 안전하지 않은 디바이스를 차단할 수 있도록 합니다.
- **Security Assurance:** 특정 FIDO 인증 수준 또는 사용자 검증 강도를 충족하는 인증기만 허용하는 등 보안 정책을 적용하는 데 도움이 됩니다.
- **Interoperability:** 서로 다른 인증기를 사용하는 다양한 브라우저, 플랫폼, 기기 전반에 걸쳐 일관된 동작과 보안 기대치를 보장합니다.

구성

WebAuthnOptions.MDS 속성을 사용하여 MDS 사용을 구성할 수 있습니다. 아래에서 주요 속성을 찾을 수 있습니다:

- **Enabled:** true(기본값)인 경우 webauthn 요청이 구성된 MDS 파일에 대해 검증됩니다.
- **MDS_FileName:** MDS 파일이 저장된 경로입니다. 다음 URL에서 다운로드할 수 있습니다: <https://mds3.fidoalliance.org/>
- **RootCert_FileName:** Root Certificate가 저장되는 경로. 인증서 체인을 검증하려면 정의되어야 합니다. <https://valid.r3.roots.globalsign.com/> 에서 다운로드할 수 있습니다
- **Leaf_CertificateCRL:** true이면(기본값은 false) blob의 leaf 인증서가 CRL(Certificate Revocation List)에 대해 검증됩니다.
- **CRL_FileName:** CRL 파일이 저장되는 경로. 정의되지 않고 Leaf_CertificateCRL이 활성화된 경우, CRL을 자동으로 다운로드하려고 시도합니다.

WebAuthn Authorization

인증 성공 후 새 WebSocket 또는 HTTP 연결을 여는 데 사용할 수 있는 bearer token을 서버가 보내기를 원하면, token = true 매개변수를 전달하십시오. 예:

```
{
  "username": "alice@example.com", "token": true
}
```

인증이 성공하면 서버는 다음과 같은 응답을 보냅니다:

```
{
  "verified": "ok",
  "authentication": {
    "token": "C760C1C39E3D4E829693A13F18F5CFDE537B516336FC48F7BAB0276176F9E6DE"
  }
}
```

OnWebAuthnUnauthorized 이벤트는 요청이 권한 부여되지 않아 연결이 끊길 때 호출됩니다. 여기서 어떤 엔드포인트가 WebAuthn 인증을 요구하고 어떤 엔드포인트가 요구하지 않는지 구성할 수 있습니다.

WebAuthn Authorization | HTTP

새 토큰을 얻으면, 이 bearer 토큰과 함께 authorization 헤더를 보내기만 하면 됩니다. TsgcHTTP1Client의 CustomHeaders 속성을 사용하여 Bearer Token을 구성할 수 있습니다. 예:

```
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

새 토큰을 얻으면, 이 bearer 토큰과 함께 authorization 헤더를 보내기만 하면 됩니다. OnHandShake 이벤트를 사용하여 연결 요청에 Bearer Token을 추가할 수 있습니다. 예:

```
void __fastcall OnClientHandshake(TsgcWSConnection *Connection, TStringList *Headers)
{
    Headers->Add("Authorization: Bearer C760C1C39E3D4E829693A13F18F5CFDE537B516336FC48F7BAB0276176F9E6DE");
}
```

Webauthn | Javascript Client

WebAuthn(Web Authentication API)은 공개 키 암호화를 사용하여 안전한 비밀번호 없는 인증을 가능하게 하는 W3C 표준입니다. 사용자는 비밀번호 대신 **하드웨어 기반 인증자**(지문 판독기, Face ID, YubiKey 등) 또는 플랫폼 인증자(Touch ID와 같은 내장형)를 사용하여 등록하고 인증합니다.

Javascript 클라이언트를 사용하여 등록 및 인증을 처리하는 방법은 아래에서 확인하십시오.

WebAuthn Registration

WebAuthn Registration의 작동 방식

- **사용자가 등록을 시작합니다:**
 - 사용자가 사용자 이름을 제공하고 **Register**를 클릭합니다.
- **브라우저가 서버에 Options를 요청함:**
 - 프론트엔드가 등록 옵션을 얻기 위해 POST 요청을 합니다.
- **브라우저가 자격 증명을 생성합니다:**
 - `SimpleWebAuthnBrowser.startRegistration()`를 통한 `navigator.credentials.create()` API를 사용하여 자격 증명 생성됩니다.
- **서버가 등록을 검증함:**
 - 브라우저가 자격 증명을 서버로 다시 보냅니다.
 - 서버가 등록을 검증하고 해당 사용자의 공개 키를 저장합니다.

TsgcWSAPIServer_WebAuthn 구성 요소에는 WebAuthn 프로토콜을 테스트하기 위한 html 파일이 있습니다. 이 HTML 파일에는 WebAuthn과 상호 작용하기 위한 최소한의 UI와 JavaScript가 포함되어 있습니다.

sgcWebAuthn.html 둘러보기

구조 개요:

- **Username Input:** 사용자의 식별자를 캡처합니다.
- **버튼:**
 - Register – WebAuthn 등록을 시작합니다.
 - Authenticate – 로그인 시작합니다(유사하게 처리됨).
- **Debug Console:** 실시간 디버깅 정보(WebAuthn의 JSON)를 표시합니다.

1. 입력용 HTML UI

```
<input type="text" id="username" name="username" autocomplete="username webauthn" />
<button id="btnRegBegin"><strong>Register</strong></button>
```

2. JavaScript: 버튼 클릭 핸들러

```
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. 서버 응답(HTML의 가짜 엔드포인트)

```
fetch('/sgcWebAuthn/Registration/Options', ...)
fetch('/sgcWebAuthn/Registration/Verify', ...)
```

4. 등록 완료

```
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

WebAuthn 인증 작동 방식

- 사용자가 인증을 시작합니다:
 - 사용자가 사용자 이름을 제공하고 **Register**를 클릭합니다.
- 브라우저가 서버에 **Options**를 요청함:
 - 프론트엔드가 인증 옵션을 얻기 위해 POST 요청을 합니다.
- 브라우저가 자격 증명을 생성합니다:
 - SimpleWebAuthnBrowser.startAuthentication()을 통해 navigator.credentials.get() API를 사용합니다.
- 서버가 인증을 검증함:
 - 브라우저가 자격 증명을 서버로 다시 전송합니다.
 - 서버가 서명된 결과를 검증합니다.

1. HTML UI 설정

```
<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. Authentication Options 가져오기

startAuthentication을 호출하기 전에 사용자 이름을 서버로 보냅니다

```
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'
  }),
});
```

서버는 다음을 포함하는 JSON 객체로 응답합니다:

```
{
  "challenge": "base64url-encoded-random-string",
  "allowCredentials": [
    {
      "id": "base64url-credential-id",
      "type": "public-key"
    }
  ],
  "userVerification": "preferred",
  "rpId": "yourdomain.com"
}
```

이것을 PublicKeyCredentialRequestOptions라고 합니다.

3. Authenticator 응답 수신

asseResp는 다음과 같습니다(단순화됨):

```
{
  "id": "credentialId",
  "rawId": "base64url-encoded-id",
  "response": {
    "authenticatorData": "...",
    "clientDataJSON": "...",
    "signature": "...",
    "userHandle": "..."
  },
  "type": "public-key",
  "clientExtensionResults": {}
}
```

이 응답은 사용자가 다음을 증명합니다:

- 인증자에 저장된 개인 키를 소유합니다.
- 서버의 챌린지에 서명함
- 물리적으로 존재했음(필요한 경우)

4. 서명된 Authentication Response를 서버에 전송

사용자가 인증자와 상호 작용한 후(startAuthentication())을 통해) JavaScript에서 응답 객체를 얻습니다.

```
const verificationResp = await fetch('/webauthn/authenticate/verify', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(asseResp), // 서명된 응답입니다
});
```

5. 서버의 응답 받기

서버는 다음과 같은 결과로 응답합니다:

```
{ "verified": true }
```

또는 문제가 발생한 경우:

```
{ "verified": false, "error": "Invalid signature" }
```

그리고 프론트엔드 코드에서 이를 처리합니다:

```
const result = await verificationResp.json();
```

```
if (result.verified) {  
    document.getElementById('Success').textContent = 'User authenticated!';  
} else {  
    document.getElementById('Error').textContent = 'Authentication failed!';  
}
```

Amazon AWS | SQS

Amazon SQS란 무엇입니까?

Amazon Simple Queue Service(SQS)는 microservices, 분산 시스템, serverless 애플리케이션을 분리하고 확장할 수 있게 하는 완전 관리형 메시지 큐 서비스입니다. SQS는 메시지 지향 미들웨어 관리 및 운영과 관련된 복잡성과 오버헤드를 제거하고, 개발자가 차별화된 작업에 집중할 수 있게 합니다. SQS를 사용하면 메시지를 잃거나 다른 서비스가 사용 가능해야 할 필요 없이 어떤 볼륨에서든 소프트웨어 구성 요소 간에 메시지를 보내고, 저장하고, 받을 수 있습니다.

이점

관리 오버헤드 제거

SQS를 사용하면 선결제 비용이 없고, 메시징 소프트웨어를 획득, 설치 및 구성할 필요가 없으며, 지원 인프라의 시간이 많이 소요되는 구축 및 유지 보수가 필요하지 않습니다.

안정적으로 메시지 전달

SQS를 사용하면 애플리케이션 구성 요소를 분리하여 독립적으로 실행 및 실패하도록 하여 시스템의 전체 내결함성을 높일 수 있습니다.

민감한 데이터를 안전하게 유지

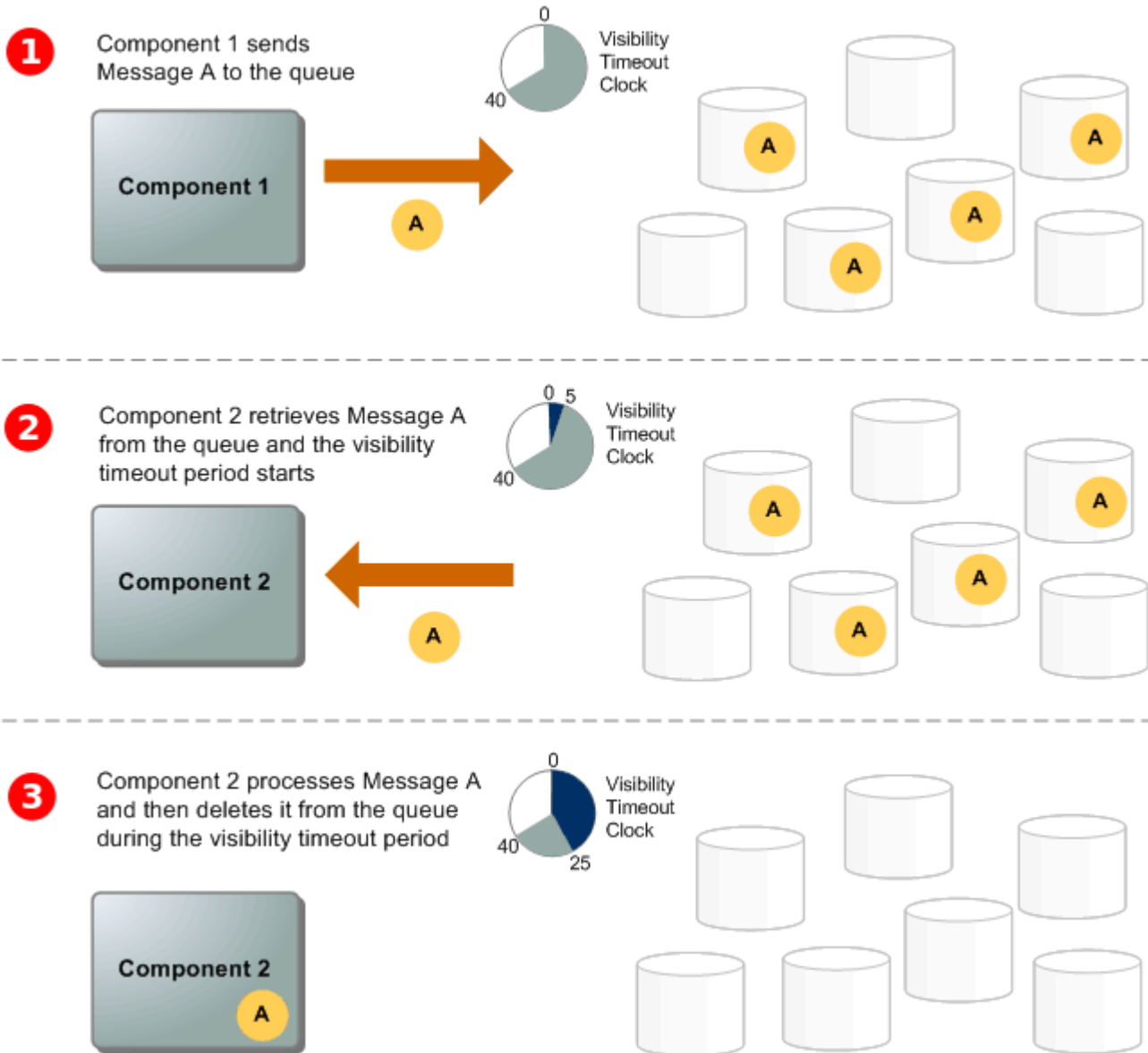
서버측 암호화(SSE)를 사용하여 각 메시지 본문을 암호화함으로써 애플리케이션 간에 민감한 데이터를 교환하기 위해 Amazon SQS를 사용할 수 있습니다.

탄력적이고 비용 효율적으로 확장

SQS는 애플리케이션에 따라 탄력적으로 확장되므로 용량 계획 및 사전 프로비저닝에 대해 걱정할 필요가 없습니다.

WorkFlow

다음 시나리오는 큐에서 Amazon SQS 메시지의 수명 주기를 생성부터 삭제까지 설명합니다.



Amazon SQS 시작하기

시작하기 전에 Setting Up Amazon SQS의 단계를 완료하십시오.

1단계: 큐 생성

1. Amazon SQS 콘솔에 로그인하십시오.
2. Create New Queue를 선택하십시오.
3. Create New Queue 페이지에서, 올바른 region에 있는지 확인한 다음 Queue Name을 입력하십시오.
4. Standard가 기본적으로 선택됩니다. FIFO를 선택하십시오.
5. 기본 매개변수로 큐를 생성하려면 Quick>Create Queue를 선택하십시오.

새 큐가 생성되어 큐 목록에서 선택됩니다.

2단계: 메시지 보내기

큐를 생성한 후 메시지를 보낼 수 있습니다. 다음 예제는 기존 큐에 메시지를 보내는 것을 보여줍니다.

1. 큐 목록에서 생성한 큐를 선택하십시오.
2. Queue Actions에서 Send a Message를 선택하십시오.
3. 메시지가 전송되고 Send a Message to QueueName 대화 상자가 표시되어, 전송된 메시지의 속성을 보여줍니다.

3단계: 메시지 수신 및 삭제

큐에 메시지를 보낸 후 이를 소비(큐에서 검색)할 수 있습니다. 큐에서 메시지를 요청할 때 어떤 메시지를 가져올지 지정할 수 있습니다. 대신 가져올 최대 메시지 수(최대 10개)를 지정합니다.

4단계: 큐 삭제

Amazon SQS 큐를 사용하지 않고(가까운 미래에 사용할 것으로 예상하지 않는 경우) Amazon SQS에서 삭제하는 것이 모범 사례입니다.

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->PurgueQueue("sqs_queue") == true)
{
    DoLog("#PurgueQueue: ok");
}
else
{
    DoLog("#PurgueQueue: 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.
```

Events

OnSQSBeforeRequest

이 이벤트는 sqs 구성 요소가 HTTP 요청을 하기 전에 호출됩니다. URL 매개변수에 액세스할 수 있으며 Handled 매개변수가 True로 설정되면 구성 요소가 HTTP 요청을 하지 않음을 의미합니다.

OnSQSError

구성 요소가 요청을 수행할 때 오류가 있으면 이 이벤트가 Error Code와 Error Description과 함께 호출됩니다.

OnSQSResponse

이 이벤트는 서버로부터의 원시 응답이 포함된 HTTP 요청 후에 호출됩니다.

Google Cloud | Google OAuth2 Keys

sgcWebSockets Google Cloud 구성 요소를 사용하고 OAuth2를 통해 인증하려면, 먼저 Google Cloud에서 OAuth2 Key를 얻어야 합니다.

Google OAuth2 키를 얻는 방법과 PubSub 샘플 애플리케이션에서 이를 구성하는 방법은 아래 단계를 참조하십시오.

먼저 **Google Cloud** 계정에 로그인하고 기존 프로젝트를 사용하거나 새 프로젝트를 생성하십시오.

그 후, **Credentials** 메뉴로 이동하여 **CREATE CREDENTIALS** 버튼을 누르고, **OAuth Client ID** 옵션을 선택하십시오.

애플리케이션 유형을 선택하고 설명적인 이름을 설정하십시오.

성공하면 Client ID와 Client Secret를 받게 됩니다.

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

pvogD9reE0t9ilL6eR1jE60Z

OK

OAuth2 데이터를 누구와도 공유하지 마십시오!

이제 sgcWebSockets PubSub 샘플로 복사하고, Project Id(프로젝트 이름이 아님)를 추가하십시오

Select a project



NEW PROJECT

RECENT

ALL

Name		ID
✓  PubSub 		pubsub-270909
 GmailApiTest 		gmailapitest-265010
 My Project 		melodic-voice-265009

CANCEL

OPEN

이것이 sgcWebSocket PubSub 샘플에서 구성해야 하는 방법입니다.

Google Pub/Sub API Client

Google API

Client Id: 843483347040-ehcskpfsp4180rlbdfoe6mc32e3ncmn0.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

예를 들어 새 topic을 만들어 볼 수 있습니다. 처음에는 OAuth2 연결을 승인해야 하므로, Google에서 제공하는 OAuth2 자격 증명으로 계정 액세스 권한을 요청하기 위해 새 웹 브라우저가 표시됩니다.

 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](#)

연결을 허용하고, 성공하면 이 API로 작업을 시작할 수 있습니다.

Google Pub/Sub API Client

Google API

Client Id>4180rlbdf6e6mc32e3ncmn0.apps.googleusercontent.com

Client SecretpvogD9reEOt9iLL6eR1jE60Z

Publisher

Project-idpubsub-270909Topictopic-1

Create TopicDelete TopicGet TopicList Topics

PublishMessage sent from sgcWebSockets Pub/Sub API.

Subscriber

Project-idSubscriptionsubscription-1Topictopic-1

Create Subs.Delete Subs.Get Subs.List Subs.

PullAcknowledge

#CreateTopic: { "name": "projects/pubsub-270909/topics/topic-1"}
#Publish: { "messageIds": ["1780785665748120"]}
#Publish: { "messageIds": ["1780799518708503"]}

Google Cloud | Google Service Accounts

sgcWebSockets Google Cloud 구성 요소를 사용하고 서비스 계정을 사용하여 인증하려면, 먼저 Private Key Certificate를 얻어야 합니다.

Google Cloud에서.

Google Private Key Certificate를 얻는 단계와 PubSub 샘플 애플리케이션에서 이를 구성하는 방법은 아래를 참조하십시오.

The screenshot shows the 'IAM & Admin' section of the Google Cloud console. The 'Service accounts' tab is selected, displaying a table of service accounts for the project 'PubSub'. The table is currently empty, with a message 'No rows to display'. The left sidebar shows the navigation menu with 'Service Accounts' highlighted.

CREATE SERVICE ACCOUNT를 선택하면 서비스 계정 이름과 설명을 설정해야 하는 새 페이지가 표시됩니다

The screenshot shows the 'Create service account' page in the Google Cloud IAM & Admin console. The page is titled '1 Service account details' and contains the following fields:

- Service account name:** sgcServiceAccount
- Service account ID:** sgcserviceaccount (with a display name of @pubsub-270909.iam.gserviceaccount.com)
- Service account description:** Service Account Test

A 'CREATE' button is visible at the bottom of the form.

그런 다음 최소 하나의 역할을 선택하십시오. 여기서는 클라이언트가 토픽을 게시하고 구독할 수 있도록 PubSub Admin을 선택하지만, 권한이 적은 다른 역할을 선택할 수 있습니다.

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

CONTINUE를 누르면 마지막으로 다른 사용자에게 접근 권한을 부여할 수 있습니다

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

완료되면 DONE을 누르면 새 레코드가 표시됩니다.

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

다음 단계는 새 키를 생성하는 것입니다. actions 열에서 Create Key 옵션을 선택하십시오. 구성을 JSON 형식으로 다운로드 하려면 JSON을 선택하면 새 키가 생성됩니다.

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	⋮

마지막으로, sgcWebSockets PubSub 클라이언트에 Google이 제공한 데이터를 입력하기만 하면 됩니다. 구성 JSON 파일을 로드하려면 **LoadSettingsFromFile**을 사용할 수 있습니다.

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-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBA
QC3xz0L2Wkr9Hvj
HOM/i5Qum78NIIc8nVrF8HlQkVr8awSIV

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

Google Cloud Pub/Sub란 무엇입니까?

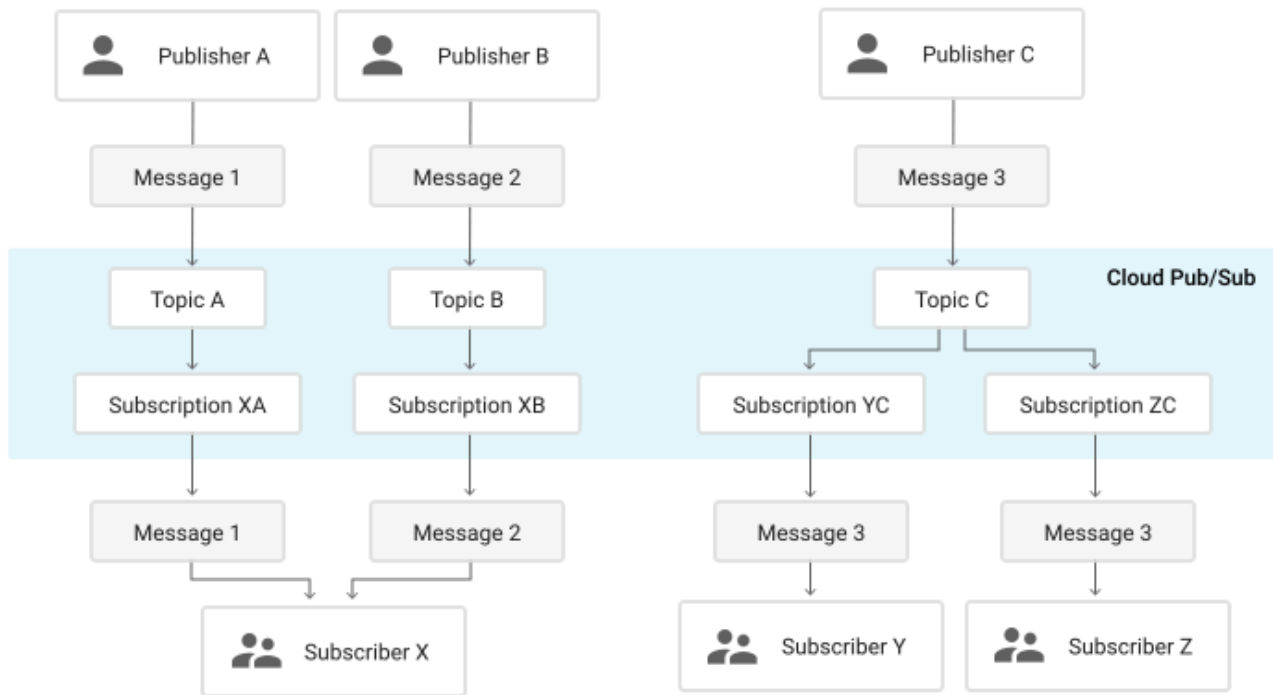
Pub/Sub은 엔터프라이즈 message-oriented middleware의 유연성과 안정성을 클라우드로 가져옵니다. 동시에 Pub/Sub은 확장 가능하고 내구성 있는 이벤트 수집 및 전달 시스템으로, 최신 스트림 분석 파이프라인의 기반 역할을 합니다. 송신자와 수신자를 분리하는 다대다 비동기 메시징을 제공하여 독립적으로 작성된 애플리케이션 간에 안전하고 고가용성 통신을 가능하게 합니다. Pub/Sub은 개발자가 Google Cloud Platform 및 외부에서 호스팅되는 시스템을 빠르게 통합하는 데 도움이 되는 저지연, 내구성 있는 메시징을 제공합니다.

Features

At-least-once 전달 동기, 영역 간 메시지 복제 및 메시지별 수신 추적은 모든 규모에서 at-least-once 전달을 보장합니다.	Open 7개 언어의 Open API 및 클라이언트 라이브러리가 크로스 클라우드 및 하이브리드 배포를 지원합니다.	정확히 한 번 처리 Cloud Dataflow는 Cloud Pub/Sub 스트림의 안정적이고 표현력 있는 정확히 한 번 처리를 지원합니다.
기본적으로 글로벌 전 세계 어디에서나 게시하고 어디에서나 소비하며, 일관된 지연 시간을 제공합니다. 복제가 필요하지 않습니다.	프로비저닝 없음, 모든 것이 자동 Cloud Pub/Sub에는 shard나 파티션이 없습니다. 쿼터를 설정하고, 게시하고, 소비하기만 하면 됩니다.	규정 준수 및 보안 Cloud Pub/Sub은 세분화된 액세스 제어와 종단간 암호화를 제공하는 HIPAA 규정 준수 서비스입니다.
통합 Cloud Storage 및 Gmail 업데이트 이벤트, 서버리스 이벤트 기반 컴퓨팅을 위한 Cloud Functions와 같은 여러 서비스와의 통합을 활용하십시오.	탐색 및 재생 백로그를 임의의 시점이나 스냅샷으로 되감아 메시지를 재처리할 수 있습니다. 오래된 데이터를 버리려면 빨리 감기를 하십시오.	

게시자-구독자 관계

publisher 애플리케이션은 메시지를 생성하여 topic으로 보냅니다. Subscriber 애플리케이션은 topic에 대한 subscription을 생성하여 해당 topic으로부터 메시지를 수신합니다. 통신은 일대다(fan-out), 다대일(fan-in), 다대다가 될 수 있습니다.



일반적인 사용 사례

- **네트워크 클러스터의 워크로드 분산.** 예를 들어, 대규모 작업 큐를 Google Compute Engine 인스턴스와 같은 여러 워커에 효율적으로 분배할 수 있습니다.
- **비동기 워크플로 구현.** 예를 들어, 주문 처리 애플리케이션은 topic에 주문을 배치할 수 있으며, 거기서 하나 이상의 worker가 처리할 수 있습니다.
- **이벤트 알림 배포.** 예를 들어, 사용자 가입을 수락하는 서비스는 새 사용자가 등록될 때마다 알림을 보낼 수 있으며, 다운스트림 서비스는 이벤트 알림을 받기 위해 구독할 수 있습니다.
- **분산 캐시 새로 고침.** 예를 들어 애플리케이션은 변경된 객체의 ID를 업데이트하기 위해 무효화 이벤트를 게시할 수 있습니다.
- **여러 시스템에 로깅.** 예를 들어, Google Compute Engine 인스턴스는 모니터링 시스템, 나중에 쿼리하기 위한 데이터베이스 등에 로그를 기록할 수 있습니다.
- **다양한 프로세스 또는 장치에서의 데이터 스트리밍.** 예를 들어, 주거용 센서는 클라우드에 호스팅된 백엔드 서버로 데이터를 스트리밍할 수 있습니다.
- **신뢰성 향상.** 예를 들어, 단일 영역 Compute Engine 서비스는 공통 topic을 구독하여 추가 영역에서 작동할 수 있으며, 영역 또는 지역의 장애로부터 복구할 수 있습니다.

권한 부여

Google Pub/Sub 구성 요소 클라이언트는 다음 방법으로 Google 서버에 로그인할 수 있습니다:

- **gcaOAuth2:** OAuth2 프로토콜
- **gcaJWT:** JWT 토큰.

OAuth2

로그인은 사용자가 자신의 계정으로 로그인하고 PubSub 요청을 승인하는 웹 브라우저를 사용하여 수행됩니다.

- **GoogleCloudOptions.OAuth2.ClientId:** OAuth2 프로토콜을 통해 인증하기 위해 Google에서 제공하는 ClientID입니다.
- **GoogleCloudOptions.OAuth2.ClientSecret:** OAuth2 프로토콜을 통해 인증하기 위해 Google이 제공한 Client Secret 문자열입니다.

- **GoogleCloudOptions.OAuth2.Scope:** OAuth2의 스코프입니다. 일반적으로 기본값보다 더 많은 접근이 필요하지 않은 한 기본값을 수정할 필요가 없습니다.
- **GoogleCloudOptions.OAuth2.LocalIP:** OAuth2 프로토콜은 인증 서버로부터의 응답을 수신 대기하는 로컬 서버가 필요합니다. 이것은 IP 또는 DNS 이름입니다. 기본적으로 127.0.0.1입니다.
- **GoogleCloudOptions.OAuth2.LocalPort:** 로컬 서버 수신 대기 포트입니다.
- **GoogleCloudOptions.OAuth2.RedirectURL:** LocalPort + LocalIP와 다른 리디렉션 URL을 설정해야 하는 경우, 이 속성에 설정할 수 있습니다(예: http://127.0.0.1:8080/oauth2).

Service Accounts

로그인은 Google이 제공한 private key로 요청에 서명하여 수행됩니다. 이 메서드는 사용자 상호 작용이 없는 자동화된 서비스 또는 애플리케이션에 권장됩니다.

- **GoogleCloudOptions.JWT.ClientEmail:** 새 서비스 계정을 생성할 때 제공된 클라이언트 이메일 이름. JSON 구성 파일의 "client_email" 노드.
- **GoogleCloudOptions.JWT.PrivateKeyId:** Google에서 제공하는 Private Key ID입니다. JSON 구성 파일의 "private_key_id" 노드입니다.
- **GoogleCloudOptions.JWT.PrivateKey:** Google이 제공한 비공개 키 인증서입니다. JSON 구성 파일의 "private_key" 노드입니다.

TLS Options

TLSOptions는 Google Cloud Pub/Sub와 통신하는 데 사용되는 보안 채널의 구성을 중앙 집중화합니다.

- **TLSOptions.IOHandler:** 배포 환경에 맞는 TLS 구현을 선택하십시오(OpenSSL, SChannel...).
- **TLSOptions.Version:** 인프라가 요구할 때 협상을 주어진 TLS 버전으로 제한합니다.
- **TLSOptions.VerifyCertificate:** 인증서 검증을 제어하여 더 엄격한 보안 정책을 활성화합니다.
- **TLSOptions.OpenSSL_Options.LibPath:** 애플리케이션과 함께 제공되는 OpenSSL 바이너리가 포함된 디렉토리를 설정하십시오.
- **TLSOptions.SChannel_Options.UseLegacyCredentials:** SChannel이 요구할 때 Windows 레거시 자격 증명 처리를 활성화합니다.

예제:

```
TsgcHTTPGoogleCloud_PubSub_Client *oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client(NULL);
oPubSub->TLSOptions->IOHandler = iohOpenSSL;
oPubSub->TLSOptions->Version = tls1_3;
oPubSub->TLSOptions->VerifyCertificate = true;
oPubSub->TLSOptions->OpenSSL_Options->LibPath = oslpDefaultFolder;
```

새 서비스 계정이 생성되면 모든 구성이 포함된 JSON 파일을 다운로드할 수 있습니다. 이 파일은 PubSub 구성 요소로 처리할 수 있으며, **LoadSettingsFromFile** 메서드를 호출하고 JSON 파일명을 인수로 전달하기만 하면 됩니다.

Most common uses

- 구성
 - [Google OAuth2 Keys](#)
 - [Service Accounts](#)

Google Pub/Sub Client

OAuth2

Google Pub/Sub API로 작업하려면, sgcWebSockets Pub/Sub 구성 요소가 기본 인증으로 OAuth2를 사용하므로, 먼저 google 계정에서 **ClientId** 및 **ClientSecret**를 설정해야 합니다.

```
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

서비스 계정에는 JWT를 빌드하여 권한 부여 토큰으로 전달하는 것이 필요합니다.

```
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...";
```

이것은 모든 Rest API 호출에 사용될 Authorization Token Key를 Google로부터 얻기 위해 필요합니다.

모든 메서드는 응답을 반환하며, 이는 성공하거나 오류를 반환할 수 있습니다.

Projects.Snapshots

Method	매개변수	설명	Example
CreateSnapshot	project, snapshot, subscription	요청된 구독에서 스냅샷을 생성합니다. 스냅샷은 subscriptions.seek 작업에 사용되며, 이를 통해 메시지 승인을 대량으로 관리할 수 있습니다. 즉, 기존 구독의 메시지 승인 상태를 스냅샷이 캡처한 상태로 설정할 수 있습니다.	CreateSnapshot('pubsub-270909', 'snapshot-1', 'subscription-1')
DeleteSnapshot	project, snapshot	기존 스냅샷을 제거합니다.	DeleteSnapshot('pubsub-270909', 'snapshot-1')
ListSnapshots	project	기존 스냅샷을 나열합니다.	ListSnapshots('pubsub-270909')

Projects.Subscriptions

Method	매개변수	설명	Example
AcknowledgeSubscription			
CreateSubscription	project, subscription, topic	주어진 토픽에 대한 구독을 생성합니다. 구독이 이미 존재하면 ALREADY_EXISTS를 반환합니다. 해당 토픽이 존재하지 않으면	CreateSubscription('pubsub-270909', 'subscription-1', 'topic-1')

		NOT_FOUND를 반환합니다.	
DeleteSubscription	project, subscription	기존 구독을 삭제합니다. 구독에 보존된 모든 메시지는 즉시 삭제됩니다.	DeleteSubscription('pubsub-27090', 'subscription-1')
GetSubscription	project, subscription	구독의 구성 세부 정보를 가져옵니다.	GetSubscription('pubsub-27090', 'subscription-1')
ListSubscriptions	project	일치하는 구독을 나열합니다.	ListSubscriptions('pubsub-27090', 'subscription-1')
ModifyAckDeadlineSubscription	project, subscription, AckIds	특정 메시지의 ack 마감 시간을 수정합니다. 이 메서드는 구독자가 메시지를 처리하는 데 더 많은 시간이 필요함을 나타내거나, 처리가 중단된 경우 메시지를 재전달할 수 있도록 만드는 데 유용합니다. 이것은 후속 메시지에 사용되는 구독 수준의 ackDeadlineSeconds를 수정하지 않습니다.	
ModifyPushConfigSubscription	project, subscription	지정된 구독의 PushConfig를 수정합니다. 이는 푸시 구독을 풀(pull) 구독으로(빈 PushConfig로 표시) 또는 그 반대로 변경하거나, 푸시 구독의 엔드 포인트 URL 및 기타 속성을 변경하는 데 사용할 수 있습니다. PushConfig의 변경 사항에 관계없이 호출 전 반에 걸쳐 메시지가 전달을 위해 계속 누적됩니다.	
Pull	project, subscription	서버에서 메시지를 가져옵니다. 지정된 구독에 대해 대기 중인 동시 풀(pull) 요청이 너무 많으면 서버가 UNAVAILABLE을 반환할 수 있습니다.	pull('pubsub-27090', 'subscription-1')
Seek	project, subscription, timeUTC, snapshot	요청에 제공된 시점 또는 지정된 스냅샷으로 기존 구독을 탐색합니다. 스냅샷은 subscriptions.seek 작업에 사용되며, 이를 통해 메시지 확인을 일괄적으로 관리할 수 있습니다. 즉, 기존 구독의	

메시지 확인 상태를 스냅샷에 의해 캡처된 상태로 설정할 수 있습니다. 구독과 스냅샷은 모두 동일한 토픽에 있어야 합니다.

Projects.Topics

Method	매개변수	설명	Example
CreateTopic	project, topic	지정된 이름으로 지정된 토픽을 생성합니다	CreateTopic('pubsub-270909', 'topic-1')
DeleteTopic	project, topic	지정된 이름의 토픽을 삭제합니다. 토픽이 존재하지 않으면 NOT_FOUND를 반환합니다. 토픽이 삭제된 후 동일한 이름으로 새 토픽을 생성할 수 있습니다. 이는 이전 구성이나 구독이 없는 완전히 새로운 토픽입니다.	DeleteTopic('pubsub-270909', 'topic-1')
GetTopic	project, topic	topic의 구성을 가져옵니다.	GetTopic('pubsub-270909', 'topic-1')
ListTopics	project	일치하는 토픽을 나열합니다.	ListTopics('pubsub-270909')
Publish	project, topic, message	토픽에 하나 이상의 메시지를 추가합니다. 토픽이 존재하지 않으면 NOT_FOUND를 반환합니다.	Publish('pubsub-270909', 'topic-1', 'My First PubSub Message.')

Projects.Topics.Subscriptions

Method	매개변수	설명	Example
ListTopicSubscriptions	project, topic	이 토픽의 구독	ListTopicSubscriptions('pubsub-270909', 'topic-1')

독
이
름
을
나
열
합
니
다.

가장 일반적인 메서드

Google Cloud Pub/Sub API와 함께 사용되는 가장 일반적인 메서드는 아래를 참조하십시오.

새 Topic을 생성하는 방법

id: pubsub-270909인 프로젝트에 topic-1이라는 토픽 이름으로 새 토픽을 생성합니다.

```
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");
```

서버로부터의 응답

```
{
  "name": "projects/pubsub-270909/topics/topic-1"
}
```

메시지 게시

새로 생성된 토픽에 새 메시지를 게시합니다.

```
oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client();
oPubSub->GoogleCloudOptions->OAuth2->ClientId = "... your google client id...";
oPubSub->GoogleCloudOptions->OAuth2->ClientSecret = "... your google client secret...";
oPubSub->Publish("pubsub-270909", "topic-1", "My First Message from sgcWebSockets.");
```

서버로부터의 응답

```
{
  "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();
}

```

새 구독을 생성하는 방법

id가 pubsub-270909인 프로젝트에 대해 구독 이름 subscription-1과 topic-1로 새 구독을 생성합니다.

```

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");

```

서버로부터의 응답

```

{
  "name": "projects/pubsub-270909/subscriptions/subscription-1",
  "topic": "projects/pubsub-270909/topics/topic-1",
  "pushConfig": {},
  "ackDeadlineSeconds": 10,
  "messageRetentionDuration": "604800s",
  "expirationPolicy": {
    "ttl": "2678400s"
  }
}

```

구독에서 메시지를 읽는 방법

생성된 이전 구독에서 메시지를 읽습니다.

```

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");

```

서버로부터의 응답

```

{
  "receivedMessages": [
    {
      "ackId": "PjA-RVNEUAYWLF1GSFE3GQhoUQ5PXiM_NSAorREFC08CKF15MEorQVh0Dj4N",
      "message": {
        "data": "TXkgRmlyc3QgTWVzc2FnZSBmcm9tIHNNY1dlY1NvY2tldHMu",
        "messageId": "1050732082561505",
        "publishTime": "2020-03-14T15:25:31.505Z"
      }
    }
  ]
}

```

메시지는 Base64로 인코딩되어 수신되므로 내용을 읽으려면 먼저 디코딩해야 합니다.

```

sgcBase_Helpers->DecodeBase64("TXkgRmlyc3QgTWVzc2FnZSBmcm9tIHNNY1dlY1NvY2tldHMu=");

```

Google Cloud | Calendar

Google Calendar API를 사용하면 앱을 Google Calendar와 통합하여 사용자와 소통하는 새로운 방법을 만들 수 있습니다. Calendar API를 사용하면 캘린더 이벤트를 표시, 생성 및 수정할 수 있을 뿐만 아니라 캘린더 또는 액세스 제어와 같은 많은 다른 캘린더 관련 객체로 작업할 수 있습니다.

API Resources

Google Calendar는 다음 리소스를 사용합니다:

- **Event:** 제목, 시작 및 종료 시간, 참석자 등의 정보를 포함하는 캘린더의 이벤트입니다. 이벤트는 단일 이벤트 또는 반복 이벤트일 수 있습니다. 이벤트는 Event 리소스로 표현됩니다. 지정된 캘린더의 Events 컬렉션은 해당 캘린더의 모든 이벤트 리소스를 포함합니다.
- **Calendar:** calendar는 이벤트의 모음입니다. 각 calendar에는 calendar 설명 또는 기본 calendar 시간대와 같은 연결된 메타데이터가 있습니다. 단일 calendar의 메타데이터는 Calendar 리소스로 표현됩니다. Calendars 컬렉션에는 기존의 모든 calendar에 대한 Calendar 리소스가 포함됩니다.
- **CalendarList:** Calendar UI에서 사용자의 캘린더 목록에 있는 모든 캘린더의 목록입니다. 캘린더 목록에 나타나는 단일 캘린더의 메타데이터는 CalendarListEntry 리소스로 표현됩니다. 이 메타데이터에는 색상이나 새 이벤트에 대한 알림과 같은 캘린더의 사용자별 속성이 포함됩니다. CalendarList 컬렉션에는 주어진 사용자에 대한 모든 CalendarListEntry 리소스가 포함됩니다. Calendars 및 CalendarList 컬렉션 간의 차이에 대한 자세한 설명은 Calendar and Calendar List를 참조하십시오.
- **Setting:** 사용자의 시간대와 같은 Calendar UI의 사용자 환경 설정입니다. 단일 사용자 환경 설정은 Setting Resource로 표현됩니다. Settings 컬렉션은 주어진 사용자의 모든 Setting 리소스를 포함합니다.
- **ACL:** 사용자(또는 사용자 그룹)에게 캘린더에 대한 지정된 수준의 액세스 권한을 부여하는 액세스 제어 규칙. 단일 액세스 제어 규칙은 ACL 리소스로 표현됩니다. 주어진 캘린더의 ACL 컬렉션에는 해당 캘린더에 대한 액세스를 부여하는 모든 ACL 리소스가 포함됩니다.
- **Color:** Calendar UI에 표시되는 색상. Colors 리소스는 Calendar UI에서 사용 가능한 모든 색상 집합을 두 그룹으로 나타냅니다: 이벤트에 사용 가능한 색상과 캘린더에 사용 가능한 색상.
- **Free/busy:** 캘린더에 일정이 예약된 시간은 "busy"로 간주되고, 캘린더에 일정이 없는 시간은 "free"로 간주됩니다. Freebusy 리소스를 사용하면 주어진 캘린더 또는 캘린더 집합에 대한 busy 시간 집합을 쿼리할 수 있습니다.

주요 기능

- 완전한 기능을 갖춘 Google Calendar Client API V3.
- API에서 지원하는 모든 메시드는 클라이언트 API를 사용하여 호출할 수 있습니다.
- HTTP/2 프로토콜을 사용하는 클라이언트 요청(*Enterprise Edition만 해당).
- PageNextToken을 사용한 부분 응답의 자동 처리.
- Calendar 및 Event 데이터 속성에 쉽게 액세스할 수 있습니다.
- 인증 방법:
 - OAuth2: 사용자 상호 작용이 필요합니다.
 - Service Accounts (Domain-Wide Delegation 필요): windows 서비스, 데몬...

구성

Google Calendar 구성 요소 클라이언트는 다음 속성을 가지고 있습니다:

OAuth2

- **GoogleCloudOptions.OAuth2.ClientId:** OAuth2 프로토콜을 통해 인증하기 위해 Google에서 제공하는 ClientID입니다.
- **GoogleCloudOptions.OAuth2.ClientSecret:** OAuth2 프로토콜을 통해 인증하기 위해 Google이 제공한 Client Secret 문자열입니다.
- **GoogleCloudOptions.OAuth2.Scope:** OAuth2의 스코프입니다. 일반적으로 기본값보다 더 많은 접근이 필요하지 않은 한 기본값을 수정할 필요가 없습니다.
- **GoogleCloudOptions.OAuth2.LocalIP:** OAuth2 프로토콜은 인증 서버로부터의 응답을 수신 대기하는 로컬 서버가 필요합니다. 이것은 IP 또는 DNS 이름입니다. 기본적으로 127.0.0.1입니다.
- **GoogleCloudOptions.OAuth2.LocalPort:** 로컬 서버 수신 대기 포트입니다.

- **GoogleCloudOptions.OAuth2.RedirectURL:** LocalPort + LocalIP와 다른 리디렉션 URL을 설정해야 하는 경우, 이 속성에 설정할 수 있습니다(예: `http://127.0.0.1:8080/oauth2`).

Scopes 속성을 사용하여 클라이언트 API의 Scopes를 수정할 수 있습니다. 클라이언트가 지원하는 scope를 선택하기만 하면 됩니다.

JWT

로그인은 google이 제공하는 개인 키를 사용하여 요청에 서명하여 수행되며, 이 방법은 사용자 상호 작용이 없는 자동화된 서비스나 애플리케이션에 권장됩니다. **Domain-Wide Delegation**으로 Service Account를 구성해야 합니다.

- **GoogleCloudOptions.JWT.ClientEmail:** 새 서비스 계정을 생성할 때 제공된 클라이언트 이메일 이름. JSON 구성 파일의 "client_email" 노드.
- **GoogleCloudOptions.JWT.PrivateKeyId:** Google에서 제공하는 Private Key ID입니다. JSON 구성 파일의 "private_key_id" 노드입니다.
- **GoogleCloudOptions.JWT.PrivateKey:** Google이 제공한 비공개 키 인증서입니다. JSON 구성 파일의 "private_key" 노드입니다.
- **GoogleCloudOptions.JWT.Subject:** Domain-Wide Delegation을 사용하여 서비스 계정에 연결된 워크스페이스 이메일 계정입니다.

TLS Options

Google 서버와 설정된 보안 연결을 사용자 지정하려면 **TLSOptions** 속성을 사용하십시오.

- **TLSOptions.IOHandler:** TLS 스택을 선택합니다(OpenSSL, SChannel 등).
- **TLSOptions.Version:** 특정 TLS 프로토콜 버전을 강제합니다(예: `tls1_2` 또는 `tls1_3`).
- **TLSOptions.VerifyCertificate:** 서버 인증서 검증을 활성화하거나 비활성화합니다.
- **TLSOptions.OpenSSL_Options.LibPath:** 애플리케이션과 함께 배포된 OpenSSL 라이브러리가 들어 있는 폴더를 가리킵니다.
- **TLSOptions.Schannel_Options.UseLegacyCredentials:** SChannel을 사용할 때 일부 Windows 버전에서 요구되는 레거시 자격 증명 흐름을 활성화합니다.

다음 스니펫은 TLS 옵션을 구성하는 방법을 보여줍니다:

```
TsgcHTTPGoogleCloud_Calendar_Client *oCalendar = new TsgcHTTPGoogleCloud_Calendar_Client(NULL);
oCalendar->TLSOptions->IOHandler = iohOpenSSL;
oCalendar->TLSOptions->Version = tls1_3;
oCalendar->TLSOptions->VerifyCertificate = true;
oCalendar->TLSOptions->OpenSSL_Options->LibPath = opensslDefaultFolder;
```

Most common uses

- 구성
 - [Google Calendar 갱신 토큰](#)
 - [Google Calendar 서비스 계정](#)
- Synchronization
 - [Google Calendar Sync Calendars](#)
 - [Google Calendar Sync Events](#)

캘린더 동기화

TsgcHTTPGoogleCloud_Calendar_Client 구성 요소를 사용하면 직접 Google API 호출을 사용하거나 간편한 Calendars 메서드를 사용하여 캘린더를 동기화할 수 있습니다.

메서드	매개변수	Description
NewCalendar	aSummary: 캘린더의 제목입니다.	새 Calendar를 생성합니다
DeleteCalendar	aId: 캘린더의 식별자.	기존 Calendar를 삭제합니다.

UpdateCalendar	aResource: 캘린더 데이터가 있는 객체입니다.	기존 캘린더를 업데이트합니다.
LoadCalendars		모든 캘린더를 로드하고 이 데이터로 Calendars 속성을 채웁니다.
LoadCalendarsChanged	aSyncToken: 캘린더를 업데이트하는데 사용된 마지막 토큰.	Token 집합에서 캘린더의 모든 변경 사항을 로드합니다.

Calendar 클라이언트에는 **Calendars**라는 속성이 있으며, 이전 메서드 중 하나를 호출한 후 캘린더 데이터에 액세스할 수 있습니다. 이 속성은 자동으로 동기화됩니다.

Synchronize Events

TsgcHTTPGoogleCloud_Calendar_Client 구성 요소를 사용하면 직접 Google API 호출을 사용하거나 간편한 Event 메서드를 사용하여 이벤트를 동기화할 수 있습니다.

메서드	매개변수	Description
NewEvent	aCalendarId: 캘린더의 id입니다. aResource: 이벤트 데이터가 있는 객체입니다.	새 Event를 생성합니다.
DeleteEvent	aCalendarId: 캘린더의 id입니다. aid: 이벤트의 식별자입니다.	기존 Event를 삭제합니다.
UpdateEvent	aCalendarId: 캘린더의 id. aResource: 이벤트 데이터가 있는 객체.	기존 Event를 업데이트합니다.
LoadEvents	aCalendarId: 캘린더의 id입니다.	캘린더의 모든 이벤트를 로드합니다.
LoadEventsChanged	aCalendarId: 캘린더의 id입니다. aSyncToken: 캘린더를 업데이트하는데 사용된 마지막 토큰입니다.	Token 집합에서 캘린더의 모든 이벤트를 로드합니다.

Calendars 속성을 사용하여 이벤트 데이터에 접근할 수 있습니다. 목록에서 기존 캘린더를 선택하고 해당 **Events** 속성에 접근하십시오.

Google Calendar API Calls

메서드	Description

ACL_Delete	접근 제어 규칙을 삭제합니다.
ACL_Get	접근 제어 규칙을 반환합니다.
ACL_Insert	액세스 제어 규칙을 만듭니다.
ACL_List	캘린더의 접근 제어 목록에 있는 규칙을 반환합니다.
ACL_Patch	액세스 제어 규칙을 업데이트합니다. 이 메서드는 patch 의미를 지원합니다.
ACL_Update	접근 제어 규칙을 업데이트합니다.
ACL_Watch	ACL 리소스의 변경 사항을 감시합니다.

메서드	Description
CalendarList_Delete	사용자의 calendar 목록에서 calendar를 제거합니다.
CalendarList_Get	사용자의 캘린더 목록에서 캘린더를 반환합니다.
CalendarList_Insert	기존 캘린더를 사용자의 캘린더 목록에 삽입합니다.
CalendarList_List	사용자의 캘린더 목록에 있는 캘린더를 반환합니다.
CalendarList_Patch	사용자의 캘린더 목록에서 기존 캘린더를 업데이트합니다. 이 메서드는 패치 시맨틱을 지원합니다.
CalendarList_Update	사용자의 캘린더 목록에서 기존 캘린더를 업데이트합니다.
CalendarList_Watch	CalendarList 리소스의 변경을 감시합니다.

메서드	Description
Calendar_Clear	기본 캘린더를 지웁니다. 이 작업은 계정의 기본 캘린더와 연결된 모든 이벤트를 삭제합니다.
Calendar_Delete	보조 캘린더를 삭제합니다. 기본 캘린더의 모든 이벤트를 지우려면 calendars.clear를 사용하십시오.
Calendar_Get	캘린더의 메타데이터를 반환합니다.
Calendar_Insert	보조 캘린더를 생성합니다.
Calendar_Patch	캘린더의 메타데이터를 업데이트합니다. 이 메서드는 patch 의미를 지원합니다.

Calendar_Update	캘린더의 메타데이터를 업데이트합니다.

메서드	Description
Channel_Stop	이 채널을 통한 리소스 감시를 중지합니다.

메서드	Description
Color_Get	캘린더 및 이벤트에 대한 색상 정의를 반환합니다.

메서드	Description
Event_Delete	이벤트를 삭제합니다.
Event_Get	이벤트를 반환합니다.
Event_Import	이벤트를 가져옵니다. 이 작업은 기존 이벤트의 비공개 사본을 캘린더에 추가하는 데 사용됩니다.
Event_Insert	이벤트를 생성합니다.
Event_Instances	지정된 반복 이벤트의 인스턴스를 반환합니다.
Event_List	지정된 캘린더의 이벤트를 반환합니다.
Event_Move	이벤트를 다른 캘린더로 이동합니다. 즉, 이벤트의 organizer를 변경합니다.
Event_Patch	이벤트를 업데이트합니다. 이 메서드는 patch 의미 체계를 지원합니다. 지정된 필드 값은 기존 값을 대체합니다. 요청에서 지정하지 않은 필드는 변경되지 않은 상태로 유지됩니다. 배열 필드가 지정된 경우 기존 배열을 덮어쓰며, 이는 이전 배열 요소를 모두 삭제합니다.
Event_QuickAdd	간단한 텍스트 문자열을 기반으로 이벤트를 생성합니다.
Event_Update	이벤트를 업데이트합니다.
Event_Watch	Events 리소스의 변경 사항을 감시합니다.

메서드	Description
Freebusy_Query	캘린더 집합에 대한 free/busy 정보를 반환합니다.

메서드	Description
Settings_Get	단일 사용자 설정을 반환합니다.
Settings_List	인증된 사용자의 모든 사용자 설정을 반환합니다.
Settings_Watch	Settings 리소스의 변경 사항을 감시합니다.

Switch Account

한 계정에서 다른 계정으로 전환하려면 먼저 **Clear** 메서드를 호출하여 현재 캘린더 세션을 지우고 새 세션을 구성하십시오.

Google Calendar | Load Calendars

계정에서 모든 캘린더를 가져오는 프로세스는 매우 쉽습니다. 다음 단계를 따르기만 하면 됩니다:

1. **LoadCalendars** 메서드를 호출하십시오.
2. 메서드가 True를 반환하면 **Calendars** 속성에 액세스하여 목록을 반복하여 모든 Calendar에 액세스할 수 있습니다.

```
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

캘린더에서 모든 이벤트를 가져오는 프로세스는 매우 쉽습니다. 다음 단계를 따르기만 하면 됩니다:

1. **LoadEvents** 메서드를 호출하고 **CalendarId**를 매개변수로 전달하십시오.
2. 메서드가 True를 반환하면 **Calendars.Events** 속성에 액세스하고 목록을 반복하여 캘린더의 모든 Events에 액세스할 수 있습니다.

```
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

Google Calendar API는 OAuth2를 사용하여 Google 서버에 대해 인증합니다. sgcWebSockets에는 전체 인증 프로세스를 처리하는 구성 요소가 있지만, 애플리케이션이 닫히고 다시 연결을 시도하는 경우 두 가지 옵션이 있습니다:

1. Google API를 사용하여 다시 인증하십시오
2. Refresh Token을 사용하십시오(여전히 유효한 경우). 그러면 인증 프로세스를 피할 수 있습니다.

Using RefreshToken

처음 인증할 때 OnAuthToken 이벤트를 사용하여 **RefreshToken**이 존재하면 저장하십시오. 예를 들어 INI 파일에 저장할 수 있습니다:

```
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();
    }
}
```

그런 다음 애플리케이션을 다시 시작할 때 RefreshToken이 있으면 RefreshToken 메서드를 호출하고 토큰을 인수로 전달하십시오(먼저 Google Calendar API 키를 설정해야 함). 성공하면 재인증할 필요 없이 Google servers에 로그인됩니다.

```
GoogleCalendar->RefreshToken("your refresh token here");
```

Google Calendar | Service Account

Google Calendar 클라이언트는 사용자 상호 작용 없이 서비스로 작동할 수 있으므로 Windows 서비스, 데몬 등을 실행하려는 경우에 유용합니다.

Google Cloud는 이 유형의 프로젝트를 실행하기 위해 OAuth2 자격 증명 대신 **서비스 계정**을 생성하도록 요구하며, Google Calendar API는 캘린더에 액세스하는 데 필요한 자격 증명을 얻기 위해 서비스 계정이 **도메인 전체 위임**을 사용하도록 요구합니다.

[Google Service Accounts](#) 생성 방법에 대해 자세히 읽을 수 있습니다.

Google Cloud 계정이 service account로 구성되고 Domain-Wide Delegation을 사용하여 workspace 이메일 계정에 연결되면, 다음 단계를 따라 Google Calendar 클라이언트가 이와 함께 작동하도록 구성할 수 있습니다.

- **GoogleCloudOptions.Authentication** 속성을 **gcaJWT** 값으로 설정하십시오.
- **LoadSettingsFromFile** 메서드를 사용하여 Google Cloud 계정에서 생성된 JSON 파일을 가져옵니다. 이 파일에는 JWT를 암호화하는 개인 키와 클라이언트에 필요한 기타 속성이 포함되어 있습니다.
- JSON 파일을 가져온 후 다음 속성이 자동으로 채워집니다:
 - **ClientEmail**: 서비스 계정 이름입니다
 - **PrivateKeyId**: 비공개 키 파일의 id입니다
 - **PrivateKey**: 비공개 키 파일입니다
- 마지막으로, **GoogleCloudOptions.JWT.Subject** 속성에 서비스 계정에 연결된 Workspace 이메일 계정을 설정하십시오.

클라이언트를 구성한 후에는 사용자 상호작용 없이 Google Calendar API에 요청을 보낼 수 있습니다.

Google Cloud FCM

Firebase Cloud Messaging (FCM)은 무료로 메시지를 안정적으로 보낼 수 있는 크로스 플랫폼 메시징 솔루션입니다.

FCM을 사용하면 새 이메일이나 기타 데이터를 동기화할 수 있음을 클라이언트 앱에 알릴 수 있습니다. 사용자 재참여 및 유지를 유도하기 위해 알림 메시지를 보낼 수 있습니다. 인스턴트 메시징 같은 사용 사례의 경우, 메시지는 최대 4096바이트의 페이로드를 클라이언트 앱에 전송할 수 있습니다.

구성 요소는 HTTP v1 API를 지원합니다.

권한 부여

Google FCM 구성 요소 클라이언트는 다음 방법으로 Google 서버에 로그인할 수 있습니다:

- **gcaOAuth2:** OAuth2 프로토콜
- **gcaJWT:** JWT 토큰.

OAuth2

로그인은 사용자가 자신의 계정으로 로그인하고 FCM 요청을 승인하는 웹 브라우저를 사용하여 수행됩니다.

- **GoogleCloudOptions.OAuth2.ClientId:** OAuth2 프로토콜을 통해 인증하기 위해 Google에서 제공하는 ClientID입니다.
- **GoogleCloudOptions.OAuth2.ClientSecret:** OAuth2 프로토콜을 통해 인증하기 위해 Google이 제공한 Client Secret 문자열입니다.
- **GoogleCloudOptions.OAuth2.Scope:** OAuth2의 스코프입니다. 일반적으로 기본값보다 더 많은 접근이 필요하지 않은 한 기본값을 수정할 필요가 없습니다.
- **GoogleCloudOptions.OAuth2.LocalIP:** OAuth2 프로토콜은 인증 서버로부터의 응답을 수신 대기하는 로컬 서버가 필요합니다. 이것은 IP 또는 DNS 이름입니다. 기본적으로 127.0.0.1입니다.
- **GoogleCloudOptions.OAuth2.LocalPort:** 로컬 서버 수신 대기 포트입니다.
- **GoogleCloudOptions.OAuth2.RedirectURL:** LocalPort + LocalIP와 다른 리디렉션 URL을 설정해야 하는 경우, 이 속성에 설정할 수 있습니다(예: http://127.0.0.1:8080/oauth2).

Service Accounts

로그인은 Google이 제공한 private key로 요청에 서명하여 수행됩니다. 이 메서드는 사용자 상호 작용이 없는 자동화된 서비스 또는 애플리케이션에 권장됩니다.

- **GoogleCloudOptions.JWT.ClientEmail:** 새 서비스 계정을 생성할 때 제공된 클라이언트 이메일 이름. JSON 구성 파일의 "client_email" 노드.
- **GoogleCloudOptions.JWT.Subject:** 새 서비스 계정을 생성할 때 제공되는 클라이언트 이메일 이름입니다. JSON 구성 파일의 "client_email" 노드입니다.
- **GoogleCloudOptions.JWT.PrivateKeyId:** Google에서 제공하는 Private Key ID입니다. JSON 구성 파일의 "private_key_id" 노드입니다.
- **GoogleCloudOptions.JWT.PrivateKey:** Google이 제공한 비공개 키 인증서입니다. JSON 구성 파일의 "private_key" 노드입니다.

TLS Options

Firebase Cloud Messaging 연결은 **TLSOptions** 속성을 통해 조정할 수 있습니다.

- **TLSOptions.IOHandler:** 사용할 TLS 스택을 선택하십시오(OpenSSL, SChannel...).
- **TLSOptions.Version:** 프로젝트에 필요한 특정 TLS 프로토콜 버전을 요청합니다.
- **TLSOptions.VerifyCertificate:** Google에 연결할 때 서버 인증서 검증을 강제합니다.
- **TLSOptions.OpenSSL_Options.LibPath:** OpenSSL 런타임 라이브러리가 배포되는 위치를 정의합니다.
- **TLSOptions.SChannel_Options.UseLegacyCredentials:** Windows SChannel이 레거시 자격 증명 협상을 요구할 때 활성화하십시오.

Example configuration:

```
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;
```

새 서비스 계정이 생성되면 모든 구성이 포함된 JSON 파일을 다운로드할 수 있습니다. 이 파일은 FCM 구성 요소에서 처리할 수 있습니다. **LoadSettingsFromFile** 메서드를 호출하고 JSON 파일 이름을 인수로 전달하기만 하면 됩니다.

Most common uses

- 구성
 - [Google OAuth2 Keys](#)
 - [Service Accounts](#)

Google FCM Client

OAuth2

Google FCM API로 작업하려면 `sgcWebSockets` FCM 구성 요소가 기본 인증 방법으로 OAuth2를 사용하므로 먼저 Google 계정에서 **ClientId**와 **ClientSecret**을 설정해야 합니다.

```
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

서비스 계정에는 JWT를 빌드하여 권한 부여 토큰으로 전달하는 것이 필요합니다.

```
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...";
```

이것은 모든 Rest API 호출에 사용될 Authorization Token Key를 Google로부터 얻기 위해 필요합니다.

모든 메서드는 응답을 반환하며, 이는 성공하거나 오류를 반환할 수 있습니다.

메시지 보내기

알림 메시지를 보내려면 `SendMessage` 메서드를 사용하십시오. 이 함수에는 2개의 인수가 있습니다:

1. Project: google cloud 계정의 프로젝트 id입니다.
2. Payload: json 텍스트 메시지입니다. 예:

```
{
  "message": {
    "topic": "news",
    "notification": {
      "title": "Breaking News",
      "body": "New news story available."
    }
  },
  "data": {
```

```
    "story_id": "story_12345"  
  }  
}
```

TsgcWebView2

TsgcWebView2는 Microsoft Edge WebView2(Chromium)를 래핑하는 시각적 VCL 구성 요소입니다. 폼에 드롭하여 Delphi 애플리케이션에 현대적이고 완전한 기능을 갖춘 웹 브라우저를 임베드하십시오. Delphi 7부터 Delphi 13까지 지원합니다.

요구사항

- Microsoft Edge WebView2 Runtime(Windows 10/11에 포함되거나 Microsoft에서 다운로드 가능)
- WebView2Loader.dll(포함됨, 실행 파일 옆에 배치)
- Windows only

빠른 시작

폼에 TsgcWebView2를 배치하고 URL로 이동합니다.

```
sgcWebView21 := TsgcWebView2.Create(Self);
sgcWebView21.Parent := Self;
sgcWebView21.Align := alClient;
sgcWebView21.DefaultURL := 'https://www.example.com';
```

Features

- **Navigation**
 - [Navigate](#), [NavigateToString](#), [GoBack](#), [GoForward](#), [Reload](#), [Stop](#), [POST navigation](#)
- **JavaScript**
 - [ExecuteScript \(async\)](#), [ExecuteScriptSync](#), [AddInitScript](#)
- **Cookie Management**
 - [GetCookies](#), [AddOrUpdateCookie](#), [DeleteCookie](#), [DeleteAllCookies](#)
- **다운로드 제어**
 - [OnDownloadStarting](#), [OnDownloadProgress](#), [OnDownloadCompleted](#)
- **설정 및 구성**
 - [ScriptEnabled](#), [DevToolsEnabled](#), [ContextMenuEnabled](#) 등
- **고급 기능**
 - [인쇄](#), [오디오/음소거](#), [인증서](#), [Favicon](#), [가상 호스트](#), [스크린샷](#)

Properties

- **DefaultURL**: string — 초기화 후 탐색할 URL입니다.
- **AutoInitialize**: Boolean — True인 경우 윈도우 핸들이 할당될 때 WebView2가 자동으로 생성됩니다. 기본값: True.
- **ZoomFactor**: Double — zoom 수준(1.0 = 100%).
- **UserDataFolder**: string — 쿠키, 캐시, 권한 저장 경로입니다.
- **BrowserExecutableFolder**: string — 고정 버전 WebView2 Runtime 배포의 경로입니다.
- **Settings**: TsgcWebView2Settings — 디자인타임 브라우저 설정(ScriptEnabled, DevToolsEnabled 등)입니다.
- **URL**: string (읽기 전용) — 현재 페이지 URL입니다.
- **DocumentTitle**: string (읽기 전용) — 현재 페이지 제목.
- **CanGoBack**: Boolean(읽기 전용) — 뒤로 탐색이 가능한지 여부입니다.
- **CanGoForward**: Boolean(읽기 전용) — 앞으로 탐색이 가능한지 여부입니다.
- **Initialized**: Boolean (읽기 전용) — WebView2가 준비되었는지 여부입니다.
- **IsMuted**: Boolean — 오디오 음소거/음소거 해제.
- **IsDocumentPlayingAudio**: Boolean(읽기 전용) — 페이지가 오디오를 재생 중인지 여부입니다.
- **FaviconURI**: string (읽기 전용) — 현재 페이지 파비콘의 URI입니다.
- **StatusBarText**: string (읽기 전용) — 현재 상태 표시줄 텍스트.
- **CookieManager**: TsgcWebView2CookieManager(읽기 전용) — 쿠키 관리 객체입니다.
- **WebView**: ICoreWebView2 (읽기 전용) — 직접 COM 인터페이스 액세스.
- **Controller**: ICoreWebView2Controller(읽기 전용) — 직접 COM 인터페이스 액세스입니다.
- **Environment**: ICoreWebView2Environment(읽기 전용) — 직접 COM 인터페이스 접근입니다.

Methods

- **Navigate**(aURL: string) — URL로 이동합니다.
- **NavigateToString**(aHTML: string) — HTML 콘텐츠를 직접 로드합니다.
- **GoBack / GoForward** — 기록을 탐색합니다.
- **Reload / Stop** — 로딩을 다시 로드하거나 중지합니다.
- **ExecuteScript**(aScript: string) — JavaScript를 비동기적으로 실행합니다(결과는 OnScriptExecuted에 있음).
- **ExecuteScriptSync**(aScript: string): string — JavaScript를 동기적으로 실행하고 JSON 결과를 반환합니다.
- **AddInitScript**(aScript: string) — 모든 페이지 로드 시 실행되는 JavaScript를 추가합니다.
- **RemoveInitScript**(aScriptId: string) — 이전에 추가한 init script를 제거합니다.
- **NavigateWithPostData**(aURL, aMethod, aPostData, aHeaders: string) — 사용자 정의 HTTP method, body 및 headers로 이동합니다.
- **PrintToPdf**(aFilePath: string) — 페이지를 PDF로 저장합니다.
- **ShowPrintUI** — 네이티브 인쇄 대화 상자를 표시합니다.
- **CapturePreviewToFile**(aFilePath: string; aFormat: Integer) — 스크린샷(0=PNG, 1=JPEG)입니다.
- **OpenDevToolsWindow** — Edge DevTools를 엽니다.
- **OpenTaskManagerWindow** — Edge 작업 관리자를 엽니다.
- **SetVirtualHostNameToFolderMapping**(aHostName, aFolderPath: string; aAccessKind: Integer) — 호스트 이름을 로컬 폴더에 매핑합니다.
- **ClearVirtualHostNameToFolderMapping**(aHostName: string) — 매핑을 제거합니다.
- **PostWebMessageAsString**(aMessage: string) — 웹 콘텐츠에 메시지를 보냅니다.
- **PostWebMessageAsJson**(aJson: string) — 웹 콘텐츠에 JSON을 보냅니다.
- **GetProfileName**: string — 브라우저 프로필 이름을 가져옵니다.
- **ClearBrowsingData**(aKinds: Cardinal) — 특정 브라우징 데이터 유형을 지웁니다.
- **ClearAllBrowsingData** — 모든 브라우징 데이터를 지웁니다.
- **PostSharedBufferToScript**(aSharedBuffer: IUnknown; aAccess: Integer; aAdditionalDataAsJson: string) — 웹 콘텐츠와 메모리 버퍼를 공유합니다.
- **InitializeWebView / FinalizeWebView** — 수동 초기화 제어.

Events

- **OnInitialized** — WebView2가 준비되면 발생합니다.
- **OnNavigationStarting**(aURL; alsUserInitiated, alsRedirected: Boolean; var aCancel: Boolean) — 탐색 전.
- **OnNavigationCompleted**(alsSuccess: Boolean; aWebErrorStatus: Integer) — 내비게이션 후.
- **OnSourceChanged**(alsNewDocument: Boolean) — URL이 변경되었습니다.
- **OnDocumentTitleChanged**(aTitle: string) — 페이지 제목이 변경되었습니다.
- **OnWebMessageReceived**(aSource, aWebMessageAsJson, aWebMessageAsString: string) — 웹 콘텐츠로 부터의 메시지입니다.
- **OnNewWindowRequested**(aURL: string; alsUserInitiated: Boolean; var aHandled: Boolean) — 팝업/새 탭 요청.
- **OnScriptExecuted**(aErrorCode: HRESULT; aResultAsJson: string) — 비동기 JS 결과.
- **OnContentLoading**(alsErrorPage: Boolean) — 콘텐츠 로딩이 시작되었습니다.
- **OnHistoryChanged** — 탐색 기록이 변경되었습니다.
- **OnProcessFailed**(aKind: Integer) — 브라우저 프로세스가 충돌했습니다.
- **OnPermissionRequested**(aURL: string; aKind: Integer; alsUserInitiated: Boolean; var aState: Integer) — 권한 요청.
- **OnWindowCloseRequested** — window.close()가 호출되었습니다.
- **OnDownloadStarting**(aURL, aResultFilePath: string; var aCancel, aHandled: Boolean; var aFilePath: string) — 다운로드 시작됨.
- **OnDownloadProgress**(aBytesReceived, aTotalBytes: Int64) — 다운로드 진행 상황.
- **OnDownloadCompleted**(aFilePath: string; aState: Integer) — 다운로드가 완료되었습니다.
- **OnContextMenuRequested**(aMenuItems: string; aContextKind: Integer; aLocation: TPoint; var aHandled: Boolean) — 우클릭 메뉴입니다.
- **OnFaviconChanged**(aFaviconURL: string) — 페이지 아이콘 변경됨.
- **OnClientCertificateRequested**(aHost: string; aPort: Integer; var aHandled: Boolean) — 클라이언트 인증서가 필요합니다.
- **OnServerCertificateError**(aRequestURL: string; aErrorStatus: Integer; var aAction: Integer) — 인증서 오류입니다.
- **OnStatusBarTextChanged**(aText: string) — 상태 표시줄 텍스트가 변경되었습니다.
- **OnDOMContentLoaded** — DOM 준비됨.
- **OnBasicAuthRequested**(aURL: string; var aUserName, aPassword: string; var aHandled: Boolean) — HTTP basic auth입니다.
- **OnCapturePreviewCompleted**(aErrorCode: HRESULT) — 스크린샷이 완료되었습니다.

TsgcWebView2 | Navigation

TsgcWebView2는 URL로 이동하고, HTML 콘텐츠를 로드하며, 브라우저 기록을 제어하기 위한 여러 메서드를 제공합니다.

기본 내비게이션

URL을 로드하려면 **Navigate** 메서드를 사용하십시오. 구성 요소는 요청 전에 `OnNavigationStarting`을, 완료될 때 `OnNavigationCompleted`를 발생시킵니다.

```
// Navigate to a URL
sgcWebView21.Navigate('https://www.example.com');
```

디자인 타임 또는 초기화 전에 **DefaultURL** 속성을 설정할 수도 있습니다. 구성 요소는 WebView2가 준비된 후 이 URL로 자동으로 이동합니다.

```
// Set the default URL before initialization
sgcWebView21.DefaultURL := 'https://www.example.com';
```

HTML Content

서버나 파일 없이 HTML 콘텐츠를 직접 로드하려면 **NavigateToString**을 사용하십시오.

```
// 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

브라우저 기록을 탐색하려면 **GoBack**과 **GoForward**를 사용하십시오. 이러한 메서드를 호출하기 전에 **CanGoBack**과 **CanGoForward**를 확인하십시오.

```
// Navigate back
if sgcWebView21.CanGoBack then
  sgcWebView21.GoBack;

// Navigate forward
if sgcWebView21.CanGoForward then
  sgcWebView21.GoForward;
```

현재 페이지를 새로 고치려면 **Reload**를 사용하고 보류 중인 탐색을 취소하려면 **Stop**을 사용하십시오.

```
// Reload the current page
sgcWebView21.Reload;

// Stop loading
sgcWebView21.Stop;
```

POST Navigation

사용자 지정 메서드, 본문, 헤더로 HTTP 요청을 보내려면 **NavigateWithPostData**를 사용하십시오. 이는 폼 데이터를 제출하거나 브라우저에서 직접 REST API를 호출하는 데 유용합니다.

```
// 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');
```

탐색 이벤트

탐색이 시작되기 전에 탐색을 검사하거나 취소하려면 **OnNavigationStarting**을 사용하십시오. 요청을 차단하려면 **aCancel**을 True로 설정하십시오.

```
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;
```

탐색이 성공했는지 실패했는지 확인하려면 **OnNavigationCompleted**를 사용하십시오.

```
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;
```

새 창 처리

페이지가 팝업 또는 target="_blank"가 있는 링크를 열면 **OnNewWindowRequested** 이벤트가 발생합니다. 기본 팝업을 방지하고 대신 동일한 창에서 탐색하려면 **aHandled**를 True로 설정하십시오.

```
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

TsgcWebView2는 브라우저 컨텍스트에서 JavaScript를 실행하고, 결과를 검색하고, Delphi 애플리케이션과 웹 콘텐츠 간의 양방향 통신을 설정하는 메서드를 제공합니다.

비동기 실행

JavaScript를 비동기적으로 실행하려면 **ExecuteScript**를 사용하십시오. 결과는 **OnScriptExecuted** 이벤트에서 JSON 문자열로 반환됩니다.

```
// 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;
```

유효한 모든 JavaScript 표현식을 실행할 수 있습니다. 반환 값은 항상 JSON으로 직렬화됩니다.

```
// 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

결과를 직접 반환하는 블로킹 실행에는 **ExecuteScriptSync**를 사용하십시오. 값이 즉시 필요할 때 더 간단하지만, 스크립트가 완료될 때까지 호출 스레드를 블로킹합니다.

```
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

페이지의 다른 스크립트보다 먼저, 모든 페이지 로드 시 자동으로 실행되는 JavaScript를 등록하려면 **AddInitScript**를 사용하십시오. 폴리필을 주입하거나, 브라우저 API를 재정의하거나, 메시지 핸들러를 설정하는 데 유용합니다.

```
// 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);' +
  '};');
```

이전에 추가된 init 스크립트를 ID로 등록 해제하려면 **RemoveInitScript**를 사용하십시오.

Web Messaging

웹 메시징은 Delphi 애플리케이션과 브라우저에서 실행되는 JavaScript 간의 양방향 통신을 제공합니다. Delphi에서 JavaScript로 데이터를 보내려면 **PostWebMessageAsString** 또는 **PostWebMessageAsJson**을 사용하고, JavaScript에서 데이터를 받으려면 **OnWebMessageReceived**를 처리하십시오.

Delphi에서 JavaScript로 메시지 전송:

```
// Send a plain string message
sgcWebView21.PostWebMessageAsString('Hello from Delphi!');

// Send a JSON message
sgcWebView21.PostWebMessageAsJson('{"action":"refresh","id":42}');
```

JavaScript에서 메시지 수신:

```
// 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);' +
  '});');
```

Delphi에서 JavaScript로부터 메시지 수신:

```
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;
```

JavaScript에서 다음을 사용하여 Delphi에 메시지를 보냅니다:

```
// 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

TsgcWebView2는 브라우저 컨텍스트에서 쿠키를 읽고, 생성하고, 업데이트하고, 삭제하기 위한 **CookieManager** 속성을 노출합니다.

Cookie 가져오기

주어진 URI에 대한 쿠키를 검색하려면 **CookieManager.GetCookies**를 사용하십시오. 이 메서드는 반복할 수 있는 쿠키 객체 목록을 반환합니다.

```
procedure TFormMain.ButtonGetCookiesClick(Sender: TObject);
var
  vCookies: TsgcWebView2CookieList;
  i: Integer;
begin
  vCookies := sgcWebView21.CookieManager.GetCookies(
    'https://www.example.com');
  try
    for i := 0 to vCookies.Count - 1 do
      Memo1.Lines.Add(
        vCookies[i].Name + ' = ' + vCookies[i].Value);
    finally
      vCookies.Free;
    end;
  end;
end;
```

Adding Cookies

새 쿠키를 생성하거나 기존 쿠키를 업데이트하려면 **CookieManager.AddOrUpdateCookie**를 사용하십시오. 최소한 이름, 값, 도메인 및 경로를 지정하십시오.

```
// 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

이름과 URI로 특정 쿠키를 제거하려면 **DeleteCookie**를 사용하십시오. 브라우저에서 모든 쿠키를 지우려면 **DeleteAllCookies**를 사용하십시오.

```
// 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

TsgcWebView2는 브라우저가 시작한 파일 다운로드를 가로채고, 모니터링하고, 제어하는 이벤트를 제공합니다.

Download Events

OnDownloadStarting 이벤트는 다운로드가 시작될 때 발생합니다. 다운로드를 취소하거나, 대상 파일 경로를 변경하거나, 기본 동작으로 진행하도록 둘 수 있습니다.

```
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

OnDownloadProgress 이벤트는 다운로드 중에 주기적으로 발생하며, 수신된 바이트와 예상되는 총 바이트를 보고합니다.

```
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;
```

다운로드 완료

OnDownloadCompleted 이벤트는 다운로드가 완료될 때 발생합니다. 다운로드가 성공했는지, 취소되었는지, 실패했는지 확인하려면 **aState**를 확인하십시오.

```
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.');
```

사용자 정의 다운로드 경로

다운로드를 사용자 정의 위치로 리디렉션하려면 OnDownloadStarting에서 **aFilePath** 매개변수를 설정하십시오. 기본 저장 대화 상자를 억제하려면 **aHandled**를 True로 설정하십시오.

```
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

TsgcWebView2는 브라우저 동작을 제어하는 TsgcWebView2Settings 유형의 **Settings** 속성을 노출합니다. 이러한 설정은 디자인 타임에 구성하거나 초기화 후 런타임에 변경할 수 있습니다.

TsgcWebView2Settings Properties

- **ScriptEnabled**: Boolean — JavaScript 실행을 활성화하거나 비활성화합니다. 기본값: True.
- **WebMessageEnabled**: Boolean — 웹 메시징(postMessage)을 활성화하거나 비활성화합니다. 기본값: True.
- **DefaultScriptDialogsEnabled**: Boolean — 기본 JavaScript 대화 상자(alert, confirm, prompt)를 활성화하거나 비활성화합니다. 기본값: True.
- **StatusBarEnabled**: Boolean — 브라우저 상태 표시줄을 표시하거나 숨깁니다. 기본값: True.
- **DevToolsEnabled**: Boolean — Edge DevTools에 대한 액세스를 허용하거나 차단합니다. 기본값: True.
- **ContextMenuEnabled**: Boolean — 브라우저 마우스 오른쪽 버튼 컨텍스트 메뉴를 활성화 또는 비활성화합니다. 기본값: True.
- **ZoomControlEnabled**: Boolean — 사용자가 시작한 확대/축소(Ctrl+스크롤, Ctrl+더하기/빼기)를 허용하거나 차단합니다. 기본값: True.
- **BuiltInErrorPageEnabled**: Boolean — 탐색 실패에 대한 내장 오류 페이지를 표시하거나 숨깁니다. 기본값: True.

Design-Time Configuration

Object Inspector에서 Settings 속성을 설정하십시오. WebView2 환경이 초기화된 후 자동으로 적용됩니다.

```
// 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

구성 요소가 초기화된 후 런타임에 설정을 변경할 수 있습니다. 변경 사항은 즉시 적용됩니다.

```
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

UserDataFolder 속성은 브라우저가 쿠키, 캐시, 권한 및 기타 프로필 데이터를 저장하는 디렉토리를 지정합니다. 각 고유한 폴더는 격리된 브라우저 프로필을 생성합니다.

```
// Use a custom data folder for isolated profiles
sgcWebView21.UserDataFolder := 'C:\AppData\MyApp\Profile1';
```

초기화 전에(구성 요소가 WebView2 환경을 생성하기 전에) 이 속성을 설정하십시오. 설정하지 않으면 애플리케이션의 임시 디렉터리에 기본 폴더가 사용됩니다.

BrowserExecutableFolder

BrowserExecutableFolder 속성은 고정 버전 WebView2 Runtime 배포를 가리킵니다. 시스템 설치 런타임에 의존하는 대신 애플리케이션과 함께 특정 브라우저 버전을 배포하려면 이를 사용하십시오.

```
// Use a fixed-version runtime bundled with the application
sgcWebView21.BrowserExecutableFolder :=
  ExtractFilePath(ParamStr(0)) + 'WebView2Runtime';
```

이 속성을 초기화 전에 설정하십시오. 비워 두면 구성 요소는 시스템에 설치된 Evergreen WebView2 Runtime을 사용합니다.

TsgcWebView2 | 고급 기능

TsgcWebView2는 인쇄, 스크린샷, 오디오 제어, 인증서 처리 등을 포함한 고급 브라우저 기능에 대한 액세스를 제공합니다.

Print Support

현재 페이지를 PDF 파일로 저장하려면 **PrintToPdf**를 사용하고, 네이티브 인쇄 대화 상자를 표시하려면 **ShowPrintUI**를 사용하십시오.

```
// Save the current page as PDF
sgcWebView21.PrintToPdf('C:\Output\page.pdf');
```

```
// Show the native print dialog
sgcWebView21.ShowPrintUI;
```

스크린샷 캡처

현재 페이지의 스크린샷을 찍으려면 **CapturePreviewToFile**을 사용하십시오. format 매개변수는 이미지 유형을 지정합니다: PNG의 경우 0, JPEG의 경우 1. 캡처가 완료되면 **OnCapturePreviewCompleted** 이벤트가 발생합니다.

```
// Capture as PNG
sgcWebView21.CapturePreviewToFile('C:\Output\screenshot.png', 0);

// Capture as JPEG
sgcWebView21.CapturePreviewToFile('C:\Output\screenshot.jpg', 1);
```

```
procedure TFormMain.sgcWebView21CapturePreviewCompleted(
  Sender: TObject; aErrorCode: HRESULT);
begin
  if aErrorCode = S_OK then
    ShowMessage('Screenshot saved.')
  else
    ShowMessage('Screenshot failed: ' + IntToStr(aErrorCode));
end;
```

Audio / Mute Control

IsMuted 속성을 사용하여 오디오 재생을 음소거하거나 음소거 해제하십시오. 페이지가 현재 오디오를 재생 중인지 감지하려면 **IsDocumentPlayingAudio**를 확인하십시오.

```
// Toggle mute
sgcWebView21.IsMuted := not sgcWebView21.IsMuted;

// Check if audio is playing
if sgcWebView21.IsDocumentPlayingAudio then
  LabelStatus.Caption := 'Audio is playing';
```

인증서 처리

서버가 클라이언트 인증서를 요구할 때 응답하려면 **OnClientCertificateRequested**를 사용하십시오. TLS 인증서 오류(예: 개발 시 자체 서명된 인증서)를 처리하려면 **OnServerCertificateError**를 사용하십시오.

```
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;
```

가상 호스트 매핑

호스트 이름을 로컬 폴더에 매핑하려면 **SetVirtualHostNameToFolderMapping**을 사용하십시오. 이를 통해 웹 콘텐츠가 file:// 경로 대신 가상 URL을 사용하여 로컬 파일을 참조할 수 있습니다.

```
// 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');
```

Profile Management

현재 브라우저 프로필 이름을 검색하려면 **GetProfileName**을 사용하십시오. 캐시된 데이터, 쿠키 및 기타 브라우징 아티팩트를 제거하려면 **ClearBrowsingData** 또는 **ClearAllBrowsingData**를 사용하십시오.

```
// 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

OnBasicAuthRequested 이벤트는 서버가 HTTP Basic 인증을 요청할 때 발생합니다. 자격 증명을 제공하고 **aHandled**를 True로 설정하십시오.

```
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

사용자가 브라우저에서 마우스 오른쪽 버튼을 클릭할 때 **OnContextMenuRequested** 이벤트가 발생합니다. 기본 메뉴를 억제하고 자체 메뉴를 표시하려면 **aHandled**를 True로 설정하십시오.

```
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

FaviconURI 속성은 현재 페이지의 favicon URI를 반환합니다. **OnFaviconChanged** 이벤트는 favicon이 변경될 때 발생합니다.

```
procedure TFormMain.sgcWebView21FaviconChanged(Sender: TObject;
  const aFaviconURI: string);
begin
  LabelFavicon.Caption := 'Favicon: ' + aFaviconURI;
end;
```

Status Bar

StatusBarText 속성은 현재 상태 표시줄 텍스트(일반적으로 마우스를 올린 링크의 URL)를 반환합니다. **OnStatusBarTextChanged** 이벤트는 텍스트가 변경될 때 발생합니다.

```
procedure TFormMain.sgcWebView21StatusBarTextChanged(Sender: TObject;
  const aText: string);
begin
  StatusBar1.SimpleText := aText;
end;
```

Task Manager

각 브라우저 프로세스의 메모리 및 CPU 사용량을 보여주는 Edge 브라우저 작업 관리자를 열려면 **OpenTaskManagerWindow**를 사용하십시오.

```
sgcWebView21.OpenTaskManagerWindow;
```

공유 버퍼

고성능 데이터 전송을 위해 Delphi 애플리케이션과 웹 콘텐츠 간에 메모리 버퍼를 공유하려면 **PostSharedBufferToScript**를 사용하십시오.

```
// 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

구성 요소 API에서 다루지 않는 고급 시나리오의 경우, **WebView**, **Controller** 및 **Environment** 속성을 사용하여 기본 WebView2 COM 인터페이스에 직접 액세스하십시오.

```
var
  vWebView: ICoreWebView2;
begin
  vWebView := sgcWebView21.WebView;
  if Assigned(vWebView) then
  begin
    // Call any ICoreWebView2 method directly
  end;
end;
```

TsgcUDPClient

*TsgcUDPClient*는 Indy 라이브러리 기반의 UDP Client를 구현합니다.

소개

TsgcUDPClient는 Indy 라이브러리 기반의 UDP Client를 구현합니다.

UDP는 전송된 메시지가 대상에 도착한다는 보장이 없는 비연결형 프로토콜이지만, TCP 프로토콜과 반대로 훨씬 빠릅니다.

1. 폼에 **TsgcUDPClient** 구성 요소를 놓으십시오
2. 사용 가능한 UDP Server에 연결하려면 **Host**와 **Port**(기본값은 80)를 설정하십시오.

```
oClient = new TsgcUDPClient();
oClient->Host = "127.0.0.1";
oClient->Port = 80;
```

3. HTTP Proxy Server를 통해 연결할 수 있으며, 프록시 속성을 정의해야 합니다:

Host: 프록시 서버 호스트 이름입니다.

Port: 프록시 서버 포트 번호입니다.

Username: 인증을 위한 사용자 이름, 익명의 경우 비워 두십시오.

Password: 인증을 위한 비밀번호, 익명의 경우 비워 두십시오.

4. 원하는 경우 이벤트를 처리할 수 있습니다
5. UDP 서버에 메시지를 보내려면 **WriteData** 메서드를 호출하십시오.

참조

- 속성 — 구성 요소에서 사용 가능한 9개의 속성입니다.
- 메서드 — 2개의 public 메서드.
- 이벤트 — 처리할 수 있는 3개의 이벤트.

TsgcUDPServer

TsgcUDPServer는 Indy 라이브러리를 기반으로 하는 UDP 서버를 구현합니다.

소개

TsgcUDPServer는 Indy 라이브러리를 기반으로 하는 UDP 서버를 구현합니다.

UDP는 연결 없는 프로토콜로, 보낸 메시지가 목적지에 도착한다는 보장이 없지만 TCP 프로토콜과 반대로 훨씬 빠릅니다.

1. TsgcUDPServer 구성 요소를 품에 넣습니다

2. 수신 대기 Port를 설정합니다.

```
oClient = new TsgcUDPServer();
oClient->Port = 80;
```

3. 서버를 시작하려면 Active = true 속성을 설정하십시오.

참조

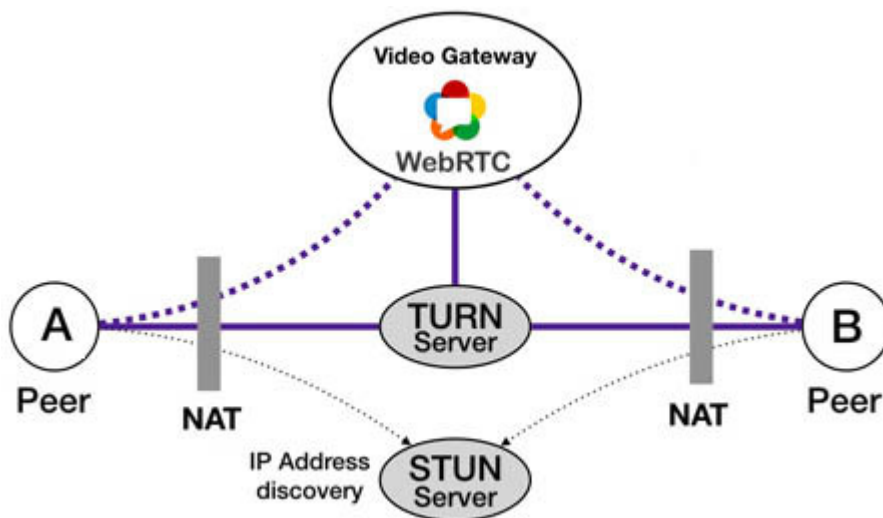
- 속성 — 구성 요소에서 사용할 수 있는 11개의 속성.
- 메서드 — 7개의 공개 메서드입니다.
- 이벤트 — 처리할 수 있는 6개의 이벤트.

STUN

STUN(Session Traversal Utilities for NAT)은 IP 네트워크에서 실시간 오디오 및 비디오에 사용되는 IETF 프로토콜입니다. STUN은 서버-클라이언트 프로토콜로, STUN 서버는 일반적으로 UDP와 TCP 모두에서 작동하며 포트 3478에서 수신합니다.

STUN 프로토콜의 주요 목적은 NAT 뒤에서 실행되는 디바이스가 자신의 공용 IP와 어떤 유형의 NAT 뒤에 있는지 발견할 수 있도록 하는 것입니다.

STUN은 NAT 뒤의 피어 간에 통신하는 메커니즘을 제공합니다. 피어는 STUN 서버에 요청을 보내 어느 것이 공개 IP 주소와 포트인지 알아냅니다. 클라이언트에서 서버로 보내는 바인딩 요청은 NAT가 할당한 IP 및 포트 바인딩을 결정하는 데 사용됩니다. STUN 클라이언트는 STUN 서버에 Binding 요청을 보내고, 서버는 클라이언트가 사용한 소스 IP 및 포트를 검사하여 이 정보를 클라이언트에 반환합니다.



STUN 서버는 기본적으로 두 가지 유형의 응답을 보냅니다: 성공 또는 오류. 모든 응답에는 바인딩 IP 주소, 오류 코드, 오류 이유 등에 대한 정보를 포함하는 속성 목록이 있습니다.

구성 요소

- **TsgcSTUNClient:** STUN 프로토콜을 구현하고 STUN 서버에 바인딩 요청을 보낼 수 있게 하는 클라이언트 구성 요소입니다.
- **TsgcSTUNServer:** STUN 프로토콜을 구현하는 서버 구성 요소입니다.

TsgcSTUNClient

TsgcSTUNClient 구성 요소.

소개

TsgcSTUNClient는 STUN 프로토콜을 구현하는 클라이언트이며 STUN 서버에 바인딩 요청을 보낼 수 있습니다.

구성 요소는 **UDP**와 **TCP**를 전송 수단으로 사용할 수 있게 합니다. UDP를 전송 수단으로 사용할 때, 짧은 시간 후에도 응답이 도착하지 않으면 요청을 다시 보내는 **재전송 메커니즘**을 구현합니다.

기본 사용법

일반적으로 STUN 서버는 UDP 포트 3478에서 실행되며 인증이 필요하지 않으므로, STUN 요청 바인딩을 보내려면 클라이언트가 연결할 위치를 알 수 있도록 서버 속성을 채우고 구성 요소가 서버로부터 응답을 받을 이벤트를 처리하십시오.

서버 구성

- Host: 서버의 IP 또는 DNS 이름, 예: stun.sgcwebsockets.com
- Port: 수신 Server 포트입니다(예: 3478).

STUN 서버에 binding 요청을 보내려면 **SendRequest** 메서드를 호출하십시오.

이벤트 처리

- 서버가 성공 응답을 반환하면 **OnSTUNResponseSuccess** 이벤트가 호출되고 **aBinding** 객체를 읽어 바인딩 정보에 액세스할 수 있습니다.
- 서버가 오류를 반환하면 **OnSTUNResponseError** 이벤트가 호출되며, **aError** 객체를 읽어 Error Code와 Reason에 액세스할 수 있습니다.

```
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);
}
```

참조

- 속성 — 구성 요소에서 사용할 수 있는 9개의 속성.
- 메서드 — 3개의 public 메서드.
- 이벤트 — 처리할 수 있는 4개의 이벤트.

Guides

- **Bindings**
 - [UDP Retransmissions](#)
 - [Long Term Credentials](#)
- - [Attributes](#)

STUN Client | UDP Retransmissions

STUN을 **UDP**를 통해 실행할 때 **STUN 메시지**가 네트워크에 의해 **손실**될 수 있습니다. STUN 요청/응답 트랜잭션의 신뢰성은 클라이언트 애플리케이션 자체에 의한 요청 메시지의 재전송을 통해 달성됩니다.

클라이언트는 RTO("Retransmission TimeOut") 간격으로 시작하여 STUN 요청 메시지를 재전송해야 하며, 각 재전송 후 두 배로 늘려야 합니다. RTO는 왕복 시간의 추정치입니다.

기본적으로 sgcWebSockets STUN Client는 RTO 500 ms와 Max Retries 값 7로 이미 구성되어 있습니다.

예를 들어, RTO가 500ms라고 가정하면, 요청은 0ms, 500ms, 1500ms, 3500ms, 7500ms, 15500ms 및 31500ms 시점에 전송됩니다. 클라이언트가 39500ms 후에도 응답을 받지 못하면, 클라이언트는 트랜잭션이 시간 초과된 것으로 간주합니다.

```
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

long-term credential 메커니즘은 클라이언트와 서버 간에 공유되는 사용자 이름과 비밀번호 형태의 long-term credential에 의존합니다. 이 credential은 사용자에게 대해 프로비저닝되며 사용자가 시스템의 구독자가 아니게 되거나 변경될 때까지 유효한 것으로 가정되므로 long-term으로 간주됩니다.

sgcWebSockets STUN 클라이언트에서 다음 코드를 사용하여 Long-term 자격 증명을 구성할 수 있습니다.

```
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();
```

서버가 장기 자격 증명을 요구하고 클라이언트가 보낸 자격 증명이 잘못된 경우, 클라이언트는 **OnSTUNResponseError** 이벤트에서 401 Unauthorized 오류를 응답으로 받습니다.

STUN Client | Attributes

서버가 클라이언트에 메시지를 보낼 때마다, binding 요청에 대한 응답 메시지로 STUN 메시지에는 응답에 대한 정보가 있는 attribute 목록이 포함됩니다.

TsgcSTUN_Message 클래스와 그 Attributes 속성을 사용하여 이러한 속성에 접근할 수 있으며, 이 속성은 유용한 정보를 담은 TsgcSTUN_Attribute 객체 목록을 포함합니다.

```
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

STUN 서버는 Authentication 사용 여부, Fingerprint Attribute 확인, 대체 서버 전송 등으로 구성할 수 있습니다.

소개

TsgcSTUNServer는 STUN 프로토콜을 구현하고 STUN 클라이언트로부터 binding 요청을 처리할 수 있는 서버입니다.

STUN 서버는 Authentication 사용 여부, Fingerprint Attribute 확인, 대체 서버 전송 등으로 구성할 수 있습니다.

기본 사용법

일반적으로 STUN 서버는 UDP 포트 3478에서 실행되며 인증을 요구하지 않으므로, STUN 서버를 구성하려면 수신 포트(기본값 3478)를 설정하고 서버를 시작하십시오.

서버 구성

- Port: 수신 Server 포트입니다(예: 3478).

STUN 서버를 시작하려면 속성 **Active = True**를 설정하십시오.

```
TsgcSTUNServer oSTUN = new TsgcSTUNServer();
oSTUN->Port = 3478;
oSTUN->Active = true;
```

참조

- Properties — 구성 요소에서 사용 가능한 8개의 속성.
- Methods — 2개의 public 메서드.
- 이벤트 — 처리할 수 있는 이벤트 4개입니다.

Guides

- Configurations
 - [장기 자격 증명](#)
 - [Alternate Server](#)

STUN Server | Long-Term Credentials

일반적으로 STUN 서버는 인증 없이 구성되므로 모든 STUN 클라이언트가 인증 없이 바인딩 요청을 보내고 서버로부터 응답을 받을 수 있습니다.

sgcWebSockets STUN Server는 Long-Term Credentials를 지원하므로, Long-Term 자격 증명 정보가 있는 바인딩 요청만 허용하도록 TsgcSTUNServer를 구성할 수 있습니다.

구성하려면 STUNOptions.Authorization 속성에 액세스하여 활성화하십시오.

그런 다음 LongTermCredentials 속성에 접근하여 활성화하십시오. 기본적으로 이 유형의 권한 부여는 Realm 문자열과 기본 StaleNonce 값 10분(= 600초)으로 이미 구성되어 있습니다.

```
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

alternate 서버는 STUN 클라이언트가 시도해야 하는 다른 STUN 서버를 식별하는 alternate 전송 주소를 나타냅니다.

STUN Server는 binding 요청에 대한 응답으로 대체 서버를 보내도록 구성할 수 있습니다. 이 동작을 구성하려면 STUNOptions.BindingAttributes.AlternateServer 속성에 액세스하여 필요한 값을 여기에 구성하십시오.

```
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;
```

클라이언트가 Alternate Server 응답 속성을 수신하면 새 서버에 binding 요청을 보내려고 시도합니다.

TURN

Traversal Using Relays around NAT(TURN) 프로토콜은 서버가 장치 간에 데이터 패킷을 중계할 수 있게 합니다.

발신자와 수신자 양쪽의 공인 IP 주소가 발견되지 않으면, TURN은 엔드포인트 간 통화를 중계하는 대체 기법을 제공합니다.

WebRTC 세션 연결은 여러 WebRTC 서버의 도움으로 수행되는 조율된 작업입니다. WebRTC의 NAT traversal 서버는 미디어가 제대로 연결되도록 하는 역할을 합니다. 이러한 서버는 STUN과 TURN입니다.

WebRTC 세션이 연결되는 방법

직접

두 디바이스가 모두 로컬 네트워크에 있으면 서로 연결하는 데 특별한 노력이 필요하지 않습니다. 한 디바이스가 다른 디바이스의 로컬 IP 주소를 가지고 있으면 직접 서로 통신할 수 있습니다.

공용 IP 주소로 직접

STUN 프로토콜을 통해 얻은 공개 IP 주소를 사용하여 WebRTC를 직접 연결합니다.

TURN Server를 통한 라우팅

피어가 NAT 뒤에 있고 방화벽이 있는 경우 직접 연결이 불가능하므로 피어 간 데이터를 라우팅하기 위해 TURN 서버가 필요합니다.

구성 요소

- **TsgcTURNClient:** TURN 프로토콜을 구현하고 TURN 서버에 할당, 권한 생성, indication 전송 등을 할 수 있게 하는 클라이언트 구성 요소.
- **TsgcTURNServer:** TURN 프로토콜을 구현하는 서버 구성 요소입니다.

TsgcTURNClient

TsgcTURNClient 구성 요소.

소개

TsgcTURNClient는 TURN 프로토콜을 구현하고 TURN 서버에 할당 요청을 보낼 수 있게 해주는 클라이언트입니다. 클라이언트는 STUN Client에서 상속되므로 **STUN client**가 지원하는 모든 메서드가 TURN Client에서 이미 지원됩니다.

기본 사용법

일반적으로 TURN 서버는 UDP 포트 3478에서 실행되며 인증이 필요하지 않으므로, TURN 요청을 보내려면 클라이언트가 어디에 연결할지 알 수 있도록 서버 속성을 채우고 구성 요소가 서버로부터 응답을 받을 이벤트를 처리하십시오.

서버 구성

- Host: 서버의 IP 또는 DNS 이름, 예: turn.sgcwebsockets.com
- Port: 수신 Server 포트입니다(예: 3478).

TURN 서버에 IP 주소와 포트 할당을 요청하기 위해 **Allocate** 메서드를 호출하십시오.

이벤트 처리

- 서버가 성공 응답을 반환하면 **OnTURNAllocateSuccess** 이벤트가 호출되며 **aAllocation** 객체를 읽어 Allocation 정보에 액세스할 수 있습니다.
- 서버가 오류를 반환하면 **OnSTUNResponseError** 이벤트가 호출되며, **aError** 객체를 읽어 Error Code와 Reason에 액세스할 수 있습니다.

```
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);
}
```

참조

- 속성 — 구성 요소에서 사용 가능한 10개의 속성입니다.
- 메서드 — 9개의 public 메서드.
- 이벤트 — 처리할 수 있는 11개의 이벤트.

Guides

- **Allocation**
 - [Allocate IP Address](#)
 - [Create Permissions](#)

- **Indications**
 - [Send Indication](#)
- **Channels**
 - [TURN Client Channels](#)

TURN Client | Allocate IP Address

TURN Protocol을 사용하면 NAT 뒤에 있는 피어 간에 데이터를 교환하기 위해 Relayed IP Address를 사용할 수 있습니다.

TURN 서버에 새 Relayed IP Address를 생성하려면 클라이언트는 먼저 **Allocate** 메서드를 호출해야 합니다. 이 메서드는 새 Relayed IP Address를 생성하기 위해 TURN 서버에 Request를 보내며, TURN 서버가 새 Relayed IP Address를 생성할 수 있으면 클라이언트는 성공적인 응답을 받습니다. 클라이언트는 Allocation의 lifetime에 정의된 시간 동안 다른 피어와 통신할 수 있습니다.

```
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);
}
```

수명은 만료를 피하기 위해 **Refresh** 메서드를 사용하여 업데이트할 수 있습니다. Lifetime은 만료까지의 초 수입니다. 값이 0이면 Allocation이 삭제됩니다.

```
oTURN->Refresh(600);
```

TURN Client | Create Permissions

TURN 서버에서 새 Allocation이 생성되면 권한이 없으면 다른 피어로부터의 수신 패킷을 처리할 수 없습니다. 따라서 다른 피어가 Relayed IP Address를 사용하여 통신할 수 있게 하려면, 먼저 TURN Client가 데이터를 교환할 수 있는 IP 주소에 대한 권한을 생성해야 합니다.

새 Permission을 생성하려면, **CreatePermission** 메서드를 호출하고 매개변수로 피어의 IP Address를 전달하기만 하면 됩니다. Peer IP가 TURN 서버에 이미 존재하면 새로 고쳐지고, 그렇지 않으면 생성됩니다. Permission은 새로 고쳐지지 않는 한 5분 후에 만료됩니다.

TURN 클라이언트는 활성 할당이 있는 경우에만 CreatePermission 메서드를 호출할 수 있습니다.

permission이 성공적으로 생성되면 **OnTURNCreatePermission** 이벤트가 호출됩니다.

```
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

TURN Protocol은 피어로부터 데이터를 보내고 받기 위한 2가지 메커니즘을 지원하며, 그 중 하나는 Send 및 Data 메커니즘입니다.

TURN 클라이언트는 **SendIndication** 메서드를 사용하여 서버에 데이터를 보내 피어로 중계할 수 있습니다. TURN 클라이언트는 Send Indication이 전송될 Peer IP 주소에 대한 permission이 있는지 확인해야 합니다.

SendIndication 메서드에 대한 응답은 **OnTURNDatIndication** 이벤트에서 수신됩니다.

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

void OnTURNDatIndication(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 클라이언트 | 채널

채널은 TURN 클라이언트와 서버가 **Send 및 Data** Indication보다 오버헤드가 적은 ChannelData 메시지를 사용하여 애플리케이션 데이터를 보내는 방법을 제공합니다.

ChannelData 메시지를 사용하여 피어 간에 데이터를 교환하기 전에, TURN 클라이언트는 새 채널을 생성해야 합니다. 이를 위해 Peer IP 주소와 포트를 매개변수로 전달하여 **ChannelBind** 메서드를 호출하기만 하면 됩니다.

TURN 서버가 새 채널을 바인딩할 수 있으면 TURN 클라이언트는 성공 응답 **OnTURNChannelBind** 이벤트를 수신합니다.

```
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));
}
```

채널 바인딩은 새로 고치지 않으면 10분 동안 지속됩니다. 채널을 새로 고치려면 **ChannelBind** 메서드를 다시 호출하기만 하면 됩니다.

TURN 클라이언트가 새 ChannelMessage를 수신하면, **OnTURNChannelData** 이벤트가 호출됩니다.

```
void OnTURNChannelData(TObject *Sender, const TsgcSocketConnection *aSocket,
    const TsgcTURNChannelData *aChannelData)
{
    DoLog("#Channel Data: [" + IntToStr(aChannelData->ChannelID) + "] " +
        sgcGetStringFromBytes(aChannelData->Data));
}
```

TsgcTURNServer

TURN Server는 Long-Term Authentication, Allocation, Permissions, Channel Data 등을 지원합니다.

소개

TsgcTURNServer는 TURN 프로토콜을 구현하는 서버이며 TURN 클라이언트의 요청을 처리할 수 있게 합니다. 구성 요소는 TsgcSTUNServer를 상속하므로, 모든 메서드와 속성을 TsgcTURNServer에서 사용할 수 있습니다.

TURN Server는 Long-Term Authentication, Allocation, Permissions, Channel Data 등을 지원합니다.

기본 사용법

일반적으로 TURN 서버는 UDP 포트 3478에서 실행되며 Long-Term 자격 증명을 요구하므로, TURN 서버를 구성하려면 수신 대기 포트(기본값 3478)를 설정하고 서버를 시작하십시오.

서버 구성

- Port: 수신 Server 포트입니다(예: 3478).
- TURNOptions.Authentication.LongTermCredentials에서 Long-Term Credentials 속성을 정의하십시오
- TURN 클라이언트가 TURN 서버에 요청을 보낼 때 비밀번호를 설정하려면 OnSTUNRequestAuthorization을 처리하십시오.

TURN 서버를 시작하려면 **Active = True** 속성을 설정하십시오.

```
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";
    }
}
```

참조

- 속성 — 구성 요소에서 사용 가능한 9개의 속성.
- 메서드 — 2개의 공용 메서드.
- 이벤트 — 처리할 수 있는 11개의 이벤트.

Guides

- Configurations
 - 장기 자격 증명
- Allocations
 - Allocations

TURN Server | Long Term Credentials

일반적으로 TURN 서버는 TURN 요청에 대해서는 Authentication을 사용하고 STUN 요청에 대해서는 Authentication 없이 구성됩니다.

sgcWebSockets TURN Server는 Long-Term Credentials를 지원하므로, Long-Term 자격 증명 정보가 있는 요청만 허용하도록 TsgcTURNServer를 구성할 수 있습니다.

이를 구성하려면, TURNOptions.Authorization 속성에 액세스하여 활성화하십시오.

그런 다음 LongTermCredentials 속성에 접근하여 활성화하십시오. 기본적으로 이 유형의 권한 부여는 Realm 문자열과 기본 StaleNonce 값 10분(= 600초)으로 이미 구성되어 있습니다.

```
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

모든 TURN 작업은 할당을 중심으로 이루어지며 모든 TURN 메시지는 Allocation과 연결됩니다. 할당은 다음으로 구성됩니다:

- 릴레이된 전송 주소
- 5-Tuple: 클라이언트의 IP 주소, 클라이언트의 IP 포트, 서버 IP 주소, 서버 포트, 전송 프로토콜입니다.
- 인증 정보입니다.
- 각 중계 전송 주소에 대한 만료 시간입니다.
- 각 릴레이된 전송 주소에 대한 권한 목록입니다.
- 각 릴레이된 전송 주소에 대한 채널 바인딩 목록입니다.

TURN 클라이언트가 Allocate 요청을 보내면, 이 TURN 메시지는 서버에 의해 처리되어 새 Relayed Transport Address를 생성하려고 시도합니다. 기본적으로 사용 가능한 UDP 포트가 있으면 새 Relayed Address를 생성하지만, **OnTURNBeforeAllocate** 이벤트를 사용하여 새 Allocation 요청을 거부할 수 있습니다.

```
void OnTURNBeforeAllocate(TObject *Sender, const TsgcSocketConnection *aSocket,
    const string aIP, Word aPort, ref bool Reject)
{
    if (your own rules) == false
    {
        Reject = false;
    }
}
```

프로세스가 계속되면 서버는 새 allocation을 생성하고 **OnTURNCreateAllocation** 이벤트가 호출됩니다. 이 이벤트는 TsgcTURNAllocationItem 클래스를 통해 Allocation에 대한 정보를 제공합니다.

```
void OnTURNCreateAllocation(TObject *Sender, const TsgcSocketConnection *aSocket,
    const TsgcTURNAllocationItem *Allocation)
{
    DoLog("New Allocation: " + Allocation->RelayIP + ":" + IntToStr(Allocation->RelayPort));
}
```

Allocation이 만료되거나 lifetime이 0인 Refresh Request를 클라이언트로부터 수신하여 삭제될 때 **OnTURNDeleteAllocation** 이벤트가 발생합니다.

```
void OnTURNDeleteAllocation(TObject *Sender, const TsgcSocketConnection *aSocket,
    const TsgcTURNAllocationItem *Allocation)
{
    DoLog("Allocation Deleted: " + Allocation->RelayIP + ":" + IntToStr(Allocation->RelayPort));
}
```

ICE

Interactive Connectivity Establishment(ICE) 프로토콜은 NAT 통과에 사용됩니다. ICE는 Session Traversal Utility for NAT(STUN)과 Traversal Using Relay NAT(TURN)을 포함한 방법의 조합을 사용합니다. Network Address Translator(NAT)의 존재는 Voice over IP(VoIP) 및 WebRTC 구현에 문제를 일으킵니다.

구성 요소

- **TsgcICEClient:** ICE 프로토콜을 구현하고 candidate를 얻고, 교환하고, 검증할 수 있는 클라이언트 구성 요소입니다.

TsgcICEClient

ICEOptions.

소개

TsgcICEClient는 ICE 프로토콜을 구현하고 TURN 서버에 할당 요청을 보낼 수 있게 하는 클라이언트입니다. 클라이언트에는 **TsgcTURNClient**와 **TsgcWebSocketClient**가 필요합니다.

구성

ICE 클라이언트에는 다음 속성이 있습니다

- **ICEOptions.CheckList.MaxCandidates**: 클라이언트가 처리할 수 있는 최대 후보 수이며, 기본값은 100입니다.
- **ICEOptions.GatherCandidates.STUN**: 기본적으로 true이며, STUN 프로토콜(reflexive 주소)을 사용하여 후보를 가져옵니다.
- **ICEOptions.GatherCandidates.TURN**: 기본적으로 true이며, TURN 프로토콜을 사용하여 후보(릴레이된 주소)를 얻습니다.

ICE 클라이언트는 후보를 수집하기 위해 **TURN 클라이언트**가 필요하므로, **GatherCandidates** 메서드를 호출하기 전에 **TsgcTURNClient**를 **TsgcICEClient**의 TURN 속성에 연결하십시오. **TURN 클라이언트**를 구성하는 방법에 대한 자세한 정보를 얻을 수 있습니다.

```
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();
```

참조

- 속성 — 구성 요소에서 사용 가능한 3개의 속성.
- 메서드 — 3개의 public 메서드.
- 이벤트 — 처리할 수 있는 6개의 이벤트입니다.

Guides

- **Candidates**
 - [Gather Candidates](#)
 - [Pair Candidates](#)

ICE | Gather Candidates

ICE가 후보 수집을 시작하며, 일반적으로 로컬 IP Addresses, STUN 프로토콜을 사용한 reflexive 주소 및 TURN 프로토콜을 사용한 relayed 주소를 얻습니다.

수집을 시작하려면 **GatherCandidates** 메서드를 호출하십시오. 이렇게 하면 먼저 로컬 IP 주소를 얻고, 그런 다음 STUN 서버에 연결하여 reflexive IP 주소를 얻고, 마지막으로 TURN 서버에 연결하여 relayed IP 주소를 얻는 내부 타이머가 시작됩니다.

새 후보가 획득될 때마다 **OnICECandidate** 이벤트가 비동기적으로 호출됩니다. 후보를 수집하는 동안 오류가 발생하면 **OnICECandidateError** 이벤트가 트리거됩니다.

```
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

후보(로컬 및 원격)를 얻고 SDP 설명이 설정되면, ICE caller 클라이언트는 데이터를 교환할 수 있는 후보를 찾기 위해 모든 쌍 후보를 처리하기 시작할 수 있습니다. 이 프로세스를 시작하려면 **ProcessCandidates** 메서드를 호출하십시오.

ProcessCandidates 메서드는 STUN 바인딩 패킷을 보내 모든 쌍 후보를 평가합니다. STUN 바인딩 패킷이 다른 피어로부터 응답으로 수신되면 두 피어 간의 연결이 가능하다는 의미이므로 해당 쌍이 지명됩니다.

페어링이 성공하면 **OnICECandidatePairNominated** 이벤트가 비동기적으로 트리거됩니다. 페어링에 오류가 있거나 시간 초과 후 연결할 수 없는 경우 **OnICECandidatePairFailed** 이벤트가 트리거됩니다.

TsgcRTCPeerConnection

*TsgcRTCPeerConnection*은 UDP를 통한 P2P로 피어를 연결할 수 있는 클라이언트 구성 요소입니다.

소개

[*현재 이 구성 요소는 개발 중입니다]

TsgcRTCPeerConnection은 UDP를 통한 P2P로 피어를 연결할 수 있는 클라이언트 구성 요소입니다. 흐름은 4단계로 나눌 수 있습니다:

- 시그널링
- 연결 중
- 보안 적용
- Communicating

이러한 단계를 구현하기 위해 클라이언트는 다음 프로토콜을 사용합니다:

- **WebSocket:** 이 프로토콜은 시그널링에 사용되며, 클라이언트가 Session Description Protocol과 로컬, 공개 및 Relayed IP 주소를 교환합니다.
- **UDP:** 이것은 전송 프로토콜이며, 클라이언트는 UDP를 사용하여 피어 간에 메시지를 보내고 받습니다.
- **DTLS:** TLS와 유사하게, 피어 간의 메시지를 보호하여 제3자가 메시지를 읽거나 쓰는 것을 방지하는 암호화 사양입니다.
- **STUN:** 공개 IP 주소를 얻기 위한 프로토콜입니다.
- **TURN:** 피어가 NAT 뒤에 있을 때 IP 주소를 중계하는 프로토콜입니다.
- **ICE:** 피어 간에 어떤 IP 주소와 포트에 액세스할 수 있는지 찾는 프로토콜입니다.

시그널링

클라이언트가 시작될 때 누구와 통신할지 또는 무엇에 대해 통신할지 전혀 알지 못합니다. Signaling은 다음과 같은 세부 정보를 포함하는 SDP(Session Description Protocol)를 사용합니다:

- 피어에 도달할 수 있는 IP와 Port.
- 통신을 보호하는 데 사용되는 Fingerprint의 인증서입니다.
- User and Password.
- ...

Signaling은 데이터를 교환하기 위해 WebSocket 프로토콜을 사용합니다. 이는 서브프로토콜을 통해 작동하며 서버 측의 TsgcWSPServer_RTCPeerConnection 구성 요소에 구현됩니다.

TsgcRTCPeerConnection은 이미 내부적으로 TsgcWSPClient_RTCPeerConnection이 첨부된 websocket 클라이언트를 생성합니다.

IP와 Port를 얻기 위해 클라이언트는 STUN/TURN 프로토콜을 사용하여 이 정보를 얻습니다. 따라서 STUN/TURN 서버도 필요합니다.

Links:

- [RTCPeerConnection WebSocket Server](#)
- [RTCPeerConnection WebSocket Client](#)
- [RTCPeerConnection STUN TURN](#)
- [RTCPeerConnection Signaling](#)

연결 중

2개의 피어가 후보와 SDP를 알게 되면, 클라이언트는 ICE라는 또 다른 표준 프로토콜을 사용합니다.

ICE(Interactive Connection Establishment)는 두 피어 간의 연결 설정을 허용합니다. 피어는 동일한 네트워크에 있거나 NAT 뒤에 있을 수 있습니다... ICE는 중앙 서버 없이 직접 연결을 설정하기 위한 솔루션입니다. 연결이 P2P가 될 수 없는 경우 ICE는 TURN 서버를 사용하여 데이터를 릴레이하기 위해 TURN을 사용합니다.

ICE가 두 피어 간에 연결할 수 있는 유효한 후보를 찾으면, 다음 단계는 통신을 암호화하는 것입니다.

Links:

- [RTCPeerConnection ICE](#)

보안 적용

피어가 연결된 후 통신은 안전해야 합니다. 이는 UDP를 통한 통신을 보호하는 데 사용되는 암호화 프로토콜인 DTLS를 사용하여 수행됩니다.

DTLS 핸드셰이크가 성공적으로 처리되면 다른 프로토콜인 SRTP(Secure Real-Time Transport Protocol)가 사용됩니다. 현재 SRTP는 구현되어 있지 않습니다.

Links:

- [RTCPeerConnection DTLS](#)

Communicating

2개의 피어가 보안 프로토콜을 사용하면, 통신은 2개의 프로토콜을 사용하여 수행됩니다:

- **RTP**: Real Time Transport Protocol: SRTP로 암호화된 미디어를 교환하는 데 사용됩니다.
- **SCTP**: Stream Control Transmission Protocol으로, dTLS로 암호화된 DataChannel 메시지를 송수신하는 데 사용됩니다.

현재 이 프로토콜들은 구현되지 않았지만, UDP를 통한 DTLS를 사용하여 데이터를 보내고 받을 수 있습니다.

Links:

- [RTCPeerConnection Data](#)

참조

- 속성 — 구성 요소에서 사용 가능한 2개의 속성.
- 메서드 — 3개의 public 메서드.
- 이벤트 — 처리할 수 있는 14개의 이벤트.

RTCPeerConnection | WebSocket Server

TsgcRTCPeerConnection 클라이언트는 시그널링을 위해 WebSocket 서버가 필요합니다. 클라이언트는 WebSocket 프로토콜을 사용하여 피어의 SDP와 후보(IP 및 포트)를 교환하며, 이를 통해 피어가 통신할 수 있습니다.

을 구성하려면 **WebSocket 서버** sgcWebSockets 라이브러리에서 사용 가능한 WebSocket 서버 중 아무거나 사용하고 **TsgcWSPServer_RTCPeerConnection** 을 연결할 수 있으며, 이는 RTCPeerConnection이 사용하는 서브프로토콜입니다.

```
TsgcWebSocketServer *oServer = new TsgcWebSocketServer();
TsgcWSPServer_RTCPeerConnection *oProtocol = new TsgcWSPServer_RTCPeerConnection();
oProtocol->Server = oServer;
oServer->Port = 8080;
oServer->Active = true;
```

새 websocket 클라이언트가 websocket 서버에 연결될 때마다, 서버는 동일한 채널에서 수신 대기 중인 다른 피어가 있는지 확인하고 그에 따라 데이터를 전달합니다.

RTCPeerConnection | WebSocket Client

RTCPeerConnection은 WebSocket 서버와 통신하기 위해 사용자 지정 서브프로토콜을 사용하는 WebSocket 클라이언트를 내부적으로 생성합니다. RTCOptions.WebSocket 속성에서 WebSocket 연결을 정의하는 값을 찾을 수 있습니다

- **Host:** 서버의 dns 또는 ip 주소, 예: 127.0.0.1 또는 www.esegece.com.
- **Port:** websocket 서버의 수신 대기 포트입니다.
- **TLS:** 서버가 보안 연결을 사용하는 경우 활성화하십시오.
- **Channel:** 피어 간에 데이터를 교환하는 데 사용되는 채널 이름입니다(두 피어 모두 동일한 채널 이름을 가져야 하며 최대 피어 수는 2입니다).

RTCPeerConnection | STUN TURN

TsgcRTCPeerConnection은 STUN/TURN 프로토콜을 사용하여 공개 IP 주소와 릴레이된 IP 주소(필요한 경우)를 얻습니다. 따라서 이 정보를 얻으려면 STUN/TURN 서버가 필요합니다. STUN/TURN 서버에 대한 자세한 내용은 다음 링크에서 읽을 수 있습니다: [TURN Server](#).

STUN/TURN 서버가 실행되면 TsgcRTCPeerConnection의 RTCOptions.ICE 속성에서 TURN 서버 속성을 구성할 수 있습니다.

- **Host:** TURN 서버의 ip 주소 또는 dns. 예: 127.0.0.1 또는 www.esegece.com.
- **Port:** 일반적으로 기본 포트 3478입니다.
- **Username:** TURN 서버가 Long-term credentials(기본값)를 사용하는 경우의 username입니다.
- **Password:** TURN 서버가 Long-term 자격 증명(기본값)을 사용하는 경우의 비밀번호입니다.

RTCPeerConnection | Signaling

TsgcRTCPeerConnection이 RTCOptions 속성을 구성하고 서버(WebSocket 및 STUN/TURN)가 실행되면, 클라이언트는 candidate 수집 프로세스를 시작할 수 있습니다.

클라이언트는 먼저 websocket 서버에 연결하고, 연결이 성공하면 로컬 SDP를 보냅니다. 그런 다음 로컬 및 공개 IP 주소를 얻으려고 시도하며, 공개 IP 주소를 얻기 위해 STUN 서버에 binding 요청을 보내 공개 IP 주소와 TURN 서버의 릴레이된 IP 주소를 얻습니다. 새 후보를 얻을 때마다 이 정보가 websocket 서버로 전달되고, 이는 다른 피어로 전달됩니다.

RTCPeerConnection이 Local SDP, Remote SDP 및 후보를 갖게 되면 모든 후보 쌍을 확인하여 서로 연결할 수 있는지 확인하는 프로세스를 시작합니다. 후보 쌍이 성공적으로 연결되면 유효한 후보 쌍이며 보안 프로세스가 흐름을 계속합니다.

RTCPeerConnection | ICE

ICE(Interactive Connectivity Establishment)는 2개의 피어를 연결하는 데 사용되는 프로토콜로, 2개의 피어 간의 모든 가능한 경로를 결정한 다음 연결되도록 보장합니다. 이러한 경로는 로컬 및 원격 transport 주소의 페어링인 Candidate Pair라고도 합니다. 이러한 주소는 로컬 IP 주소, 공개 IP 주소 또는 Relayed Transport 주소일 수 있습니다. 각 피어는 사용하려는 모든 주소를 수집하고, 교환한 다음, 연결을 시도합니다.

주소 수집

Addresses를 수집할 때 다음 이벤트가 호출될 수 있습니다.

OnRTCLocalCandidate

이 이벤트는 새 로컬 후보가 발견되었을 때 호출됩니다.

OnRTCRemoteCandidate

이 이벤트는 websocket 서버가 이 피어에게 원격 candidate를 보낼 때 호출됩니다.

OnRTCLocalDescription

이벤트는 TsgcRTCPeerConnection이 로컬 SDP를 필요로 할 때 호출됩니다

OnRTCRemoteDescription

이 이벤트는 websocket 서버가 원격 SDP를 이 피어에 보낼 때 호출됩니다.

Connectivity Testing

피어가 연결할 수 있는지 테스트하기 위해 다른 피어에게 바인딩 요청을 보낼 때 다음 이벤트가 호출될 수 있습니다.

OnRTCCandidatePairNominated

두 피어가 모두 이 candidate 페어를 사용하여 연결할 수 있을 때 이벤트가 호출됩니다.

OnRTCCandidatePairFailed

피어가 이 후보 쌍을 사용하여 연결할 수 없을 때 이 이벤트가 호출됩니다.

OnRTCConnect

이 이벤트는 유효한 후보 쌍이 있고 DTLS가 활성화되지 않은 경우에 호출됩니다. DTLS가 활성화된 경우 이 이벤트는 성공적인 DTLS 핸드셰이크 후에 호출됩니다.

RTCPeerConnection | DTLS

유효한 후보 쌍(두 피어가 연결하고 서로 데이터를 교환할 수 있음)이 생기면, 연결을 보안하게 만들 때입니다. DTLS는 교환된 데이터의 콘텐츠 검사 또는 수정을 방지하기 위해 데이터를 암호화하는 암호화 프로토콜입니다.

DTLS에는 **openssl 라이브러리(openssl 1.1+)**가 필요합니다

DTLS 구성은 TsgcRTCPeerConnection의 RTCOptions.DTLSOptions 속성에서 찾을 수 있습니다. DTLS를 활성화하려면 RTCOptions.DTLS 속성을 True로 설정하십시오. 아래에서 주요 속성을 확인하십시오:

RootCertFile: 루트 인증서 파일의 경로입니다.

CertFile: 인증서 파일 경로입니다.

KeyFile: 인증서 키 파일의 경로입니다.

VerifyCertificate: 인증서를 검증해야 하는 경우 이 속성을 활성화하십시오.

VerifyDepth: 검증 시 허용되는 최대 링크 수를 나타내는 Integer 속성입니다

OpenSSL_Options: openssl 라이브러리의 구성입니다.

APIVersion: 사용할 OpenSSL API를 정의할 수 있습니다. openssl API 1.1+ 만 DTLS를 지원합니다.

oslAPI_1_1: API 1.1 OpenSSL을 사용하며, 당사의 사용자 정의 Indy 라이브러리가 필요하고 OpenSSL 1.1.1 라이브러리(TLS 1.3 지원 포함)를 사용할 수 있게 합니다.

oslAPI_3_0: API 3.0 OpenSSL을 사용하며, 사용자 지정 Indy 라이브러리가 필요하고 OpenSSL 3.0.0 라이브러리(TLS 1.3 지원 포함)를 사용할 수 있습니다.

LibPath: 여기에서 openssl 라이브러리가 위치한 곳을 구성할 수 있습니다.

oslpNone: 이것이 기본값이며, openssl 라이브러리는 바이너리가 있는 동일한 폴더 또는 알려진 경로에 있어야 합니다.

oslpDefaultFolder: 모든 IDE 퍼스널리티에 대해 라이브러리가 위치해야 하는 openssl 경로를 자동으로 설정합니다.

oslpCustomFolder: 이 옵션이 선택된 경우, LibPathCustom 속성에 전체 경로를 정의하십시오.

LibPathCustom: LibPath = oslpCustomFolder일 때 openssl 라이브러리가 있는 전체 경로를 여기에 정의하십시오.

UnixSymLinks: Unix 시스템에서 SymLinks 로딩을 활성화하거나 비활성화합니다(기본적으로 활성화되어 있으며, OSX64에서는 예외):

oslsSymLinksDefault: 기본적으로 활성화되지만 OSX64에서는 예외입니다(MacOS Monterey 이후 버전 없이 라이브러리를 로드하려고 하면 실패함).

oslsSymLinksLoadFirst: 버전 라이브러리를 로드하기 전에 SymLink를 로드하고 먼저 시도합니다.

oslsSymLinksLoad: 버전 라이브러리를 로드하려고 시도한 후 SymLinks를 로드합니다.

oslsSymLinksDontLoad: SymLinks를 로드하지 않습니다.

RTCPeerConnection | Data

연결이 성공하면 **WriteData** 메서드를 사용하여 다른 피어에 메시지를 보낼 수 있습니다. 연결은 UDP를 통한 DTLS를 사용하며, 가능한 경우 P2P 연결이 됩니다(P2P가 불가능한 경우 릴레이된 주소 사용).

Send Data

다른 피어에 데이터를 보내려면 WriteData 메서드를 사용하십시오. 문자열 또는 바이트 배열을 보낼 수 있습니다.

```
TsgcRTCPeerConnection *oRTCPeerConnection = new TsgcRTCPeerConnection();  
...  
oRTCPeerConnection->WriteData("Hello from sgcWebSockets!!!");
```

데이터 수신

TsgcRTCPeerConnection이 다른 피어로부터 데이터를 수신할 때마다 OnRTCMessage 이벤트가 호출됩니다.

```
void OnRTCMessage(TObject *Sender, const TBytes aBytes)  
{  
    ShowMessage(TEncoding->UTF8->GetString(aBytes));  
}
```

Datasnap

기본적으로 DataSnap은 HTTP 요청을 처리하기 위해 **TIdHttpWebBrokerBridge**를 서버로 사용합니다. 그러나 더 많은 프로토콜을 지원하고 훨씬 더 나은 성능을 제공하는 다른 서버로 이 서버를 교체하여 DataSnap 애플리케이션의 가능성을 확장할 수 있습니다. 서버 성능을 개선하는 HTTP/2, 훨씬 더 많은 연결을 처리할 수 있게 하는 IOCP 등 최신 프로토콜을 활용할 수 있습니다.

서버

서버	주요 기능	Description
TsgcWSHTTPWebBrokerBridgeServer	WebSocket Protocol HTTP 1.* Protocol XHR Protocol IOCP	Indy 라이브러리 기반으로, 동일한 포트에서 WebSocket과 HTTP 프로토콜을 지원합니다. IOCP도 활성화할 수 있습니다.
TsgcWSHTTP2WebBrokerBridgeServer	WebSocket Protocol HTTP 1.* Protocol HTTP/2 Protocol XHR Protocol IOCP	Indy 라이브러리를 기반으로 하며 동일한 포트에서 WebSocket 및 HTTP/2 프로토콜을 지원합니다. IOCP도 활성화할 수 있습니다.
TsgcWSServer_HTTPAPI_WebBrokerBridge	WebSocket Protocol HTTP 1.* Protocol HTTP/2 Protocol XHR Protocol IOCP	HTTP.SYS Microsoft HTTP API를 기반으로 하며, 동일한 포트에서 WebSocket 및 HTTP/2 프로토콜을 지원합니다. 기본적으로 IOCP가 사용됩니다. 최상의 성능을 위해 권장됩니다.

TsgcWSHTTPWebBrokerBridgeServer

TsgcWSHTTPWebBrokerBridgeServer는 서버 기반으로 **TIdHttpWebBrokerBridge**를 사용하며, DataSnap, HTTP 및 WebSocket 연결에 단일 서버를 사용하려는 경우 유용합니다.

TsgcWSHTTPWebBrokerBridgeServer는 **TsgcWebSocketHTTPServer**에서 상속되므로 이 서버를 참조할 수 있습니다.

TIdHttpWebBrokerBridge를 TsgcWSHTTPWebBrokerBridgeServer로 교체하려면 다음 단계를 따르십시오:

1. TsgcWSHTTPWebBrokerBridgeServer의 새 인스턴스를 생성합니다.
2. TIdHttpWebBrokerBridge에 대한 모든 호출을 TsgcWSHTTPWebBrokerBridgeServer로 교체하십시오.
3. WebSocket 연결을 처리하려면 [TsgcWebSocketHTTPServer](#)를 참조하십시오.

구성

Datasnap 구성 요소는 **Source** 폴더에만 위치합니다. 이 객체들은 sgcWebSockets 패키지에 포함되지 않으므로 컴파일된 폴더에서는 찾을 수 없으며, 런타임에 생성해야 합니다.

필요한 파일을 프로젝트에 추가하거나 sgcWebSockets 패키지의 Source 폴더로 경로를 설정하기만 하면 됩니다. 필요한 파일:

- sgcWebSocket_Server_WebBrokerBridge

프로젝트가 IdHTTPWebBrokerBridge를 사용하는 경우 sgclIdHTTPWebBrokerBridge로 변경하십시오(이는 Enterprise Edition에만 적용됩니다).

Events

```
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";
    }
}
```

로드 밸런서

서버가 [TsgcWebSocketLoadBalancerServer](#) 뒤에 있는 경우 CORS와 관련된 문제가 발생할 수 있습니다. 이러한 문제를 방지하려면 다음 코드를 사용하십시오

```
void __fastcall TWebModule1::WebModuleBeforeDispatch(TObject *Sender, TWebRequest *Request, TWebResponse *Response)
{
    // CORS 관련 코드
}
```

```
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"));
    Handled = true;
}
if (FServerFunctionInvokerAction != nullptr)
{
    FServerFunctionInvokerAction->Enabled = AllowServerFunctionInvoker;
}
}
```

TsgcWSHTTP2WebBrokerBridgeServer

TsgcWSHTTP2WebBrokerBridgeServer는 HTTP/2 프로토콜이 활성화된 **TsgcWebSocketHTTPServer**를 서버 기반으로 사용하며, DataSnap, HTTP/2 및 WebSocket 연결에 단일 서버를 사용하려는 경우 유용합니다.

TsgcWSHTTP2WebBrokerBridgeServer는 **TsgcWebSocketHTTPServer**에서 상속되므로, 이 서버를 참조할 수 있습니다.

TIdHttpWebBrokerBridge를 TsgcWSHTTP2WebBrokerBridgeServer로 교체하려면 다음 단계를 따르십시오:

1. TsgcWSHTTP2WebBrokerBridgeServer의 새 인스턴스를 생성하십시오.
2. TIdHttpWebBrokerBridge에 대한 모든 호출을 TsgcWSHTTP2WebBrokerBridgeServer로 교체하십시오.
3. WebSocket 연결을 처리하려면 [TsgcWebSocketHTTPServer](#)를 참조하십시오.

구성

Datasnap 구성 요소는 **Source** 폴더에만 위치합니다. 이 객체들은 sgcWebSockets 패키지에 포함되지 않으므로 컴파일된 폴더에서는 찾을 수 없으며, 런타임에 생성해야 합니다.

필요한 파일을 프로젝트에 추가하거나 sgcWebSockets 패키지의 Source 폴더로 경로를 설정하기만 하면 됩니다. 필요한 파일:

- sgcWebSocket_Server_Base_WebBrokerBridge
- sgcWebSocket_Server_WebBrokerBridge_HTTP2

프로젝트가 IdHTTPWebBrokerBridge를 사용하는 경우 sgIdHTTPWebBrokerBridge로 변경하십시오(이는 Enterprise Edition에만 적용됩니다).

Events

```
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_WebBrokerBridge는 HTTP/2 프로토콜이 활성화된 **TsgcWebSocketServer_HTTPAPI**를 서버 베이스로 사용하며, DataSnap, HTTP/2, WebSocket 연결에 단일 서버를 사용하려는 경우 유용합니다.

TsgcWSServer_HTTPAPI_WebBrokerBridge는 **TsgcWebSocketServer_HTTPAPI**에서 상속되므로 이 서버를 참조할 수 있습니다.

TIdHttpWebBrokerBridge를 TsgcWSServer_HTTPAPI_WebBrokerBridge로 교체하려면 다음 단계를 따르십시오:

1. TsgcWSServer_HTTPAPI_WebBrokerBridge의 새 인스턴스를 생성합니다.
2. TIdHttpWebBrokerBridge에 대한 모든 호출을 TsgcWSServer_HTTPAPI_WebBrokerBridge로 교체하십시오.
3. WebSocket 연결을 처리하려면 [TsgcWSServer_HTTPAPI_WebBrokerBridge](#)를 참조하십시오.

구성

Datasnap 구성 요소는 **Source** 폴더에만 위치합니다. 이 객체들은 sgcWebSockets 패키지에 포함되지 않으므로 컴파일된 폴더에서는 찾을 수 없으며, 런타임에 생성해야 합니다.

필요한 파일을 프로젝트에 추가하거나 sgcWebSockets 패키지의 Source 폴더로 경로를 설정하기만 하면 됩니다. 필요한 파일:

- sgcWebSocket_Server_Base_WebBrokerBridge
- sgcWebSocket_Server_HTTPAPI_WebBrokerBridge

프로젝트가 IdHTTPWebBrokerBridge를 사용하는 경우 sgIdHTTPWebBrokerBridge로 변경하십시오(이는 Enterprise Edition에만 적용됩니다).

Events

```
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

이전에 **Swagger Specification**으로 알려졌던 **OpenAPI Specification**은 RESTful 웹 서비스를 설명, 생성, 소비, 시각화 하기 위한 기계 판독 가능 인터페이스 파일에 대한 사양입니다. 이전에는 Swagger 프레임워크의 일부였으나 2016년에 Linux Foundation의 오픈 소스 협업 프로젝트인 OpenAPI Initiative가 감독하는 별도의 프로젝트가 되었습니다. Swagger 및 기타 일부 도구는 인터페이스 파일이 주어지면 코드, 문서, 테스트 케이스를 생성할 수 있습니다.

OpenAPI 인터페이스 파일을 기반으로 구현된 애플리케이션은 메서드, 매개변수 및 모델의 문서를 자동으로 생성할 수 있습니다. 이는 문서, 클라이언트 라이브러리 및 소스 코드를 동기화 상태로 유지하는 데 도움이 됩니다.

Pascal Parser

sgcOpenAPI Generator 는 OpenAPI Spec이 주어지면 API 클라이언트 라이브러리(SDK 생성)를 자동으로 생성할 수 있습니다. 다음 OpenAPI 사양이 지원됩니다:

- **OpenAPI 3.***
- **Swagger 2.*** (2.0에서 3.0으로 자동 변환됨)
- **Swagger 1.*** (1.0에서 3.0으로 자동 변환됨)

sgcOpenAPI를 사용하면 JSON/YAML OpenAPI 또는 Swagger가 주어졌을 때 **Native Pascal Language** 로 자동으로 클라이언트 API 인터페이스를 생성할 수 있습니다. 현재 Delphi 7부터 최신 Delphi 버전까지 지원합니다.

sgcOpenAPI Generator 를 사용하면 OpenAPI / Swagger 사양에서 문서 파일을 생성할 수 있습니다.

[OpenAPI Parser Pascal](#)에 대해 자세히 알아보십시오.

OpenAPI Client

생성된 Client Interface에는 OpenAPI 사양에 정의된 모든 함수/메서드가 포함됩니다. 상수와 열거형도 생성됩니다.

다음 인증 방법이 지원됩니다:

- Basic Authentication
- [OAuth2 Code](#) (대화형)
- [OAuth2 Credentials](#) (비대화형)
- JWT

[OpenAPI Client](#)에 대해 자세히 알아보십시오.

OpenAPI 서버

내장 OpenAPI 서버를 사용하여 Delphi에서 OpenAPI 3.0 서비스를 호스팅할 수도 있습니다. 서버는 사양에 대해 요청을 자동 라우팅하고, /openapi.json에서 사양을 제공하고 /docs에서 Swagger UI를 제공하며, CORS 및 요청/본문 검증을 즉시 지원합니다.

API를 정의하는 두 가지 방법:

- [Spec-First](#) – 런타임에 기존 OpenAPI 3.0 JSON/YAML 파일을 로드합니다.
- [Code-First](#) – RTTI 속성으로 Delphi 클래스에 주석을 추가하면 사양이 자동으로 생성됩니다(Delphi XE7+ 필요).

[OpenAPI Server](#)에 대해 자세히 알아보십시오.

APIs

다음 API가 컴파일되었으며 지원됩니다:

- [Amazon AWS](#)
- [Google Cloud APIs](#)
- [Microsoft Azure](#)
- [Other APIs](#)

OpenAPI | Parser Pascal

sgcOpenAPI Parser 는 JSON 형식의 OpenAPI 3.0 사양 을 읽고 네이티브 Pascal 코드의 Delphi 클라이언트를 자동으로 생성합니다.

sgcOpenAPI Parser는 다음 사양과 호환됩니다:

- **OpenAPI 3.***
- **Swagger 2.*** (2.0에서 3.0으로 자동 변환됨)
- **Swagger 1.*** (1.0에서 3.0으로 자동 변환됨)

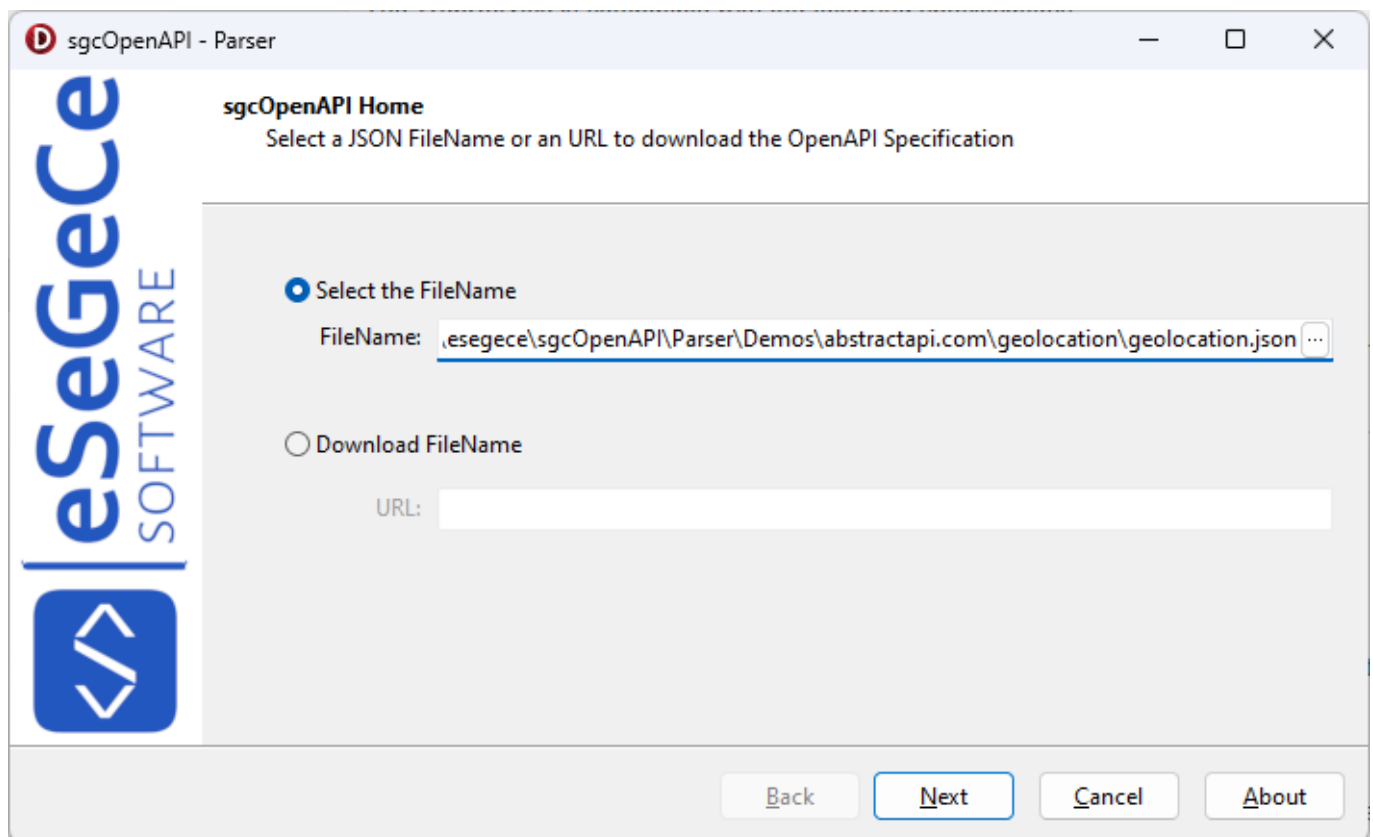
사양 파일은 **JSON** 형식이어야 하며, 사양이 YAML 형식인 경우 자동으로 변환됩니다. 파서를 사용하면 여러 스키마로 분할된 사양을 가져올 수 있습니다. URL을 제공하면 사양을 다운로드하려고 시도하고, 파일 이름을 제공하면 로컬 드라이브에서 외부 스키마를 로드하려고 시도합니다.

OpenAPI Parser 실행 파일은 [명령줄 매개변수](#)를 처리하거나 [라이브러리](#)로 사용할 수 있습니다.

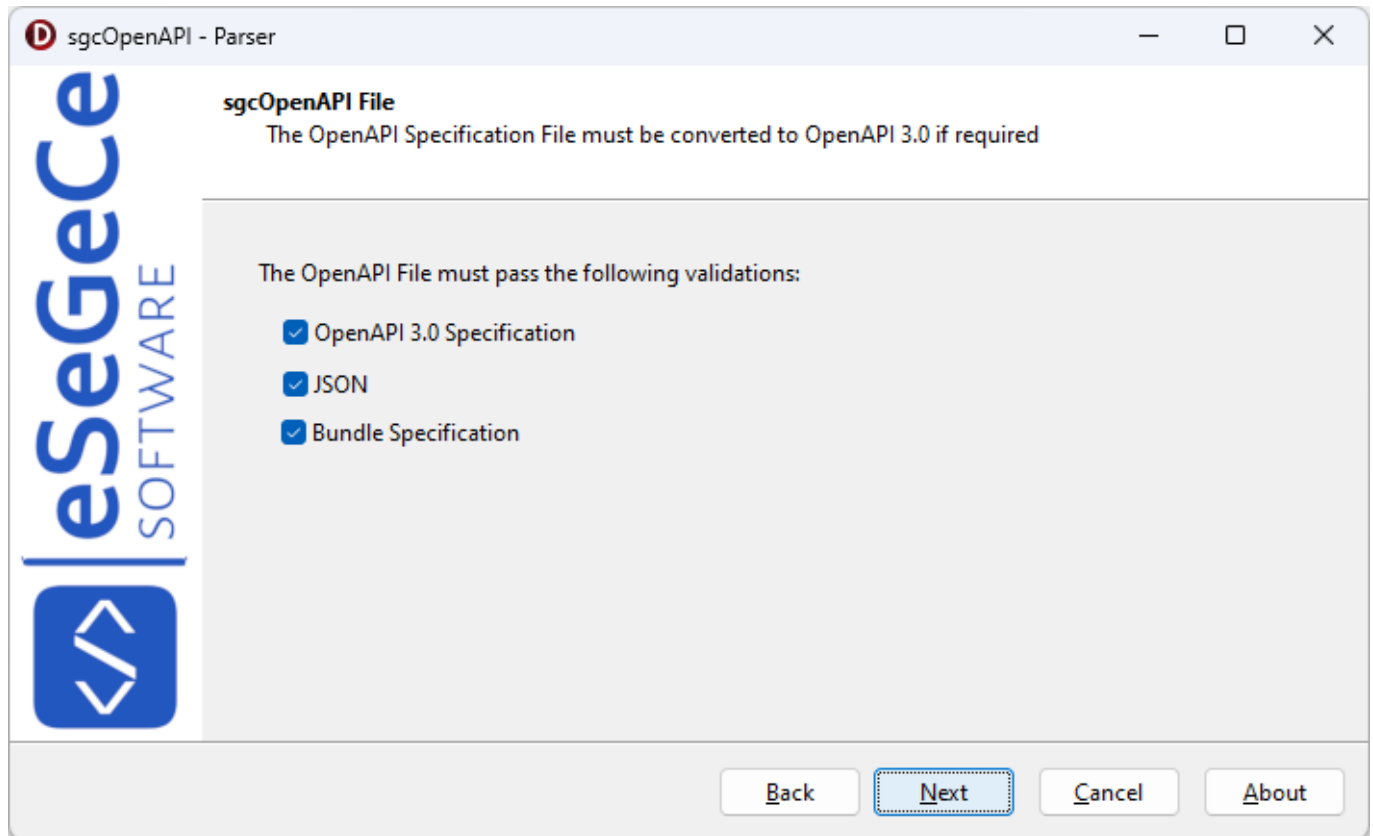
OpenAPI 사양 가져오기

첫 번째 단계는 openAPI 사양을 가져오는 것입니다. JSON 형식의 openAPI 3.0 사양이 있으면 **OpenAPI WebService**를 사용하여 필요한 Delphi 파일을 생성할 수 있습니다. 아래 단계를 따르십시오:

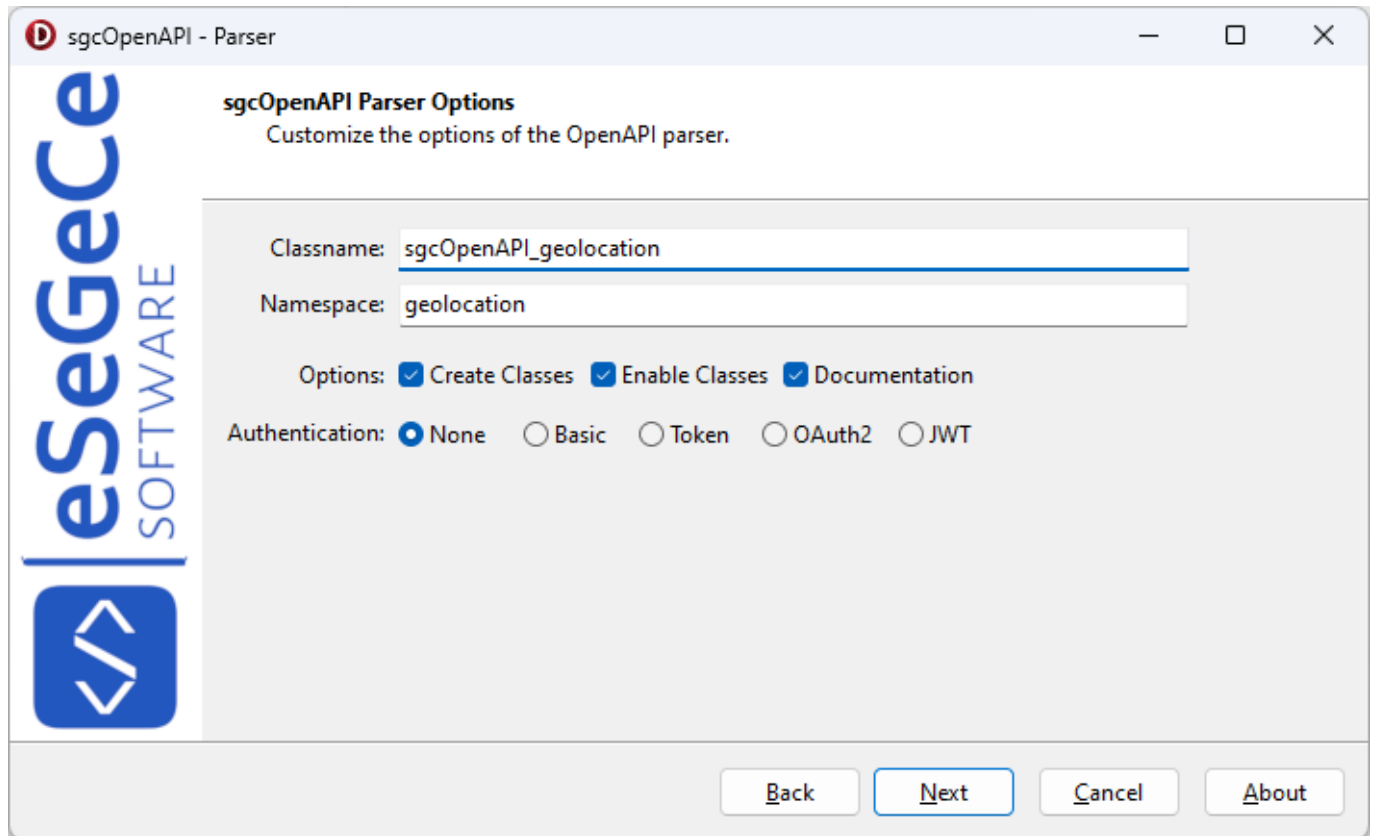
- **sgcOpenAPI Parser를 실행하고** 가져올 사양 또는 다운로드할 URL을 선택하십시오.



- 사양이 검증되고 Parser와 호환되면 다음 단계로 계속 진행할 수 있습니다. 사양이 swagger 2.0과 같은 이전 버전을 사용하면 자동으로 변환됩니다. 사양이 여러 schema로 분할되어 있으면 parser가 단일 사양으로 번들링하려고 시도합니다. 파일을 변환하는 동안 문제가 발생하면 오류 메시지가 나타납니다.



- 이제 문서를 파싱하기 전에 다음 옵션을 사용자 지정할 수 있습니다
 - **ClassName:** 이것은 기본 ClassName이며, 기본적으로 "sgcOpenAPI_"로 시작하고 사양 파일 이름을 추가합니다.
 - **Namespace:** 이것은 생성된 pascal 파일의 이름이며, 기본적으로 확장자 ".pas"가 있는 사양 파일과 동일한 이름입니다.
 - **Create Classes:** 선택하면 사양 파일에서 클래스를 생성합니다. 예: 요청이 JSON 객체를 전송해야 하고 이 JSON 객체가 지정된 경우, 파서는 JSON 객체의 필드가 있는 클래스를 생성합니다.
 - **Enable Classes:** 선택하면 사양 파일에서 생성된 클래스가 JSON 객체와 함께 사용되도록 활성화합니다 (Rad Studio XE7+ 필요).
 - **Documentation:** 선택하면 파서가 필드, 클래스 및 메서드에 주석을 추가합니다.
 - **Authentication:** 존재하는 경우 인증을 선택합니다.
 - **None:** API가 인증 방법을 사용하지 않습니다.
 - **Basic:** API가 BASIC 인증을 사용합니다.
 - **Token:** HTTP 헤더로 토큰을 전송해야 합니다. 이 토큰은 다른 외부 메서드에서 얻습니다.
 - **OAuth2:** 요청이 HTTP 요청을 인증하기 위해 OAuth2를 사용합니다.
 - **JWT:** 요청이 HTTP 요청을 인증하기 위해 JWT를 사용합니다.



sgcOpenAPI Parser Options
Customize the options of the OpenAPI parser.

Classname:

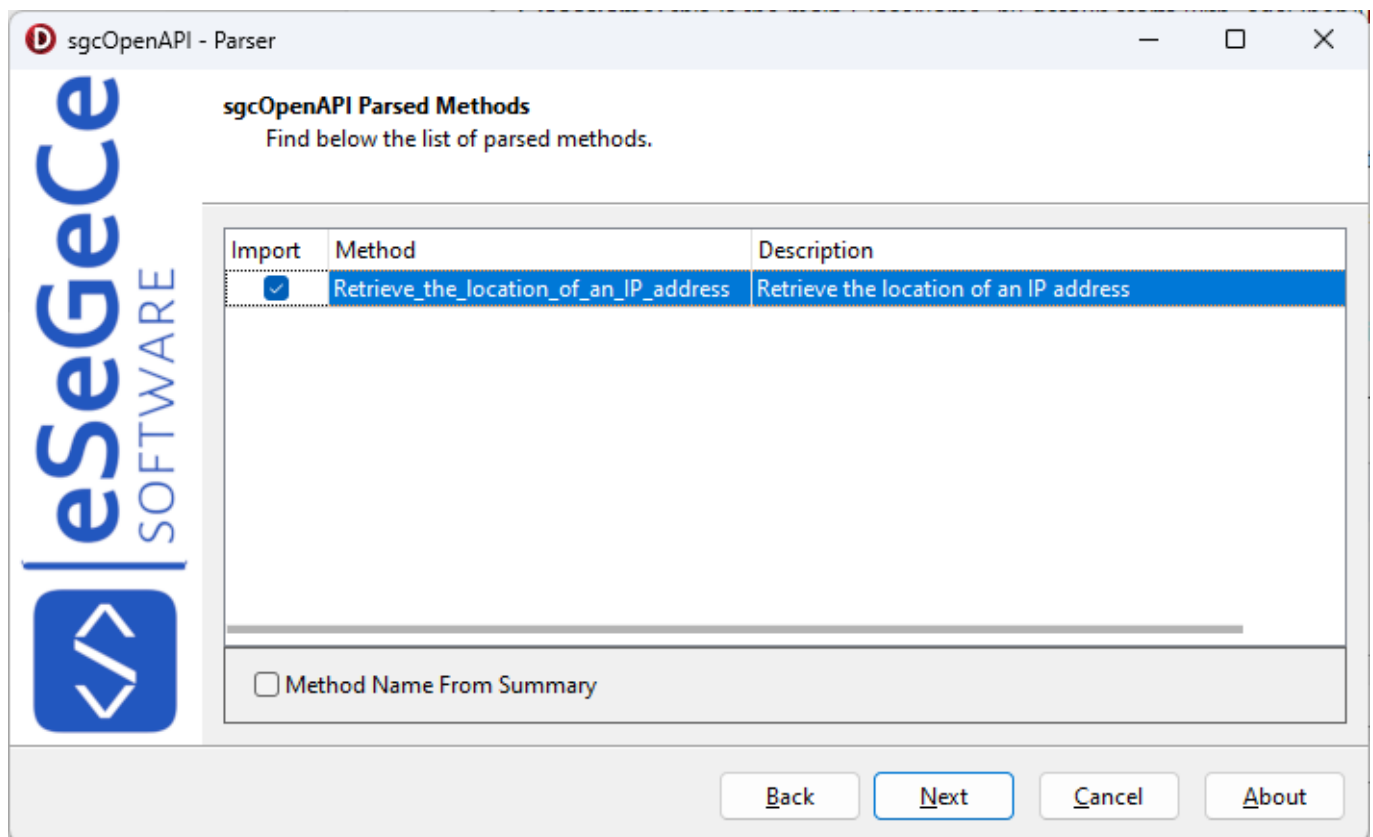
Namespace:

Options: ☒ Create Classes ☒ Enable Classes ☒ Documentation

Authentication: ☒ None ☐ Basic ☐ Token ☐ OAuth2 ☐ JWT

Buttons: Back, Next, Cancel, About

- 이제 파싱된 메서드 목록이 있는 그리드가 정보로 표시됩니다. 기본적으로 파서는 OperationId를 생성된 메서드의 이름으로 사용하지만, OperationId가 정의되지 않았거나 값이 유효하지 않은 경우 여기서 summary를 메서드 이름으로 사용할 수 있습니다.



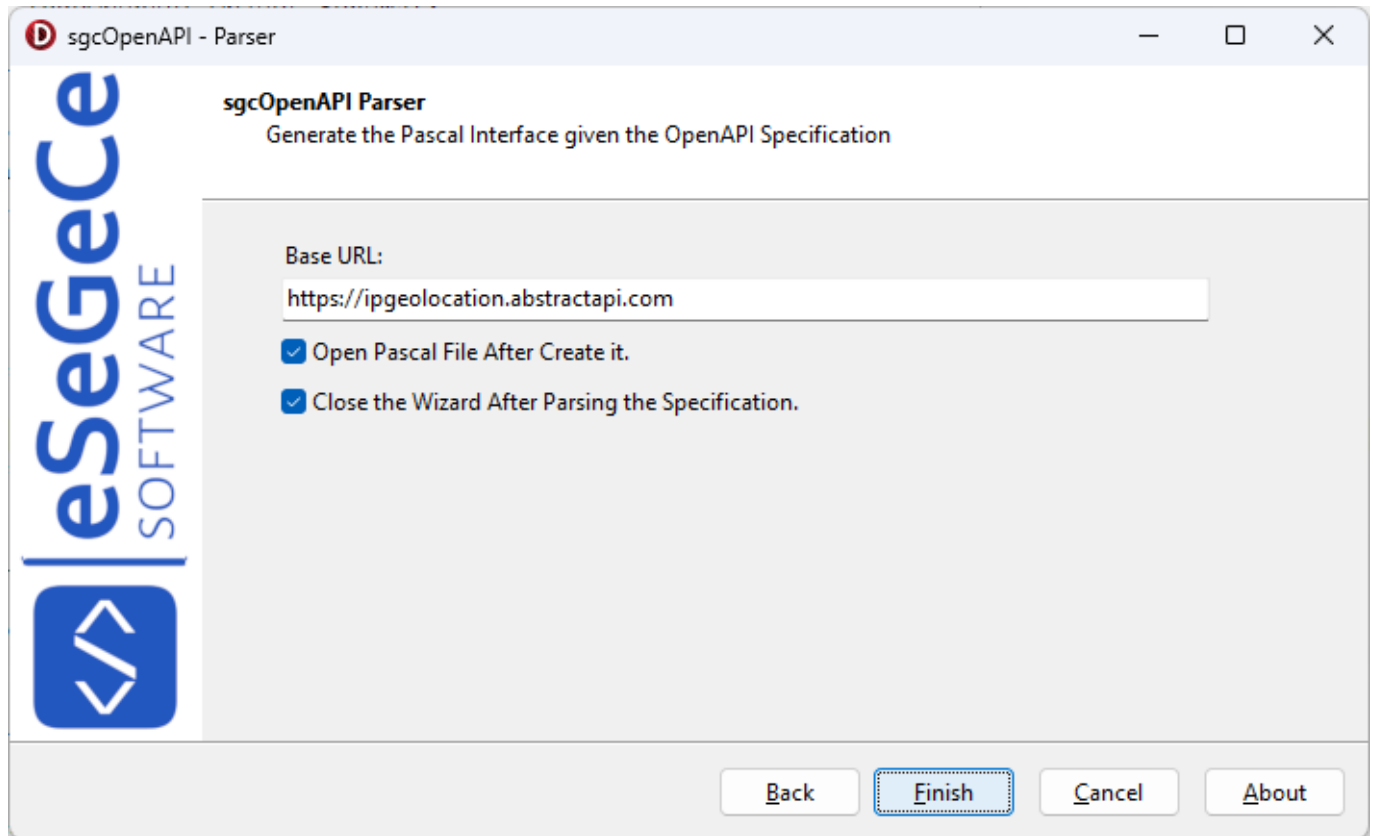
sgcOpenAPI Parsed Methods
Find below the list of parsed methods.

Import	Method	Description
☒	Retrieve_the_location_of_an_IP_address	Retrieve the location of an IP address

☐ Method Name From Summary

Buttons: Back, Next, Cancel, About

- 마지막으로, **Base URL**이 올바른 값을 가지고 있는지 확인하십시오. 여기에서 생성된 파일이 생성 후 자동으로 열릴지, 그리고 파일 생성 후 마법사가 닫힐지 사용자 정의할 수 있습니다.



- **Finish**를 눌러 사양을 파싱하고 pascal 파일을 생성하십시오.

예제

IP 주소의 위치를 검색하기 위해 abstractapi.com에서 사용하는 간단한 openAPI 사양을 사용하겠습니다.

데모를 테스트하기 전에 API Key를 얻으려면 abstractapi.com에서 무료 계정을 생성해야 합니다.

<https://app.abstractapi.com/users/signup>

sgcOpenAPI Parser Setup을 설치하고, sgcOpenAPI.exe를 열어 "Demos\abstractapi.com\geolocation" 폴더에 있는 OpenAPI 사양을 임포트하십시오. 임포트되어 데모와 동일한 폴더에 "geolocation.pas"라는 이름으로 저장되면, 데모 프로젝트를 열고 컴파일한 후 실행하십시오. Abstractapi 웹사이트에서 얻은 API 키를 채우고 Geolocation을 누르십시오.


sgcOpenAPI - Geolocation
— □ ×

API Key:

IP Address:

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

사전(맵, 해시맵 또는 연관 배열이라고도 함)은 키/값 쌍의 집합입니다. OpenAPI를 사용하면 키가 문자열인 사전을 정의할 수 있습니다. 사전을 정의하려면 type: object를 사용하고 additionalProperties 키워드를 사용하여 키/값 쌍의 값 유형을 지정하십시오. 예를 들어, 다음과 같은 문자열 대 문자열 사전:

```
{
  "en": "English",
  "fr": "French"
}
```

OpenAPI Parser는 Embarcadero의 JSON 클래스를 사용하며, TDictionary를 JSON으로 변환할 때 키/값 쌍을 생성하는 대신 TDictionary의 모든 내부 객체를 생성하므로 출력이 올바르지 않습니다. json 문자열을 TDictionary 객체로 변환하려고 할 때도 마찬가지입니다.

Convert AdditionalProperties to JSON

OpenAPI Parser는 AdditionalProperties 클래스를 TsgcAdditionalProperties 유형으로 생성하므로, 클래스를 json 문자열로 변환해야 하는 경우 TsgcAdditionalProperties를 상속하는 새 클래스를 생성하고, 필요한 모든 필드를 생성한 다음 속성 클래스에 할당해야 합니다.

Example: 속성이 2개인 클래스가 주어졌을 때, 하나는 Additional properties이고 json 출력은 다음과 같아야 합니다

```
{"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;
```

TsgcAdditionalProperties에서 상속되는 새 클래스를 생성하고 2개의 속성을 생성하십시오

```
TCustomAdditionalProperties = class(TsgcAdditionalProperties)
private
  FKey1: string;
  FKey2: string;
public
  property Key1: string read FKey1 write FKey1;
  property Key2: string read FKey2 write FKey2;
end;
```

마지막으로 이 클래스를 AdditionalProperties 속성에 할당하십시오:

```
TMyClass *oClass = new TMyClass();
oClass->Property1 = "value1";
TCustomAdditionalProperties *oAdditionalProperties = new TCustomAdditionalProperties();
oAdditionalProperties->Key1 = "value1";
oAdditionalProperties->Key2 = "value2";
oClass->AdditionalProperties = oAdditionalProperties;
```

JSON을 AdditionalProperties로 변환

JSON 클래스의 제한으로 인해 OpenAPI Generator는 JSON 문자열이 파싱된 후 TsgcAdditionalProperties.Dictionary 속성에 AdditionalProperties를 로드하는 추가 메서드를 생성합니다. 따라서 AdditionalProperties의 내용에 액세스하려면 TsgcAdditionalProperties.Dictionary 속성에 액세스하기만 하면 됩니다.

OpenAPI | Command Line

sgcOpenAPI Parser는 OpenAPI 사양을 Delphi (Pascal) 클라이언트 파일로 변환하는 명령줄 인터페이스를 포함합니다. 파서는 출력을 생성하기 전에 사양(OpenAPI 3 JSON + 번들 스키마)을 검증하고 정규화합니다.

Usage

기본 구문

```
sgcOpenAPI.exe -i "c:\openapi.json" -o "c:\openapi.pas"
```

Parameters

모든 매개변수는 대소문자를 구분하지 않습니다. 짧은 형식과 긴 형식은 동등합니다.

- **-i -input** : [필수] OpenAPI 사양의 경로 또는 다운로드할 URL. 입력이 YAML 또는 Swagger 1/2인 경우 자동으로 OpenAPI 3 JSON으로 변환됩니다.
- **-o -output** : [필수] 생성할 출력 Pascal 파일(Delphi 클라이언트)입니다.
- **-nc -nclasses** : [선택 사항] schema에 대한 Delphi 클래스를 생성하지 않습니다(인터페이스 / 클라이언트 메서드 만).
- **-nj -njson** : [선택적] 생성된 클래스에 대해 JSON 직렬화를 활성화하지 않습니다.
- **-nd -ndoc** : [선택적] 출력에 문서 주석을 생성하지 않습니다.
- **-m -method** : [선택 사항] 생성된 메서드에 대한 메서드 명명 전략:
 - **0** : **OperationId** 사용(기본값).
 - **1** : **Summary** 사용.
 - **2** : **Endpoint** 경로 사용.
- **-a -auth** : [선택] 생성된 클라이언트에서 활성화할 인증 모드:
 - **0** : 없음.
 - **1** : Basic.
 - **2** : Token.
 - **3** : OAuth2.
 - **4** : JWT.
- **-u -url** : [선택적] OpenAPI 파일에서 감지된 base URL을 재정의합니다.
- **-l -log** : [선택적] 파싱하는 동안 로그 출력을 표시합니다.
- **-user** : [선택적] 라이선스 사용자 이름(라이선스가 활성화되지 않은 경우에만 필요).
- **-password** : [선택 사항] 라이선스 비밀번호입니다(라이선스가 활성화되지 않은 경우에만 필요).
- **-h -help** : 명령줄 도움말을 표시합니다.

Examples

1) 로컬 파일에서 Delphi 클라이언트 생성

```
sgcOpenAPI.exe -i "c:\specs\openapi.json" -o "c:\clients\orders.pas"
```

2) URL에서 클라이언트 생성, 클래스 및 문서 비활성화

```
sgcOpenAPI.exe -i "https://api.example.com/openapi.json" -o "c:\clients\api.pas" -nc -nd
```

3) Summary를 메서드 이름으로 사용하고 OAuth2 활성화

```
sgcOpenAPI.exe -i "c:\specs\openapi.json" -o "c:\clients\auth.pas" -m 1 -a 3
```

4) Base URL 재정의 및 로그 출력 표시

```
sgcOpenAPI.exe -i "c:\specs\openapi.json" -o "c:\clients\override.pas" -u "https://api.example.com/v2" -l
```

Notes

- 입력 사양이 유효하지 않거나 번들된 OpenAPI 3 JSON 파일이 아닌 경우, 파서는 생성 전에 자동으로 변환하고 번들하려고 시도합니다.
- URL을 사용할 때, 파서는 검증하고 출력을 생성하기 전에 사양을 다운로드합니다.

OpenAPI | Library

OpenAPI Parser는 공유 라이브러리(DLL)로 사용할 수 있으므로 명령줄 도구를 호출하지 않고 자체 애플리케이션에서 Pascal 클라이언트를 생성할 수 있습니다. 이 라이브러리는 CLI에서 사용 가능한 것과 동일한 파서 기능을 래핑하는 소수의 함수 집합을 내보냅니다.

Library functions

- **CreateOpenAPIPascalFile** - 기본 옵션을 사용하여 OpenAPI 사양에서 Pascal 클라이언트 파일을 생성합니다.
- **CreateOpenAPIPascalFileEx** - 파서 옵션을 완전히 제어하면서 Pascal 클라이언트 파일을 생성합니다.
- **GetOpenAPILastErrors** - 파서가 생성한 마지막 오류 메시지를 반환합니다.

CreateOpenAPIPascalFile

기본 parser 옵션을 사용하여 Pascal 클라이언트 파일을 생성합니다(클래스 생성, JSON 지원, 문서화, **OperationId**의 메서드 이름, 인증 없음, 기본 URL 재정의 없음).

```
typedef bool (__stdcall *TCreateOpenAPIPascalFile)(const char* aInputFile, const char* aOutputFile);
```

매개변수

- **aInputFile**: 파싱할 OpenAPI 사양의 전체 경로(또는 URL). 파일이 YAML 또는 Swagger 1/2인 경우 파서는 출력을 생성하기 전에 이를 OpenAPI 3 JSON으로 정규화합니다.
- **aOutputFile**: 생성할 Pascal unit의 전체 경로입니다.

반환 값

파일이 성공적으로 생성되면 **True**를 반환하고, 그렇지 않으면 **False**를 반환합니다. 실패하면 **GetOpenAPILastErrors**를 호출하여 오류 메시지를 검색하십시오.

예제

CreateOpenAPIPascalFileEx

파서 옵션을 명시적으로 제어하면서 Pascal 클라이언트 파일을 생성합니다.

```
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);
```

매개변수

- **aInputFile**: 구문 분석할 OpenAPI 사양의 전체 경로(또는 URL).
- **aOutputFile**: 생성할 Pascal unit의 전체 경로입니다.
- **aGenerateClasses**: **True**인 경우 생성기는 스키마에 정의된 Pascal 모델(클래스)을 생성합니다. **False**인 경우 클라이언트 인터페이스만 생성됩니다.
- **aGenerateJSON**: **True**인 경우 클래스에 JSON 직렬화 헬퍼가 생성됩니다. **False**인 경우 JSON 지원이 생략됩니다.
- **aGenerateDocumentation**: **True**인 경우, OpenAPI 사양에 대한 문서 파일이 생성됩니다.
- **aMethodName**: 메서드 이름이 생성되는 방식을 제어합니다:
 - **0** = OperationId(기본값).

- **1** = Summary.
- **2** = Endpoint.
- **aAuthentication:** 클라이언트에 인증 헬퍼를 추가합니다:
 - **0** = None.
 - **1** = Basic.
 - **2** = Token.
 - **3** = OAuth2.
 - **4** = JWT.
- **aBaseURL:** OpenAPI 사양의 기본 URL을 재정의합니다. 원래 값을 유지하려면 빈 문자열을 전달하십시오.
- **aEnableLog:** **True**인 경우, 파서는 문제 해결에 도움이 될 수 있는 진행 및 유효성 검사 메시지를 기록합니다.

반환 값

파일이 성공적으로 생성되면 **True**를 반환하고, 그렇지 않으면 **False**를 반환합니다. 실패하면 **GetOpenAPILastErrors**를 호출하여 오류 메시지를 검색하십시오.

예제

GetOpenAPILastErrors

파서가 생성한 마지막 오류(검증, 변환 또는 파싱 오류)를 설명하는 메시지를 반환합니다. **CreateOpenAPIPascalFile** 또는 **CreateOpenAPIPascalFileEx**가 **False**를 반환한 후 이 함수를 호출하십시오.

```
typedef const char* (__stdcall *TGetOpenAPILastErrors)();
```

반환 값

오류 수를 반환합니다

예제

OpenAPI | API

sgcOpenAPI Parser는 파싱된 pascal 파일의 더 자세한 정보를 얻기 위해 사용자 정의 dll을 사용할 수 있습니다. **sgcOpenAPI_CreateSection** 메서드가 있는 새 dll을 만들고 파서가 있는 동일한 폴더에 라이브러리를 복사하기만 하면 됩니다. 새 파일 섹션이 생성될 때마다 api는 CreateSection 메서드를 호출하며 필요한 경우 내용을 수정할 수 있습니다. 아래에서 간단한 delphi 예제를 확인하십시오:

```
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 는 OpenAPI 사양에서 HTTP 요청을 수행하는 주요 메서드와 속성을 캡슐화하는 비시각적 구성 요소입니다.

sgcOpenAPI Parser로 생성된 모든 OpenAPI 인터페이스에는 **2개의 메서드**가 있습니다.

1. **GetOpenAPIClient**: 메인 클래스의 인스턴스를 반환하는 싱글톤 함수이며, 존재하지 않으면 자동으로 생성합니다.
2. **FreeOpenAPIClient**: 메인 클래스가 생성된 경우 해제합니다.

예제

Abstractapi를 사용하여 IP 주소의 위치 정보를 검색합니다.

```
GetOpenAPIClient->Retrieve_the_location_of_an_IP_address("your api", "80.258.15.2");
```

Authentication

- **Basic**: 사용자 이름/비밀번호를 HTTP 헤더로 사용하여 인증합니다.
 - **UserName**: 사용자의 이름.
 - **Password**: 비밀.
- **OAuth2**: OAuth2를 사용하여 인증하며, 2가지 유형의 권한 부여를 지원합니다:
 - **auth2Code**: 단일 페이지 애플리케이션, 웹 애플리케이션, 네이티브 설치 애플리케이션을 포함한 대부분의 애플리케이션 유형에서 인증 및 권한 부여를 수행하는 데 사용됩니다. 이 흐름을 통해 앱은 보안 리소스에 액세스하는 데 사용할 수 있는 access_token, 추가 access_token을 얻기 위한 갱신 토큰, 그리고 로그인한 사용자의 ID 토큰을 안전하게 획득할 수 있습니다.
 - **auth2ClientCredentials**: 이 유형의 부여는 일반적으로 사용자와의 즉각적인 상호 작용 없이 백그라운드에서 실행되어야 하는 서버 간 상호 작용에 사용됩니다. 이러한 유형의 애플리케이션은 종종 데몬 또는 서비스 계정이라고 합니다.
 - 자세한 내용은 [OAuth2](#)를 참조하십시오.
- **JWT**: JWT를 사용하여 인증합니다. [JWT](#)에 대해 더 읽어보십시오.
- **Token**: bearer 토큰 또는 기타 사용자 지정 토큰 값을 전달할 수 있습니다.

TLSOptions

HTTP/1 프로토콜을 사용하여 보안 SSL/TLS 서버에 연결하는 방법을 구성할 수 있습니다

ALPNProtocols: 서버로 전송될 ALPN 프로토콜 목록입니다.

RootCertFile: 루트 인증서 파일의 경로입니다.

CertFile: 인증서 파일 경로입니다.

KeyFile: 인증서 키 파일의 경로입니다.

Password: 인증서가 비밀번호로 보호된 경우 여기에 설정하십시오.

VerifyCertificate: 인증서를 검증해야 하는 경우 이 속성을 활성화하십시오.

VerifyDepth: X.509 인증서에 대해 검증이 수행될 때 허용되는 최대 링크 수를 나타내는 Integer 속성입니다.

Version: 기본적으로 TLS 1.0을 사용하며, 서버가 더 높은 TLS 버전을 요구하는 경우 여기에서 선택할 수 있습니다.

IOHandler: TLS를 사용하여 연결하는 데 사용할 라이브러리를 선택하십시오.

iohOpenSSL: OpenSSL 라이브러리를 사용하며 Indy 구성 요소의 기본값입니다. win32/win64용 openssl 라이브러리를 배포해야 합니다.

iohSChannel: Microsoft가 Windows용으로 구현한 보안 프로토콜인 Secure Channel을 사용하며, openssl 라이브러리를 배포할 필요가 없습니다. Windows 32/64비트에서만 작동합니다.

OpenSSL_Options: openssl 라이브러리의 구성입니다.

APIVersion: 어떤 OpenSSL API를 사용할지 정의할 수 있게 합니다.

osIAPI_1_0: API 1.0 OpenSSL을 사용하며, Indy가 지원하는 최신 버전입니다

osIAPI_1_1: API 1.1 OpenSSL을 사용하며, 당사의 사용자 정의 Indy 라이브러리가 필요하고 OpenSSL 1.1.1 라이브러리(TLS 1.3 지원 포함)를 사용할 수 있게 합니다.

oslAPI_3_0: API 3.0 OpenSSL을 사용하며, 사용자 지정 Indy 라이브러리가 필요하고 OpenSSL 3.0.0 라이브러리(TLS 1.3 지원 포함)를 사용할 수 있습니다.

LibPath: 여기에서 openssl 라이브러리가 위치한 곳을 구성할 수 있습니다.

oslpNone: 이것이 기본값이며, openssl 라이브러리는 바이너리가 있는 동일한 폴더 또는 알려진 경로에 있어야 합니다.

oslpDefaultFolder: 모든 IDE 퍼스널리티에 대해 라이브러리가 위치해야 하는 openssl 경로를 자동으로 설정합니다.

oslpCustomFolder: 이 옵션이 선택된 경우, LibPathCustom 속성에 전체 경로를 정의하십시오.

LibPathCustom: LibPath = oslpCustomFolder일 때 openssl 라이브러리가 있는 전체 경로를 여기에 정의하십시오.

UnixSymLinks: Unix 시스템에서 SymLinks 로딩을 활성화하거나 비활성화합니다(기본적으로 활성화되어 있으며, OSX64에서는 예외):

oslsSymLinksDefault: 기본적으로 활성화되지만 OSX64에서는 예외입니다(MacOS Monterey 이후 버전 없이 라이브러리를 로드하려고 하면 실패함).

oslsSymLinksLoadFirst: 버전 라이브러리를 로드하기 전에 SymLink를 로드하고 먼저 시도합니다.

oslsSymLinksLoad: 버전 라이브러리를 로드하려고 시도한 후 SymLinks를 로드합니다.

oslsSymLinksDontLoad: SymLinks를 로드하지 않습니다.

SChannel_Options: Windows Certificate Store의 인증서를 사용할 수 있게 합니다.

CertHash: 는 인증서 Hash입니다. powershell에서 dir 명령을 실행하여 인증서 Hash를 찾을 수 있습니다.

CipherList: 여 기 에 서 어 떤 Cipher 가 사 용 될 지 설 정 할 수 있 습 니 다(":"로 구 분). 예:

CALG_AES_256:CALG_AES_128

CertStoreName: 인증서가 저장된 스토어 이름. 아래 중 하나를 선택하십시오:

scsnMY(기본값)

scsnCA

scsnRoot

scsnTrust

CertStorePath: 인증서가 저장된 저장소 경로입니다. 아래에서 하나를 선택하십시오:

scspStoreCurrentUser (기본값)

scspStoreLocalMachine

Proxy Options

프록시를 통한 연결을 구성하려면 이 속성을 사용하십시오.

Enabled: 프록시 연결을 활성화하려면 true로 설정하십시오.

Host: 프록시 서버 주소

Port: 프록시 서버 포트

UserName/Password: 프록시에 연결하기 위한 인증입니다. 필요한 경우에만 사용합니다.

ProxyType: 다음 프록시가 지원됩니다:

- HTTP
- Socks4
- Socks4A
- Socks5

Log

Log 속성이 활성화되면 소켓 메시지를 지정된 로그 파일에 저장하며, 디버깅에 유용합니다.

Log: HTTP 요청을 텍스트 파일에 저장하려면 활성화하십시오.

LogFileName: 파일 이름의 전체 경로입니다.

Properties

OpenAPI 클라이언트를 사용자 정의하는 데 사용할 수 있는 기타 속성:

EncodeBodyAsUTF8: 활성화하면 JSON 본문이 UTF8로 인코딩됩니다(기본값 false).

Events

OpenAPI Client를 사용할 때 처리할 수 있는 이벤트 목록은 아래를 참조하십시오.

OnBeforeRequest

이 이벤트는 HTTP 요청이 호출되기 전에 호출됩니다. 매개변수 이름, 헤더, 보안 등을 사용자 지정할 수 있습니다. 아래에서 일부 매개변수의 이름을 교체하는 방법의 예제를 확인하십시오.

```
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

이 이벤트는 파일이 업로드될 때 호출됩니다. 이 이벤트를 사용하여 업로드 진행 상황을 알 수 있습니다.

OnDownload

이 이벤트는 파일이 다운로드될 때 호출되며, 이 이벤트를 사용하여 다운로드의 진행 상황을 알 수 있습니다.

OnSSLVerifyPeer

verify certificate가 활성화된 경우, 이 이벤트에서 서버 인증서를 확인하고 수락할지 여부를 결정할 수 있습니다.

OnSSLGetHandler

이 이벤트는 SSL 핸들러가 생성되기 전에 발생합니다. 여기에서 자체 SSL 핸들러(TIdServerIOHandlerSSLBase 또는 TIdIOHandlerSSLBase에서 상속되어야 함)를 생성하고 필요한 속성을 설정할 수 있습니다

OnSSLAfterCreateHandler

사용자 정의 SSL 객체가 생성되지 않은 경우, OpenSSL 핸들러를 사용하여 기본적으로 생성합니다. SSL Handler 속성에 액세스하여 필요한 경우 수정할 수 있습니다

OpenAPI | Server

sgcOpenAPI Server는 계약이 OpenAPI 3.0 사양으로 설명되는 HTTP 서비스를 호스팅합니다. 라우트는 사양의 **paths** 섹션에서 자동으로 빌드되고, Swagger UI가 즉시 노출되며, 수신 요청은 각 작업에서 선언된 JSON-Schema에 대해 검증될 수 있습니다. 서버를 호스팅하기 위해 두 가지 구성 요소가 제공됩니다:

- **TsgcHTTPServer_OpenAPI**: 독립형 HTTP 서버입니다. 자체 완결형으로 Indy HTTP 서버를 내장하며 sgcWebSockets가 필요하지 않습니다. OpenAPI 서비스만 노출하면 되는 경우 권장됩니다.
- **TsgcWSAPIServer_OpenAPI**: sgcWebSockets와 호환됩니다. 기존 **TsgcWebSocketHTTPServer**에 연결되어 동일한 서버가 WebSocket과 HTTP/OpenAPI 트래픽을 모두 처리합니다.

두 구성 요소는 동일한 엔진, 옵션, 이벤트를 공유합니다.

TsgcHTTPServer_OpenAPI

sgcHTTPServer_OpenAPI 유닛에 정의된 독립형 서버입니다. 수신 대기 포트를 구성하고, OpenAPI 스펙을 로드하고, 서버를 활성화하십시오.

TsgcWSAPIServer_OpenAPI

sgcWebSockets와 호환되며, **sgcWebSocket_Server_API_OpenAPI** 유닛에 정의되어 있습니다. 기존 **TsgcWebSocketHTTPServer**를 **Server** 속성에 할당하면 단일 서버 인스턴스가 WebSocket 프레임과 OpenAPI HTTP 요청을 모두 디스패치할 수 있습니다. 내부적으로 독립형 구성 요소가 사용하는 동일한 엔진에 위임하므로 API surface가 동일합니다.

Built-in Endpoints

서버는 다음 엔드포인트를 자동으로 제공합니다(각 엔드포인트는 **OpenAPIOptions.Endpoint**를 통해 비활성화할 수 있습니다):

- /openapi.json**: 로드된 OpenAPI 사양을 반환하며, **OpenAPIOptions.Endpoint.ServeSpec**으로 제어됩니다.
- /docs**: API를 탐색하고 시험해 볼 수 있는 내장 Swagger UI이며, **OpenAPIOptions.Endpoint.ServeSwaggerUI**로 제어됩니다.
- OPTIONS ***: **OpenAPIOptions.CORS.Enabled**가 True일 때 전송되는 CORS 프리플라이트 응답입니다.

OpenAPIOptions

TsgcOpenAPIServer_Options 유형의 최상위 구성 속성입니다. 세 개의 하위 객체를 그룹화합니다:

Endpoint: 내장 엔드포인트 및 사양 로드.

BasePath: 스펙에 선언된 모든 라우트 앞에 추가되는 선택적 URL 접두사입니다.

SpecFile: 할당된 경우, 서버가 **Active** 상태가 되면 파일이 자동으로 로드됩니다.

ServeSpec: True(기본값)인 경우 로드된 사양이 **/openapi.json**에서 제공됩니다.

ServeSwaggerUI: True(기본값)이면 Swagger UI가 **/docs**에서 제공됩니다.

Validation: 요청 검증 플러그, 모두 기본값은 False입니다.

ValidateRequest: 전체 요청에 대한 검증을 활성화하는 마스터 스위치입니다.

ValidateRequestBody: 사양에 선언된 JSON-Schema에 대해 요청 본문을 검증합니다.

ValidateQueryParams: 선언된 스키마에 대해 쿼리 매개변수를 검증합니다.

ValidatePathParams: 경로 매개변수를 선언된 스키마에 대해 검증합니다.

ValidateRequired: 필수로 선언된 매개변수가 누락되면 요청을 거부합니다.

CORS: cross-origin 지원입니다.

Enabled: True인 경우 모든 응답에 CORS 헤더가 방출되고 **OPTIONS** 프리플라이트 요청에 자동으로 응답합니다.

AllowOrigins: **Access-Control-Allow-Origin**의 값(기본값 *).

AllowHeaders: **Access-Control-Allow-Headers**의 값입니다(기본값 **Content-Type, Authorization**).

AllowMethods: **Access-Control-Allow-Methods**의 값입니다(기본값 **GET, POST, PUT, DELETE, PATCH, OPTIONS**).

Events

OpenAPI Server를 사용할 때 처리할 수 있는 이벤트 목록은 아래를 참조하십시오.

OnBeforeRequest: 요청이 **OnRequest**로 디스패치되기 전에 발생합니다. 403 응답으로 요청을 거부하려면 **Accept** 매개변수를 **False**로 설정하십시오.

OnRequest: 메인 핸들러. **aOperationId**를 검사하고 **aContext**를 사용하여 요청을 읽고 응답을 기록하십시오. 응답을 생성한 후 **Handled**를 **True**로 설정하십시오.

OnAfterRequest: 사용자 핸들러가 반환된 후 발생합니다. 로깅 또는 메트릭에 유용합니다.

OnAuthenticate: 인증이 필요할 때 디스패치 전에 발생합니다. 401 응답으로 거부하려면 **Authenticated**를 **False**로 설정하십시오.

OnValidationError: 하나 이상의 검증 규칙이 실패하면 발생합니다. **aErrors**를 검사하고 400 응답으로 요청을 거부하려면 **Continue**를 **False**로 설정하십시오.

OnException: 사용자 핸들러에서 처리되지 않은 예외가 전파될 때 발생합니다. 기본 500을 대체해야 하는 경우 **aResponseCode**를 조정하십시오.

Spec-First vs Code-First

서버는 두 가지 작성 방식을 지원합니다:

- **Spec-First:** OpenAPI 3.0 JSON/YAML 파일이 정보의 원천입니다. 서버는 이를 파싱하여 라우트를 검색합니다.
- **Code-First:** RTTI 속성으로 주석이 달린 Delphi 클래스가 신뢰 소스입니다. OpenAPI 사양은 시작 시 생성됩니다. Delphi XE7 이상이 필요합니다.

OpenAPI | Server | Spec-First

Spec-First 접근 방식에서는 OpenAPI 3.0 사양(JSON 또는 YAML)이 신뢰할 수 있는 소스입니다: 모든 path, operation, 매개변수 및 schema를 설명합니다. 서버가 시작되면 사양을 구문 분석하고 라우트 테이블을 자동으로 빌드합니다. 각 operation에 대한 비즈니스 로직만 제공하면 됩니다.

이 워크플로는 API 계약이 이미 존재하거나, 소비자와 공유되거나, Delphi 코드 베이스 외부에서 유지 관리되는 경우 권장됩니다.

스펙 로딩

디스크에서 사양을 로드하려면 **LoadFromFile**을, 메모리 내 문자열(예: 데이터베이스에서 검색했거나 런타임에 생성된 사양)에서 로드하려면 **LoadFromString**을 사용하십시오.

로드되면, 라우트 테이블은 사양의 **paths** 섹션에서 빌드됩니다. **/pets/{petId}**와 같은 템플릿 세그먼트가 자동으로 인식되며 캡처된 값은 **aContext.PathParamAsXxx**를 통해 노출됩니다.

작업 처리

서버는 모든 수신 요청을 **OnRequest** 이벤트로 디스패치하며, 스펙에 선언된 **operationId**를 전달합니다. 작업을 올바른 핸들러로 라우팅하려면 간단한 **if/else** 체인(또는 사전 조회)을 사용하십시오.

요청 데이터 읽기

TsgcOpenAPIServerContext 객체는 요청의 모든 부분에 대한 형식화된 접근자를 노출합니다:

응답 작성

컨텍스트 객체는 응답을 작성하는 헬퍼도 제공합니다. 가장 일반적인 헬퍼는 **RespondJSON**과 **RespondError**입니다. 응답에 대한 전체 제어를 위해서는 **aContext.Response.Code**와 **aContext.Response.ContentText**를 직접 설정하십시오.

Validation

OpenAPIOptions.Validation 아래의 유효성 검사 플래그가 활성화되면, 서버는 **OnRequest**를 호출하기 전에 사양에 선언된 JSON-Schema에 대해 모든 요청을 유효성 검사합니다:

ValidateRequestBody: 요청 본문이 작업의 **requestBody**가 참조하는 스키마와 대조하여 검사됩니다.

ValidateQueryParams: 쿼리 문자열 매개변수가 선언된 스키마와 대조하여 검사됩니다.

ValidatePathParams: 경로 매개변수가 선언된 스키마에 대해 확인됩니다.

ValidateRequired: 필수로 선언된 매개변수가 누락되면 요청이 거부됩니다.

검증이 실패하면 **OnValidationError** 이벤트가 오류 목록과 함께 발생합니다. 400 응답으로 요청을 거부하려면 **Continue**를 **False**로 설정하고, **OnRequest** 핸들러가 결정하도록 하려면 **True**로 두십시오.

Demo

완전한 spec-first 샘플이 **Demos/30.Server/02.SpecFirst/** 아래에 제공됩니다.

OpenAPI | Server | Code-First

Code-First 접근에서 Delphi 클래스는 진실의 원천입니다. 메서드, 매개변수 및 RTTI 속성이 API 계약을 설명하며, OpenAPI 3.0 사양은 시작 시 이들로부터 생성됩니다. 생성되면 사양은 **LoadFromString**으로 서버에 로드되고 라우트 테이블은 파일에서 온 것처럼 빌드됩니다.

Code-First 는 새 RTTI 를 사 용 하 여 attribute 를 검 사 하 므 로 **Delphi XE7** 이 상 이 필 요 합 니 다. 기 능 은 **sgcHTTP_OpenAPI_Server_CodeFirst** 유닛에 있으며 컴파일러 지시문 **{\$IFDEF DXE7}**로 보호됩니다.

서비스 정의

API를 설명하기 위해 속성으로 Delphi 클래스에 주석을 담니다. 메서드 본문은 스텝입니다: 실제 로직은 서버의 **OnRequest** 이벤트에 들어갑니다. 메서드는 RTTI 스캐너가 시그니처를 읽을 수 있도록 존재할 뿐입니다.

Available Attributes

다음 속성은 스캐너에 의해 인식됩니다. 클래스 수준 속성은 서비스 전체를 설명하고, 메서드 수준 속성은 단일 작업을 설명하며, 매개변수 수준 속성은 값의 출처를 설명합니다.

sgcServiceContract(name, description, version): 클래스 수준입니다. OpenAPI **info** 블록을 채웁니다.

sgcRoute(path): 클래스 수준 기본 경로, 또는 클래스 기본 경로에 상대적인 메서드 수준 경로.

sgcHttpGet, sgcHttpPost, sgcHttpPut, sgcHttpDelete, sgcHttpPatch, sgcHttpHead, sgcHttpOptions: 메서드 수준. 작업의 HTTP 동사를 선언합니다.

sgcSummary(text): 메서드 수준. 사양과 Swagger UI에 표시되는 짧은 요약입니다.

sgcDescription(text): 메서드 수준입니다. 사양에 대한 긴 설명입니다.

sgcTag(name): 메서드 수준입니다. Swagger UI에서 동일한 태그로 작업을 그룹화합니다.

sgcResponse(code, description): 메서드 수준. HTTP 상태 코드와 설명이 있는 응답을 선언합니다. 여러 응답을 선언하기 위해 반복할 수 있습니다.

sgcFromPath, sgcFromQuery, sgcFromBody, sgcFromHeader: 매개변수 수준입니다. 매개변수를 읽는 위치를 선언합니다.

sgcRequired: 매개변수 수준입니다. 매개변수를 필수로 표시합니다. **ValidateRequired**가 활성화되면 누락된 매개변수가 검증에 의해 보고됩니다.

스펙 생성 및 로딩

주석이 달린 클래스에서 OpenAPI 3.0 사양을 생성하려면 **TsgcOpenAPICodeFirstScanner**를 사용한 다음, 결과를 서버에 로드하십시오.

작업 처리

스캐너는 메서드 이름에서 **operationId**를 파생합니다: **ListTasks** 메서드는 **ListTasks** operation이 됩니다. 동일한 **operationId**를 사용하여 **OnRequest**에서 디스패치하십시오.

Demo

완전한 code-first 샘플이 **Demos/30.Server/01.CodeFirst/** 아래에 제공됩니다.

OpenAPI | Amazon AWS

sgcOpenAPI Amazon AWS Client(TsgcOpenAPI_Amazon_Client)는 [TsgcOpenAPI_Client](#)에서 상속되는 자체 OpenAPI Client를 가지고 있습니다.

이 구성 요소에는 Amazon AWS 서버에 연결하는 데 필요한 모든 구성을 포함하는 **AmazonOptions**라는 속성이 있습니다.

AmazonOptions

AmazonOptions에서 Amazon AWS Servers에 대해 인증하기 위해 필요한 **AccessKey** 및 **SecretKey** (Amazon 계정에 서 미리 생성해야 함)를 정의할 수 있습니다.

access key는 리소스에 대한 프로그래밍 방식 액세스를 부여합니다. 즉, AWS 계정 root 사용자 로그인 자격 증명만큼 신중하게 access key를 보호해야 합니다.

다음은 수행하는 것이 모범 사례입니다:

1. **IAM 사용자를 생성한 다음 해당 사용자의 권한을 정의**하되 가능한 한 좁게 정의하십시오.
2. 해당 IAM 사용자 아래에서 **액세스 키를 생성**하십시오.

자격 증명을 받으면 다음 속성에 설정하십시오:

- AmazonOptions.AccessKey
- AmazonOptions.SecretKey

AmazonOptions.JSON 속성을 사용하면 응답이 JSON인지 XML인지 정의할 수 있습니다.

IAM 역할, AWS IAM Identity Center(AWS Single Sign-On의 후속)의 사용자, 그리고 federated 사용자는 임시 보안 자격 증명을 가집니다. 임시 보안 자격 증명은 정의된 기간 후 또는 사용자가 세션을 종료할 때 만료됩니다. 다음 속성에서 임시 자격 증명에 대한 토큰을 설정할 수 있습니다:

- AmazonOptions.SessionToken

Most common uses

- **Configuration**
 - [Amazon AWS Credentials](#)
- **APIs**
 - [Amazon AWS S3](#)

sgcOpenAPI AWS SDK

아래에서 현재 사용 가능한 API 목록을 찾을 수 있습니다.

- 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
- AWS CodeStar Notifications
- 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

AWS는 AWS에 액세스하는 방법에 따라 다양한 유형의 보안 자격 증명을 요구합니다. 예를 들어, AWS Management Console에 로그인하려면 사용자 이름과 비밀번호가 필요하고, **AWS에 대한 프로그래밍 방식 호출**을 하려면 **access keys**가 필요합니다.

고려 사항

- 다음 항목을 안전한 위치에 저장하십시오: AWS 계정과 연결된 이메일 주소, AWS 계정 ID, root 사용자 비밀번호 및 계정 액세스 키. root 사용자 비밀번호를 잊어버리거나 분실한 경우 이를 재설정하려면 계정과 연결된 이메일 주소에 접근할 수 있어야 합니다. 액세스 키를 잊어버리거나 분실한 경우 새 키를 생성하려면 계정에 로그인해야 합니다.
- 일상적인 AWS 작업에 사용할 관리자 권한이 있는 IAM 사용자를 생성하고 root 사용자의 비밀번호 및 액세스 키를 안전하게 보관할 것을 강력히 권장합니다. root 사용자는 root 사용자로 제한된 작업에만 사용하십시오.
- 보안 자격 증명은 계정별로 다릅니다. 여러 AWS 계정에 액세스할 수 있는 경우, 각 계정에 대해 별도의 자격 증명을 갖게 됩니다.
- AWS 자격 증명을 타사에 제공하지 마십시오.

프로그래밍 방식 접근

AWS에 프로그래밍 방식으로 호출하려면 AWS 액세스 키를 제공해야 합니다.

액세스 키를 생성할 때 access key ID(예: AKIAIOSFODNN7EXAMPLE)와 secret access key(예: wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY)를 한 세트 생성합니다. secret access key는 생성 시에만 다운로드할 수 있습니다. secret access key를 다운로드하지 않았거나 분실한 경우 새로 생성해야 합니다.

사용자(루트 사용자 또는 IAM 사용자)당 최대 두 개의 access key를 할당할 수 있습니다. 두 개의 access key를 가지는 것은 회전하려는 경우 유용합니다. access key를 비활성화하면 사용할 수 없지만 두 개의 access key 제한에는 포함됩니다. access key를 삭제하면 영원히 사라져 복원할 수 없지만 새 access key로 교체할 수 있습니다.

root 사용자로 로그인했을 때 액세스 키를 관리하려면

1. 루트 사용자로 AWS Management Console에 로그인하십시오.
2. 오른쪽 상단의 탐색 모음에서 계정 이름 또는 번호를 선택한 다음 **My Security Credentials**를 선택하십시오.
3. **Access keys (access key ID and secret access key)** 섹션을 확장하십시오.
4. 다음 중 하나를 수행하십시오:
 - 액세스 키를 생성하려면 **Create New Access Key**를 선택하십시오. 이미 두 개의 액세스 키가 있는 경우 이 버튼은 비활성화되며 새 키를 생성하기 전에 액세스 키를 삭제해야 합니다. 메시지가 표시되면 **Show Access Key** 또는 **Download Key File**을 선택하십시오. 이것은 비밀 액세스 키를 저장할 수 있는 유일한 기회입니다. 비밀 액세스 키를 안전한 위치에 저장한 후 **Close**를 선택하십시오.
 - 액세스 키를 비활성화하려면 **Make Inactive**를 선택하십시오. 확인 메시지가 표시되면 **Deactivate**를 선택하십시오. 비활성화된 액세스 키도 두 개의 액세스 키 제한에 포함됩니다.
 - 액세스 키를 활성화하려면 **Make Active**를 선택하십시오.
 - 더 이상 필요하지 않을 때 액세스 키를 삭제하려면, 액세스 키 ID를 복사한 다음 **Delete**를 선택하십시오. 액세스 키를 삭제하기 전에 **Deactivate**를 선택해야 합니다. 영구적으로 삭제하기 전에 액세스 키가 더 이상 사용되지 않는지 확인하는 것이 좋습니다. 삭제를 확인하려면 텍스트 입력 필드에 액세스 키 ID를 붙여넣은 다음 **Delete**를 선택하십시오.

IAM 사용자로 로그인한 상태에서 액세스 키를 관리하려면

1. IAM 사용자로 AWS Management Console에 로그인하십시오.
2. 오른쪽 상단의 탐색 모음에서 사용자 이름을 선택한 다음 **My Security Credentials**를 선택하십시오.
3. 다음 중 하나를 수행하십시오:
 - access key를 생성하려면 **Create access key**를 선택하십시오. 이미 두 개의 access key가 있으면 이 버튼은 비활성화되며 새 키를 생성하기 전에 access key를 삭제해야 합니다. 메시지가 표시되면 **Show secret access key** 또는 **Download .csv file**을 선택하십시오. 이것은 secret access key를 저장할 수 있는 유일한 기회입니다. secret access key를 안전한 위치에 저장한 후 **Close**를 선택하십시오.
 - access key를 비활성화하려면 **Make inactive**를 선택하십시오. 확인 메시지가 표시되면 **Deactivate**를 선택하십시오. 비활성화된 access key는 여전히 두 개의 access key 한도에 포함됩니다.

- access key를 활성화하려면 **Make active**를 선택하십시오. 확인 메시지가 표시되면 **Make active**를 선택하십시오.
- 더 이상 필요하지 않을 때 access key를 삭제하려면 access key ID를 복사한 다음 **Delete**를 선택하십시오. 이렇게 하면 access key가 비활성화됩니다. 영구적으로 삭제하기 전에 access key가 더 이상 사용되지 않는지 확인하는 것이 좋습니다. 삭제를 확인하려면 텍스트 입력 필드에 access key ID를 붙여넣은 다음 **Delete**를 선택하십시오.

sgcOpenAPI Configuration

자체 AWS Access Key를 보유하면, Amazon AWS 서버에 요청을 하기 전에 OpenAPI Amazon Client에서 이를 구성해야 합니다.

```
GetOpenAPIClient->AmazonOptions->AccessKey = "AKIAIOSFODNN7EXAMPLE";
GetOpenAPIClient->AmazonOptions->SecretKey = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY";
```

OpenAPI Amazon AWS | S3

Amazon Simple Storage Service(Amazon S3)는 업계 최고 수준의 확장성, 데이터 가용성, 보안, 성능을 제공하는 객체 스토리지 서비스입니다. 모든 규모와 업종의 고객은 Amazon S3를 사용하여 데이터 레이크, 웹사이트, 모바일 애플리케이션, 백업 및 복원, 아카이브, 엔터프라이즈 애플리케이션, IoT 기기, 빅 데이터 분석과 같은 다양한 사용 사례를 위해 원하는 양의 데이터를 저장하고 보호할 수 있습니다. Amazon S3는 특정 비즈니스, 조직, 규정 준수 요구 사항을 충족하도록 데이터에 대한 액세스를 최적화, 구성, 구성할 수 있는 관리 기능을 제공합니다.

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

sgcOpenAPI Google Client(TsgcOpenAPI_Google_Client)는 [TsgcOpenAPI_Client](#)로부터 상속하는 자체 OpenAPI Client를 가지고 있습니다.

이 구성 요소에는 Google Cloud Servers에 연결하는 데 필요한 모든 구성을 포함하는 **GoogleOptions** 라는 속성이 있습니다.

GoogleOptions

OpenAPI Google 클라이언트는 다음 방법을 사용하여 인증할 수 있습니다:

1. **OAuth2 Code**: 대화형이며, 이는 사용자의 개입이 필요함을 의미합니다.
2. **JWT (서비스 계정)**: 비대화형이므로 예를 들어 서비스로 실행될 수 있습니다.

인증은 GoogleOptions.Authentication 속성에서 구성되며, 다음 값을 허용합니다:

- **oagaOAuth2**: 대화형입니다.
- **oagaJWT**: 비대화형. **LoadSettingsFromFile** 메서드를 사용하여 JSON 파일에서 설정을 가져올 수 있습니다. 이 메서드는 다음 속성을 자동으로 채웁니다:
 - **ClientEmail**
 - **PrivateKeyId**
 - **PrivateKey**
 - **ServiceAccountOptions**: 서비스 계정을 사용하여 클라이언트를 실행할 때 다음 속성이 필요합니다:
 - **TokenURI**: 기본값은 "https://oauth2.googleapis.com/token"이지만, LoadSettingsFromFile 메서드를 사용하면 값이 업데이트됩니다.
 - **Subject**: Domain-Wide 위임을 사용할 때의 이메일 계정입니다
 - **Scopes**: 범위 목록.

Most common uses

- 구성
 - [Google Cloud OAuth2](#)
 - [Google Cloud Service Accounts](#)
- APIs
 - [Google Cloud PubSub](#)
 - [Google Cloud Calendar](#)
 - [Google Cloud Drive](#)

sgcOpenAPI Google Cloud SDK

아래에서 현재 사용 가능한 API 목록을 찾을 수 있습니다.

- Abusive Experience Report API
- Accelerated Mobile Pages (AMP) URL API
- 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
- Analytics Reporting API

- 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
- Cloud Bigtable Admin API
- Cloud Billing API
- Cloud Billing Budget API
- Cloud Build API
- Cloud Channel API
- Cloud Composer API
- 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
- Cloud OS Login API
- 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
- Cloud Text-to-Speech API
- 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
- Dialogflow API
- 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
- Google Identity Toolkit API
- 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
- Idea Hub API
- Identity and Access Management (IAM) API
- Indexing API
- Knowledge Graph Search API
- Kubernetes Engine API
- Library Agent API
- Local Services API
- Managed Service for Microsoft Active Directory API
- Manufacturer Center API
- My Business Account Management API
- My Business Business Calls API
- My Business Business Information API
- 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
- Network Services API
- 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
- Remote Build Execution API
- Replica Pool
- Resource Settings API
- Retail API
- Safe Browsing API
- 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
- Workflows API
- YouTube Analytics API
- YouTube Data API v3
- YouTube Reporting API

OpenAPI Google Cloud | OAuth2

OpenAPI Google Cloud 구성 요소를 사용하고 OAuth2를 사용하여 인증하려면, 먼저 Google Cloud에서 OAuth2 Key를 얻어야 합니다.

Google OAuth2 Key를 얻고 PubSub 샘플 애플리케이션에서 구성하는 단계는 아래에서 찾을 수 있습니다.

먼저 **Google Cloud** 계정에 로그인하고 기존 프로젝트를 사용하거나 새 프로젝트를 생성하십시오.

그 후, **Credentials** 메뉴로 이동하여 **CREATE CREDENTIALS** 버튼을 누르고, **OAuth Client ID** 옵션을 선택하십시오.

애플리케이션 유형을 선택하고 설명 이름을 설정하십시오

성공하면 **Client Id**와 **Client Secret**을 얻게 됩니다.

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

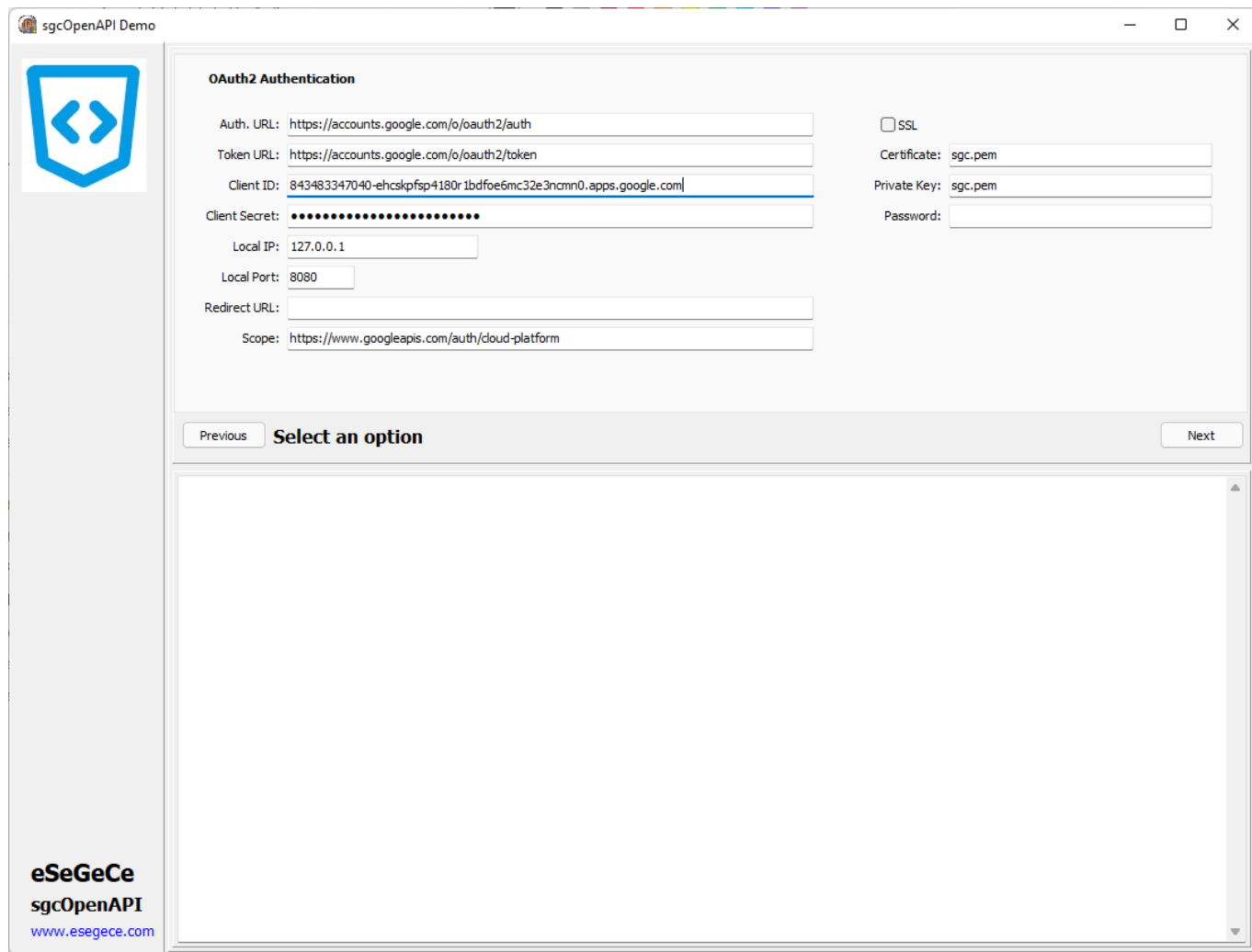
pvogD9reE0t9i1L6eR1jE60Z



OK

OAuth2 데이터를 누구와도 공유하지 마십시오!

이제 OpenAPI Google Cloud 샘플에 복사하십시오



The image shows a window titled "sgcOpenAPI Demo" with a sidebar on the left containing a logo and the text "eSeGeCe sgcOpenAPI www.esegece.com". The main area is titled "OAuth2 Authentication" and contains several input fields and checkboxes.

OAuth2 Authentication

Auth. URL:

Token URL:

Client ID:

Client Secret:

Local IP:

Local Port:

Redirect URL:

Scope:

☐ SSL

Certificate:

Private Key:

Password:

Navigation buttons: Previous, **Select an option**, Next

[OAuth2 Configuration](#)에 대해 자세히 알아보십시오.

인증되면, 먼저 **ClearOAuth2Token** 메서드를 호출하여(모든 내부 OAuth2 Token 지우기) 그런 다음 OpenAPI 요청을 호출하여 **재인증**할 수 있습니다. google 서버에 대해 재인증하기 위해 새 웹 브라우저가 표시됩니다.

OpenAPI Google Cloud | Service Accounts

OpenAPI Google Cloud 구성 요소를 사용하고 Service Accounts를 사용하여 인증하려면, 먼저 Google Cloud에서 Private Key Certificate를 얻어야 합니다.

Google Private Key Certificate를 얻는 단계와 샘플 애플리케이션에서 구성하는 방법을 아래에서 확인하십시오.

먼저 Google Cloud 계정에 로그인하고 기존 프로젝트를 사용하거나 새 프로젝트를 생성하십시오.

The screenshot shows the 'IAM & Admin' section of the Google Cloud console. The 'Service accounts' tab is selected, displaying a list of service accounts for the project 'PubSub'. The page includes a '+ CREATE SERVICE ACCOUNT' button and a 'DELETE' button. Below the header, there is a section titled 'Service accounts for project "PubSub"' with explanatory text about service accounts and organization policies. A table with columns 'Email', 'Status', 'Name', 'Description', 'Key ID', 'Key creation date', and 'Actions' is shown, but it contains no rows, indicating no service accounts are currently listed.

CREATE SERVICE ACCOUNT를 선택하면 서비스 계정 이름과 설명을 설정해야 하는 새 페이지가 표시됩니다

The screenshot shows the 'Create service account' page in the Google Cloud IAM & Admin console. The page is titled '1 Service account details' and contains three main input fields: 'Service account name' (filled with 'sgcServiceAccount'), 'Service account ID' (filled with 'sgcServiceAccount' and a domain '@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.

그런 다음 최소 하나의 Role을 선택하십시오. 클라이언트가 topic을 게시하고 구독할 수 있도록 PubSub Admin을 선택했지만, 더 적은 권한을 가진 다른 role을 선택할 수도 있습니다

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](#)

CONTINUE를 누르면 마지막으로 다른 사용자에게 접근 권한을 부여할 수 있습니다

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](#)

완료되면 DONE을 누르면 새 레코드가 표시됩니다.

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

다음 단계는 새 Key를 만드는 것입니다. actions 열에서 Create Key 옵션을 선택하십시오. JSON을 선택하여 구성을 JSON 형식으로 다운로드하면 새 Key가 생성됩니다

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	⋮

마지막으로 OpenAPI PubSub 클라이언트에 google이 제공한 데이터를 채우기만 하면 됩니다. 구성 JSON 파일을 로드하려면 `LoadSettingsFromFile`을 사용할 수 있습니다.

Domain-Wide Delegation

Google Workspace 계정이 있는 경우, 조직의 관리자는 애플리케이션이 Google Workspace 도메인의 사용자를 대신하여 사용자 데이터에 액세스하도록 승인할 수 있습니다. 예를 들어, **Google Calendar API**를 사용하여 Google Workspace 도메인의 모든 사용자 캘린더에 이벤트를 추가하는 애플리케이션은 service account를 사용하여 사용자를 대신하여 Google Calendar API에 액세스합니다. service account가 도메인의 사용자를 대신하여 데이터에 액세스하도록 승인하는 것을 때때로 service account에 "domain-wide authority 위임"이라고 합니다.

서비스 계정에 도메인 전체 권한을 위임하려면 Google Workspace 도메인의 슈퍼 관리자가 다음 단계를 완료해야 합니다:

- Google Workspace 도메인의 관리 콘솔에서 **기본 메뉴 menu > 보안 > 액세스 및 데이터 제어 > API 제어**로 이동하십시오.
- **Domain wide delegation** 창에서 **Manage Domain Wide Delegation**을 선택하십시오.
- **새로 추가**를 클릭하십시오.
- **Client ID** 필드에 서비스 계정의 **Client ID**를 입력하십시오. 서비스 계정의 client ID는 Service accounts 페이지에서 찾을 수 있습니다.
- **OAuth scopes (선택 구분)** 필드에 애플리케이션이 액세스 권한을 부여받아야 하는 scope 목록을 입력하십시오. 예를 들어, 애플리케이션이 Google Drive API와 Google Calendar API에 대한 도메인 전체 전체 액세스가 필요한 경우, <https://www.googleapis.com/auth/drive>, <https://www.googleapis.com/auth/calendar>를 입력하십시오.
- **Authorize**를 클릭하십시오.

워크스페이스 계정을 연결하고 권한을 부여한 후, OpenAPI 클라이언트에서 GoogleOptions.ServiceAccountOptions 속성을 구성하십시오:

- **Subject:** 서비스 계정에 연결된 워크스페이스 이메일 계정입니다. 예: youremail@domain.com
- **Scopes:** scope 목록입니다. 예제: <https://www.googleapis.com/auth/calendar>
- **TokenURI:** 기본값은 <https://oauth2.googleapis.com/token>입니다.

OpenAPI Google Cloud | PubSub

Pub/Sub은 서비스가 약 100밀리초 정도의 지연 시간으로 비동기적으로 통신할 수 있게 합니다.

Pub/Sub는 데이터를 수집하고 배포하기 위해 스트리밍 분석 및 데이터 통합 파이프라인에 사용됩니다. 서비스 통합을 위한 메시징 지향 미들웨어로서 또는 작업을 병렬화하기 위한 큐로서도 똑같이 효과적입니다.

토픽별 프로젝트 목록(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");
```

토픽별 프로젝트 나열(Service Accounts)

```
GetOpenAPIClient->GoogleOptions->Authentication = oagaJWT;
GetOpenAPIClient->LoadSettingsFromFile("google.json");
GetOpenAPIClient->ListV1TopicsByProject("projects/pubsub-270909");
```

OpenAPI Google Cloud Calendar

Calendar API를 사용하면 앱을 Google Calendar와 통합하여 사용자와 소통하는 새로운 방법을 만들 수 있습니다.

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.events";
GetOpenAPIClient->ListCalendarsEventsByCalendarId("email@mydomain.com", true, "json", 0, "");
```

Insert Calendar Event

```
GetOpenAPIClient->InsertCalendarsEventsByCalendarId("email@mydomain.com", "{ \"summary\": \"Test Event\", \"description\": \"\" }", true, "json", 0, "");
```

OpenAPI Google Cloud | Drive

Google Drive 계정에서 파일을 업로드 및/또는 다운로드하려면 sgcOpenAPI_googleapis.com_drive_helpers 유닛을 사용하십시오.

Upload Files

UploadFile 메서드를 사용하여 google drive 계정에 새 파일을 업로드하십시오. **OnUpload** Event를 사용하면 업로드의 진행 상황을 알 수 있습니다.

```
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

Google Drive 계정에서 파일을 다운로드하려면 **DownloadFile** 메서드를 사용하십시오. **OnDownload** 이벤트를 통해 다운로드 진행 상황을 알 수 있습니다.

```
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();
}

```

폴더 나열

drive 계정의 모든 폴더를 나열합니다. **ListFiles** 메서드를 사용하고 매개변수 q를 "application/vnd.google-apps.folder"로 전달하십시오.

```

GetOpenAPIClient().ListFiles("", true, "", false, false, 100, "", false, false, "", "", "", "", "", "", "", "", ""

```

OpenAPI | Microsoft

sgcOpenAPI Microsoft Client(TsgcOpenAPI_Microsoft_Client)는 [TsgcOpenAPI_Client](#)에서 상속하는 자체 OpenAPI Client를 가집니다.

이 구성 요소에는 Microsoft 서버에 연결하는 데 필요한 모든 구성을 포함하는 **MicrosoftOptions** 속성이 있습니다.

MicrosoftOptions

OpenAPI Microsoft 클라이언트는 다음 방법을 사용하여 인증할 수 있습니다:

1. **OAuth2 Code**: 대화형이며, 이는 사용자의 개입이 필요함을 의미합니다.
2. **OAuth2 Credentials**: 비대화형이므로 예를 들어 서비스로 실행할 수 있습니다.

인증은 MicrosoftOptions.Authentication 속성에서 구성되며, 다음 값을 허용합니다:

- **oamaOAuth2Code**: 대화형입니다.
- **oamaOAuth2Credentials**: 비대화형.

Microsoft가 요구하는 다른 속성은 다음과 같습니다:

- **TenantId**: 계정을 식별하는 값으로, Microsoft/Azure 계정에서 찾을 수 있습니다.

Most common uses

- **Configuration**
 - [Microsoft Get Tenant](#)
 - [Microsoft Register Application](#)
 - [Microsoft OAuth2 Code](#)
 - [Microsoft OAuth2 Credentials](#)
- **APIs**
 - [Microsoft Graph](#)

sgcOpenAPI Microsoft APIs

아래에서 현재 사용 가능한 API 목록을 찾을 수 있습니다.

- AutoSuggest Client
- Computer Vision Client
- Custom Image Search Client
- Custom Search Client
- Custom Vision Prediction Client
- Custom Vision Training Client
- Entity Search Client
- Image Search Client
- Local Search 클라이언트
- News Search Client
- Partial Graph API
- Spell Check Client
- Video Search Client
- Visual Search Client
- Web Search Client

sgcOpenAPI Azure APIs

아래에서 현재 사용 가능한 API 목록을 찾을 수 있습니다.

- API Client
- ACE Provisioning ManagementPartner
- ADHybridHealthService
- AdvisorManagementClient
- Anomaly Detector Client
- Anomaly Finder 클라이언트
- 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
- Azure ML Commitment Plans Management Client
- 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
- Azure SQL Database replication links
- 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
- Database Threat Detection Policy APIs
- 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
- Domains API Client
- 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
- Ink Recognizer Client
- 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
- Provider API Client
- QnAMaker Client
- QnAMaker Runtime Client
- Recommendations API Client
- 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
- 서비스 맵
- ServiceBusManagementClient
- ServiceFabricManagementClient
- SharedImageGalleryServiceClient
- SignalRManagementClient
- SiteRecoveryManagementClient
- Software Plan RP
- SqlManagementClient
- SqlVirtualMachineManagementClient
- 스토리지 캐시 관리 클라이언트
- 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

ID 및 액세스 관리를 위해 Microsoft identity platform을 사용하는 앱을 빌드하려면, Azure Active Directory(Azure AD) *tenant*에 대한 액세스가 필요합니다. 앱을 등록하고 관리하며, Microsoft 365 및 기타 웹 API의 데이터에 대한 액세스를 구성하고, Conditional Access 같은 기능을 활성화하는 곳이 Azure AD tenant입니다.

테넌트는 조직을 나타냅니다. 이는 조직이나 앱 개발자가 Microsoft와의 관계 시작 시 받는 Azure AD의 전용 인스턴스입니다. 그 관계는 예를 들어 Azure, Microsoft Intune 또는 Microsoft 365에 가입하는 것으로 시작될 수 있습니다.

각 Azure AD tenant는 다른 Azure AD tenant와 구별되며 분리되어 있습니다. 자체적인 직장 및 학교 ID, 소비자 ID(Azure AD B2C tenant인 경우), app 등록 표현을 가지고 있습니다. tenant 내부의 app 등록은 tenant 내 계정에서만 또는 모든 tenant에서 인증을 허용할 수 있습니다.

기존 Azure AD 테넌트 사용

tenant를 확인하려면:

1. Azure Portal에 로그인하십시오. 애플리케이션을 관리하는 데 사용할 계정을 사용하십시오.
2. 오른쪽 상단 모서리를 확인하십시오. 테넌트가 있는 경우 자동으로 로그인됩니다. 계정 이름 바로 아래에 테넌트 이름이 표시됩니다.

계정에 연결된 테넌트가 없는 경우, 계정 이름 아래에 GUID가 표시됩니다. Azure AD 테넌트를 생성할 때까지 앱 등록과 같은 작업을 수행할 수 없습니다.

새 Azure AD 테넌트 생성

새 테넌트를 만들기 위해 다음 정보를 제공해야 합니다:

- **Organization name**
- **초기 도메인** - 초기 도메인 <domainname>.onmicrosoft.com 은 편집하거나 삭제할 수 없습니다. 나중에 사용자 정의 도메인 이름을 추가할 수 있습니다.
- **국가 또는 지역**

OpenAPI Microsoft | 애플리케이션 등록

애플리케이션을 등록하면 앱과 Microsoft ID 플랫폼 간에 신뢰 관계가 설정됩니다. 신뢰는 단방향입니다: 앱은 Microsoft ID 플랫폼을 신뢰하지만 그 반대는 아닙니다.

앱 등록을 생성하려면 다음 단계를 따르십시오:

1. Azure Portal에 로그인하십시오.
2. 여러 테넌트에 액세스할 수 있는 경우, 상단 메뉴의 **Directories + subscriptions** 필터를 사용하여 애플리케이션을 등록하려는 테넌트로 전환하십시오.
3. **Azure Active Directory**를 검색하여 선택하십시오.
4. **Manage**에서 **App registrations > New registration**을 선택하십시오.
5. 애플리케이션의 표시 **Name**을 입력하십시오. 애플리케이션 사용자는 앱을 사용할 때(예: 로그인 중) 표시 이름을 볼 수 있습니다. 표시 이름은 언제든지 변경할 수 있으며 여러 앱 등록이 동일한 이름을 공유할 수 있습니다. 앱 등록의 자동 생성된 Application(client) ID는 표시 이름이 아니라 identity platform 내에서 앱을 고유하게 식별합니다.
6. 누가 애플리케이션을 사용할 수 있는지 지정하십시오. 때로는 *로그인 대상*이라고도 합니다.
7. 선택 **Register** 초기 앱 등록을 완료하려면

Register an application - Microsoft x +

← → ↺ 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

리디렉션 URI 추가

A 리디렉션 URI 는 인증 후 Microsoft identity platform이 사용자의 클라이언트를 리디렉션하고 보안 토큰을 보내는 위치입니다.

예를 들어 프로덕션 웹 애플리케이션에서 리디렉션 URI는 종종 앱이 실행 중인 공개 엔드포인트입니다. 예: `https://contoso.com/auth-response`. 개발 중에는 앱을 로컬에서 실행하는 엔드포인트를 추가하는 것도 일반적입니다. 예: `https://127.0.0.1/auth-response` 또는 `http://localhost/auth-response`.

이 **RedirectURI** 는 나중에 **sgcOpenAPI Microsoft Client**를 구성하는 데 사용됩니다.

자격 증명 추가

자격 증명은 웹 API에 액세스하는 기밀 클라이언트 애플리케이션에서 사용됩니다. 기밀 클라이언트의 예로는 웹 앱, 다른 웹 API, 서비스 유형 및 데몬 유형 애플리케이션이 있습니다. 자격 증명을 통해 애플리케이션이 자신으로 인증할 수 있으며 런타임에 사용자의 상호작용이 필요하지 않습니다.

기밀 클라이언트 앱 등록에 인증서와 클라이언트 비밀(문자열)을 모두 자격 증명으로 추가할 수 있습니다.

The screenshot shows the Azure portal interface for 'Contoso App 1'. The main heading is 'Contoso App 1 | Certificates & secrets'. Below the heading, there's a search bar and a 'Got feedback?' link. The left sidebar contains a navigation menu with sections: 'Manage' (Overview, Quickstart, Integration assistant, Branding, Authentication, **Certificates & secrets**, Token configuration, API permissions, Expose an API, App roles, Owners, Roles and administrators | Preview, Manifest) and 'Support + Troubleshooting' (Troubleshooting, New support request). The main content area shows a message: 'Application registration certificates, secrets and federated credentials can be found in the tabs below.' Below this, there are three tabs: 'Certificates (0)', 'Client secrets (0)', and 'Federated credentials (0)'. The 'Client secrets (0)' tab is selected. It contains a description: 'A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.' and a '+ New client secret' button. Below the button is a table with columns: 'Description', 'Expires', 'Value', and 'Secret ID'. The table is empty, with a message below it: 'No client secrets have been created for this application.'

client secret 추가

때때로 *application password*라고도 하는 client secret은 앱이 자신을 식별하기 위해 인증서 대신 사용할 수 있는 문자열 값입니다.

클라이언트 시크릿은 인증서 자격 증명보다 덜 안전한 것으로 간주됩니다. 애플리케이션 개발자는 사용 편의성 때문에 로컬 앱 개발 중에 클라이언트 시크릿을 사용하기도 합니다. 그러나 프로덕션에서 실행되는 애플리케이션에는 인증서 자격 증명을 사용해야 합니다.

1. Azure portal의 **App registrations**에서 애플리케이션을 선택하십시오.
2. **Certificates & secrets > Client secrets > New client secret** 을 선택하십시오.
3. 클라이언트 시크릿에 대한 설명을 추가하십시오.
4. 시크릿의 만료를 선택하거나 사용자 정의 수명을 지정하십시오.
 - 클라이언트 시크릿 수명은 2년(24개월) 이하로 제한됩니다. 24개월보다 긴 사용자 정의 수명은 지정할 수 없습니다.
 - Microsoft는 만료 값을 12개월 미만으로 설정할 것을 권장합니다.
5. Select **Add**.
6. *시크릿의 값을 기록* 하여 클라이언트 애플리케이션 코드에서 사용하십시오. 이 시크릿 값은 이 페이지를 떠난 후 *다시 표시되지 않습니다*.

OpenAPI Microsoft | OAuth2 Code

OAuth2 Code Grant Flow를 사용하려면 사용자가 로그인하여 필요한 권한을 얻기 위해 사용자와의 상호 작용이 필요합니다.

Tenant Id와 Credentials가 있으면 Code Grant에 대한 OAuth2 속성을 구성할 수 있습니다.

OAuth2 Code Grant용 OpenAPI Client를 구성하려면 **MicrosoftOptions.Authentication** 속성을 다음 값으로 구성하십시오:

```
GetOpenAPIClient->MicrosoftOptions->Authentication = oamaOAuth2Code;
```

그런 다음 OAuth2 매개변수를 구성할 수 있습니다.

The screenshot shows the 'sgcOpenAPI Demo' application window. 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 'OAuth2 Authentication'. It contains the following fields:

- Auth. URL:
- Token URL:
- Client ID:
- Client Secret:
- Local IP:
- Local Port:
- Redirect URL:
- Scope:
- SSL: ☒
- Certificate:
- Private Key:
- Password:

At the bottom, there are 'Previous' and 'Next' buttons, and a 'Select an option' button in the center.

```
GetOpenAPIClient->MicrosoftOptions->Authentication = oamaOAuth2Code;
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 = "
openid
";

```

Authentication 및 Token URL에 대해 Tenant ID를 구성해야 합니다. url에서 올바른 Tenant Id로 교체하기만 하면 됩니다.

Microsoft는 Local IP에서 수신 중인 경우 URL **localhost**만 허용하므로, IP 주소로 구성하는 대신 dns 이름으로 localhost를 사용하여 **redirect URL**을 설정하십시오.

scope 값은 API에 따라 다릅니다. 모든 API에 대해 Microsoft / Azure 문서를 확인하십시오.

요청이 **처음** 이루어질 때, 사용자가 Microsoft 계정에 로그인하도록 요청하는 웹 브라우저를 표시합니다. 사용자가 이전에 이미 로그인한 경우, HTTP 요청을 직접 수행합니다.

OpenAPI Microsoft | OAuth2 Credentials

OAuth2 Code Grant Flow를 사용하면 사용자와의 상호 작용이 필요하지 않으므로, 서비스, 데몬... 또는 사용자 상호 작용 없이 실행되어야 하는 모든 애플리케이션에 적합합니다.

Tenant Id와 Credentials가 있으면 Code Grant에 대한 OAuth2 속성을 구성할 수 있습니다.

OAuth2 Code Grant용 OpenAPI Client를 구성하려면 **MicrosoftOptions.Authentication** 속성을 다음 값으로 구성하십시오:

```
GetOpenAPIClient->MicrosoftOptions->Authentication = oamaOAuth2Credentials;
```

```
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 = "openid";
```

```
https://graph.microsoft.com/.default
```

```
";
```

Authentication 및 Token URL에 대해 Tenant ID를 구성해야 합니다. url에서 올바른 Tenant Id로 교체하기만 하면 됩니다.

Microsoft는 Local IP에서 수신 중인 경우 URL **localhost**만 허용하므로, IP 주소로 구성하는 대신 dns 이름으로 localhost를 사용하여 **redirect URL**을 설정하십시오.

scope 값은 API에 따라 다릅니다. 모든 API에 대해 Microsoft / Azure 문서를 확인하십시오.

OAuth2 자격 증명은 사용자 상호작용을 요구하지 않으므로, 첫 HTTP 요청 호출에서 브라우저가 열리지 않습니다.

OpenAPI Microsoft | Graph

Microsoft Graph는 다음 Microsoft 클라우드 서비스의 데이터에 액세스하기 위한 REST API와 클라이언트 라이브러리를 노출합니다:

- Microsoft 365 핵심 서비스: Bookings, Calendar, Delve, Excel, Microsoft 365 compliance eDiscovery, Microsoft Search, OneDrive, OneNote, Outlook/Exchange, People(Outlook 연락처), Planner, SharePoint, Teams, To Do, Workplace Analytics
- Enterprise Mobility + Security 서비스: Advanced Threat Analytics, Advanced Threat Protection, Azure Active Directory, Identity Manager 및 Intune
- Windows 서비스: 활동, 장치, 알림, Universal Print
- Dynamics 365 Business Central

Get Current User

```

GetOpenAPIClient->MicrosoftOptions->Authentication = oamaOAuth2Code;
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->AuthURL = "https://login.microsoftonline.co
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->TokenURL = "https://login.microsoftonline.c
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

다음 API는 eSeGeCe sgcOpenAPI Generator를 사용하여 생성되었으며 모든 사용자에게 무료로 제공됩니다. 인터페이스의 소스 코드가 트라이얼과 함께 제공되므로, Google, Amazon 또는 Microsoft와 같은 비공개 API의 라이선스를 구매하면 무엇을 기대할 수 있는지 볼 수 있습니다.

API	설명
Geolocation	Abstract IP Geolocation API는 IP 주소를 받아서 위치뿐만 아니라 주소, 시간대 등과 같은

OpenAPI | AbstractApi Geolocation

IP Geolocation API란 무엇입니까?

Abstract IP Geolocation API는 IP 주소를 받아 위치로 변환하고, 주소, 시간대 등 다른 많은 세부 정보로도 변환합니다.

IP Geolocation API의 사용 사례는 무엇입니까?

IP 지오로케이션 API와 데이터에는 많은 강력한 사용 사례가 있습니다. 여기에는 다음이 포함되지만 이에 국한되지 않습니다:

- 위치에 따라 사용자를 관련 사이트 또는 하위 사이트로 자동 리디렉션합니다
- 사용자의 위치, 국가 또는 시간대를 명시적으로 사용자 지정하도록 요구하지 않고 자동으로 감지하고 표시합니다
- 사용자의 위치를 기반으로 웹사이트나 앱의 콘텐츠 또는 경험을 사용자 정의합니다. 예: 사용자의 현지 날씨, 세금 및 VAT 요율, 통화, 뉴스, 공휴일 등을 표시합니다.
- 위치를 기반으로 사용자를 필터링합니다. 예를 들어 특정 국가의 사용자에게 서비스를 제공할 수 없는 경우입니다.
- 유럽 연합 시민을 위한 GDPR 쿠키 배너와 같이 현지 규정에서 요구하는 특정 약관을 사용자가 수락하도록 요구합니다

API Key는 어디서 얻습니까?

abstractapi.com에 등록하기만 하면 무료로 api 키를 얻을 수 있습니다.

<https://www.abstractapi.com/api/ip-geolocation-api>

샘플 코드

다음 코드는 제공된 IP Address에 대한 정보를 반환합니다

```
ShowMessage(GetOpenAPIClient->Retrieve_the_location_of_an_IP_address("asdfkjlkj32i3j2liwj3es", "88.5.12.4"));
```

Demos | Server Chat

이 데모는 [TsgcWebSocketHTTPServer](#)와 통신 프로토콜로 WebSockets를 사용하여 Server Chat을 구축하는 방법을 보여 줍니다.

새 피어가 메시지를 전송할 때마다 서버는 메시지를 읽고 연결된 모든 클라이언트에 메시지를 브로드캐스트합니다.

Start Server

서버를 시작하기 전에 수신 포트, 보안 연결 사용 여부 등을 설정하도록 구성해야 합니다...

- 먼저 [TsgcWebSocketHTTPServer](#)의 새 인스턴스를 생성합니다.
- 서버가 보안 연결을 사용하는 경우, PEM 인증서가 필요합니다. 이 인증서의 위치와 SSL용 수신 대기 포트를 설정하기만 하면 됩니다. TLS 버전과 OpenSSL API(필요한 경우)를 구성할 수 있습니다

```
// ... 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;
}
```

- 기본적으로 서버를 시작하면, 수신 대기 포트의 모든 IP에서 수신 대기하므로, binding 속성을 사용하여 1개의 특정 IP에서만 수신 대기하는 것이 권장됩니다.

```
With WSServer->Bindings->Add do
{
    Port = StrToInt(txtDefaultPort->Text);
    IP = txtHost->Text;
}
```

- 모든 옵션을 구성한 후 서버를 시작하려면 `Server.Active = true`를 호출하십시오.

Events Configuration

- 클라이언트가 서버에 연결될 때를 알기 위해 **OnConnect** 및 **OnDisconnect** 이벤트를 사용하십시오.
- 클라이언트에서 서버로 보낸 메시지는 **OnMessage** 이벤트로 수신되므로, 이 이벤트 핸들러를 사용하여 수신된 메시지를 모든 클라이언트에게 브로드캐스트하십시오


```
private void OnMessageEvent(TsgcWSConnection *Connection, string Text)
{
    server->Broadcast(Text);
}
```

HTTP 요청 디스패치

WebSocket HTTP Server를 사용하면 동일한 수신 포트에서 WebSocket과 HTTP 프로토콜을 처리할 수 있으므로, 웹 브라우저가 웹 페이지를 요청하여 서버에 액세스할 수 있습니다. **OnCommandGet**은 HTTP Request를 읽고 HTTP Response를 보내는 데 사용되는 이벤트입니다.

ARequestInfo 매개변수를 사용하여 HTTP 요청을 읽고 AResponseInfo를 사용하여 HTTP 응답을 작성하십시오.

기본적으로, ARequestInfo.Document를 사용하여 클라이언트가 어떤 문서를 요청하는지 읽고 다음 속성을 사용하여 응답을 보내십시오: ResponseNo, ContentType 및 ContentText.

예제: 클라이언트가 문서 '/jquery.js'를 요청합니다

```
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

이 데모는 WebSocket Server에 연결하여 메시지를 전송하고 이 메시지가 연결된 모든 클라이언트에 수신되는 클라이언트 채팅을 [TsgcWebSocketClient](#)를 사용하여 빌드하는 방법을 보여줍니다.

서버에 연결

- 먼저 [TsgcWebSocketClient](#)의 새 인스턴스를 생성하십시오.
- 그런 다음 서버 **Host** 및 **Port**를 구성하십시오.
- 클라이언트가 보안 연결을 사용하는 경우, 구성 요소의 **TLSOptions** 속성을 구성하십시오.

```
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;
```

- 모든 옵션을 구성한 후 서버에 연결하려면 `Client.Active = true`로 설정하십시오.

서버에 메시지 보내기

- 서버에 메시지를 보내려면 **WriteData** 메서드를 사용하여 텍스트 메시지를 보내면 서버가 동일한 메시지를 응답으로 보냅니다.

```
WSClient->WriteData(txtName->Text + ": " + txtMessage->Text);
```

Receive Messages from Server

- 클라이언트가 새 Text 메시지를 수신할 때마다, **OnMessage** 이벤트가 발생합니다.

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

Demos | Client

이 데모는 [TsgcWebSocketClient](#)를 사용하여 websocket 클라이언트를 구축하는 방법을 보여줍니다.

서버에 연결

- 먼저 [TsgcWebSocketClient](#)의 새 인스턴스를 생성하십시오.
- 그런 다음 서버 **Host** 및 **Port**를 구성하십시오.
- 기본적으로 클라이언트는 **WebSocket 프로토콜**을 사용하여 연결됩니다. 그러나 일반 **TCP 프로토콜**을 사용하여 연결하도록 클라이언트를 구성할 수 있습니다. **Specifications.RFC6455 = false**로 설정하기만 하면 클라이언트는 WebSocket 프로토콜 대신 일반 TCP 프로토콜을 사용합니다. [TCP 연결](#)에 대해 자세히 알아볼 수 있습니다.

```
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

다음 이벤트를 사용하여 클라이언트 흐름을 제어하십시오: 연결될 때, 연결이 해제될 때, 메시지를 수신할 때, 오류가 감지될 때...

```
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 | Client MQTT

이 데모는 MQTT 브로커 서버에 연결하는 방법을 보여 줍니다. WebSocket / TCP 프로토콜을 처리하려면 [TsgcWebSocketClient](#)가 필요합니다.

구성

- 먼저 새 [TsgcWebSocketClient](#) 인스턴스를 생성하고, [Client 데모](#)를 확인하십시오.
- 그런 다음 [TsgcWSPClient MQTT](#)의 새 인스턴스를 생성하십시오.
- 그런 다음 MQTT Protocol을 WebSocket 클라이언트에 할당하고 WebSocket 클라이언트에서 연결 옵션을 구성해야 합니다.

```
if (mqtt == null)
{
    mqtt = new TsgcWSPClient_MQTT();
    mqtt->OnMQTTBeforeConnect += OnMQTTBeforeConnectEvent;
    mqtt->OnMQTTConnect += OnMQTTConnectEvent;
    mqtt->OnMQTTDisconnect += OnMQTTDisconnectEvent;
    mqtt->OnMQTTSubscribe += OnMQTTSubscribeEvent;
    mqtt->OnMQTTUnSubscribe += OnMQTTUnSubscribeEvent;
    mqtt->OnMQTTPing += OnMQTTPingEvent;
    mqtt->OnMQTTPubAck += OnMQTTPubAckEvent;
    mqtt->OnMQTTPubComp += OnMQTTPubCompEvent;
    mqtt->OnMQTTPublish += OnMQTTPublishEvent;
    mqtt->OnMQTTPubRec += OnMQTTPubRecEvent;
    mqtt->Client = client;
}
mqtt->Client = client;
txtParameters->Text = "/";
chkTLS->Checked = false;
mqtt->Authentication->Enabled = false;
mqtt->Authentication->UserName = "";
mqtt->Authentication->Password = "";
mqtt->MQTTVersion = TwsMQTTVersion->mqtt311;
mqtt->HeartBeat->Interval = 5;
mqtt->HeartBeat->Enabled = true;
switch (Index)
{
    case 0: // esegece.com
        txtHost->Text = "www.esegece.com";
        txtPort->Text = "15675";
        txtParameters->Text = "/ws";
        mqtt->Authentication->Enabled = true;
        mqtt->Authentication->UserName = "sgc";
        mqtt->Authentication->Password = "sgc";
        chkOverWebSocket->Checked = true;
        break;
    case 1: // test.mosquitto.org
        txtHost->Text = "test.mosquitto.org";
        txtPort->Text = "1883";
        chkTLS->Checked = false;
        chkOverWebSocket->Checked = false;
        break;
    case 2: // mqtt.fluux.io
        txtHost->Text = "mqtt.fluux.io";
        txtPort->Text = "1883";
        chkTLS->Checked = false;
        chkOverWebSocket->Checked = false;
        mqtt->MQTTVersion = TwsMQTTVersion->mqtt5;
        break;
    case 3: // broker.hivemq.com
        txtHost->Text = "broker.mqttdashboard.com";
        txtPort->Text = "8000";
        txtParameters->Text = "/mqtt";
        chkTLS->Checked = false;
        chkOverWebSocket->Checked = true;
        mqtt->MQTTVersion = TwsMQTTVersion->mqtt5;
        break;
}
```

MQTT Events

연결 흐름은 MQTT Client 구성 요소에 의해 제어되므로, 브로커에 연결되었을 때, 새 메시지가 게시되었을 때, 연결이 해제되었을 때... 를 알기 위해 MQTT 이벤트를 처리해야 합니다.

```
private void OnMQTTConnectEvent(TsgcWSConnection *Connection, bool Session, int ReasonCode, string ReasonName, TsgcWSMQTTConnectProperty *Properties)
{
    DoLog("#MQTT Connect");
    chkTLS->Enabled = false;
    chkCompressed->Enabled = false;
    if (FMQTTSubscribeTopic != "")
    {
        mqtt->Subscribe(FMQTTSubscribeTopic);
        FMQTTSubscribeTopic = "";
    }
}

private void OnMQTTPublishEvent(TsgcWSConnection *Connection, string Topic, string Text, TsgcWSMQTTPublishProperty *Properties)
{
    DoLog(Topic + ": " + Text);
}

private void OnMQTTSubscribeEvent(TsgcWSConnection *Connection, int PacketIdentifier, TsgcWSSUBACKS *Codes, TsgcWSMQTTSubscribeProperty *Properties)
{
    DoLog("#Subscribe: " + IntToStr(PacketIdentifier));
}

private void OnMQTTDisconnectEvent(TsgcWSConnection *Connection, int ReasonCode, string ReasonName, TsgcWSMQTTDisconnectProperty *Properties)
{
    DoLog("#disconnected");
    chkTLS->Enabled = true;
    chkCompressed->Enabled = true;
}
```

Demos | Client SocketIO

이 데 모 는 Socket.IO Server 에 연 결 하 는 방 법 을 보 여 주 니 다. WebSocket / TCP 프 로 토 콜 을 처 리 하 려 면 [TsgcWebSocketClient](#)가 필요합니다.

구성

- 먼저 새 [TsgcWebSocketClient](#) 인스턴스를 생성하고, [Client 데모](#)를 확인하십시오.
- 그런 다음, [TsgcWSAPI_SocketIO](#)의 새 인스턴스를 생성하십시오.
- 그런 다음 Socket.IO API를 WebSocket 클라이언트에 할당하고 WebSocket 클라이언트에서 연결 옵션을 구성해야 합니다.

```
if (socketio == null)
{
    socketio = new TsgcWSAPI_SocketIO();
    socketio->Client = client;
}
socketio->Client = client;
txtParameters->Text = "/";
chkTLS->Checked = true;
chkOverWebSocket->Checked = true;
```

Send Messages

Socket.IO는 서버에 메시지를 전송하기 위해 **TsgcWebSocketClient**를 사용하므로, **WriteData**를 호출하고 매개변수로 socket.io 서버에 보낼 JSON 메시지를 전달하기만 하면 됩니다

```
client->WriteData("42[\"new message\", \"\" + txtSocketIOMessage.Text + "\"]");
```

메시지 수신

연결 흐름으로 수신된 메시지는 TsgcWebSocketClient에 의해 처리되므로, 이 구성 요소를 사용하여 서버에서 보낸 메시지를 읽고 연결이 활성 상태인지 여부를 알 수 있습니다.

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

Demos | Server Monitor

이 데모는 프로토콜로 WebSocket을 사용하여 3개의 HTML Monitor를 업데이트하는 방법을 보여줍니다. 서버에는 게이지 값을 무작위로 업데이트하고 websocket 메시지를 사용하여 값을 업데이트하는 내부 타이머가 있습니다. 이 메시지는 javascript 클라이언트가 읽고 Gauge 값을 업데이트합니다.

구성

- 먼저 새 [TsgcWebSocketServer](#) 인스턴스를 생성하고, [Server Chat Demo](#)를 확인하십시오.
- 그런 다음 새 Timer를 만들고 500밀리초마다 메모리, 네트워크 또는 cpu 값을 업데이트하십시오. 연결된 모든 클라이언트에 업데이트를 보내십시오.
- Javascript 클라이언트에서, 서버가 보낸 메시지를 읽고 게이지의 값을 업데이트하십시오.

```
<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 SockJS("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>

```


데모 | Server Snapshots

이 데모는 서버에서 클라이언트로 이미지를 보내는 방법과 서버 구성 요소의 broadcast 메서드를 사용하여 모든 클라이언트가 동일한 이미지를 수신하는 방법을 보여줍니다.

구성

- 먼저 새 [TsgcWebSocketHTTPServer](#) 인스턴스를 생성하고, [Server Chat Demo](#)를 확인하십시오.
- 메시지가 클라이언트로 전송될 때 더 적은 바이트를 전송하려면 압축을 활성화하십시오

```
Server.Extensions.PerMessage_Deflate.Enabled = true
```

- 그런 다음 서버는 5초마다 연결된 모든 클라이언트에게 이미지 스트림을 브로드캐스트합니다

```
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

이 데모는 WebSocket Server에 연결하여 사용자에게 표시되는 이미지인 스트림을 수신하는 방법을 [TsgcWebSocketClient](#)를 사용하여 바이너리 websocket 메시지를 읽는 방법을 보여줍니다.

서버에 연결

- 먼저 [TsgcWebSocketClient](#)의 새 인스턴스를 생성하십시오.
- 그런 다음 서버 **Host** 및 **Port**를 구성하십시오.
- 메시지가 서버에서 전송될 때 더 적은 바이트를 수신하도록 압축을 활성화하십시오.

```
Client.Extensions.PerMessage_Deflate.Enabled = true
```

- 서버가 보낸 이미지는 스트림으로 도착하므로 이미지를 읽으려면 **OnBinary** 이벤트를 사용하십시오.

```
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 File

이 데모는 websocket 프로토콜을 사용하여 웹 브라우저에서 서버로 파일을 업로드하는 방법을 보여 줍니다.

구성

- 먼저 새 [TsgcWebSocketServer](#) 인스턴스를 생성하고, [Server Chat Demo](#)를 확인하십시오.
- 파일은 바이너리 스트림으로 서버에 도착하므로, 파일을 읽으려면 **OnBinary** 이벤트를 처리해야 합니다.

```
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);
}
```

- 파일 이름을 알고 싶다면 파일이 전송되기 전에 파일 이름이 포함된 텍스트 메시지를 보낼 수 있습니다

```
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);
}
```

websockets를 사용하여 파일을 보내는 javascript 코드는 아래와 같습니다:

```
<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 | Server Authentication

이 데모는 Server Authentication을 사용하는 방법을 보여줍니다. 다양한 인증 유형에 대해 더 알고 싶다면 [Authentication](#)에 관한 다음 문서를 읽어 보십시오.

Authentication

- 먼저 [TsgcWebSocketServer](#)의 새 인스턴스를 만듭니다. Authentication 속성을 활성화하십시오.
server.Authentication.Enabled = true;
- 그런 다음 **OnAuthentication** 이벤트 핸들러에서 사용자 이름과 비밀번호가 올바른지 확인하십시오. 올바른 경우 Authenticated 속성을 true로 설정하고, 그렇지 않으면 false로 설정하십시오.

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

Demos | KendoUI_Grid

이 데모는 WebSockets를 프로토콜로 사용하고 웹 브라우저를 클라이언트로 사용하여 KendoUI Grid가 작동하는 방법을 보여줍니다. 기본적으로 클라이언트 중 하나가 변경할 때 업데이트되는 javascript grid이며, 이러한 변경 사항은 websocket 프로토콜을 사용하여 연결된 모든 클라이언트에 업데이트되므로 모든 클라이언트가 클라이언트가 만든 모든 변경 사항을 포함하여 동일한 데이터를 실시간으로 볼 수 있습니다.

구성

- 먼저 새 [TsgcWebSocketHTTPServer](#) 인스턴스를 생성하고, [Server Chat Demo](#)를 확인하십시오.
- 그런 다음 OnCommandGet을 처리하여 웹 브라우저 클라이언트가 요청한 필요한 파일을 보내야 합니다.

```
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;
    }
}
```

WebSockets Updates

클라이언트가 그리드 레코드를 업데이트하면 이 변경 사항이 websocket 프로토콜을 사용하여 연결된 모든 클라이언트에게 전송됩니다. 그리드 변경 사항에 대한 알림을 받으려면 OnMessage 이벤트를 사용하십시오. 메시지는 JSON 형식이므로 JSON 텍스트를 읽고, 디코딩하고, 다른 피어에게 응답을 보내기만 하면 됩니다.

```
void OnServerMessage(TsgcWSConnection *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

이 데모는 WebSocket Server에서 Server-Sent Events가 작동하는 방식을 보여줍니다. sgcWebSockets는 서버가 동일한 수신 포트에서 둘 이상의 프로토콜을 처리할 수 있도록 합니다.

[Server Sent Events](#)에 대해 더 자세히 읽어볼 수 있습니다.

이 데모는 서버가 Server Sent Events를 사용하여 연결된 모든 클라이언트에게 매초 시간을 보내는 방법을 보여줍니다.

서버가 시작되면, 연결된 모든 클라이언트에게 **Server Time**이 포함된 메시지를 **브로드캐스트**하므로, 클라이언트가 이 메시지를 받을 때마다 사용자에게 표시합니다.

```
private void Timer1Timer(TOobject *sender)
{
    server->Broadcast("data: " + "Server Time: " + FormatDateTime("hh:nn:ss", Now));
}
```

websocket 연결을 처리하는 **javascript 코드**는 아래에 표시됩니다:

```
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";
});
```

데모 | Server WebRTC

이 데모는 [TsgcWebsocketHTTPServer](#) 와 javascript 라이브러리로서 WebRTC를 사용하여 화상 회의 서버를 빌드하는 방법을 보여줍니다.

데모는 WebSocket 프로토콜을 사용하여 WebRTC를 신호하고 공개 STUN/TURN 서버를 사용합니다. 프로덕션 사이트의 경우 자체 STUN/TURN 서버를 사용해야 합니다. 등록된 사용자는 이미 컴파일되어 있고 서버에서 실행하는 데 필요한 모든 라이브러리가 포함된 Windows용 Coturn을 다운로드할 수 있습니다.

클라이언트는 WebRTC 연결을 지원하는 웹 브라우저여야 합니다.

구성

- 먼저 새 [TsgcWebsocketHTTPServer](#) 인스턴스를 생성하고, [Server Chat Demo](#)를 확인하십시오.
- 그런 다음, [TsgcWSPServer_WebRTC](#)의 새 인스턴스를 생성하십시오.
- 그 후, WebRTC Protocol을 WebSocket Server에 할당하고 서버 호스트와 포트를 구성해야 합니다.

```
WSServer->Port = StrToInt(txtDefaultPort->Text);
// ... bindings
With WSServer->Bindings->Add do
{
    Port = StrToInt(txtDefaultPort->Text);
    IP = txtHost>Text;
}
// ... active
WSServer->Active = true;
```

이 데모에는 WebRTC 프런트 페이지를 디스패치하는 데 사용되는 인덱스 HTML 페이지가 필요하며, 이 페이지는 데모와 함께 제공됩니다.

WebBrowser에서 실행

서버를 구성한 후, 서버를 시작하고 사용 가능한 웹 브라우저 중 하나를 선택하십시오. 새 세션을 시작하도록 요청하는 새 웹 브라우저 세션이 열립니다. 성공하면 자신의 비디오가 표시되며, 다른 웹 브라우저에서 동일한 url을 열면 두 피어가 연결된 것을 볼 수 있습니다.

데모는 기본적으로 SSL 없이 실행되며, 이는 localhost 연결에만 유효합니다. 프로덕션 사이트의 경우 SSL 연결을 사용하십시오. 서버 측에서 SSL을 구성하려면 [Server Chat Demo](#)를 확인하십시오.

Demos | Server AppRTC

이 데모는 [TsgcWebSocketHTTPServer](#)와 javascript 라이브러리인 AppRTC를 사용하여 Video Conference Server를 구축하는 방법을 보여줍니다.

데모는 WebSocket 프로토콜을 사용하여 WebRTC를 신호하고 공개 STUN/TURN 서버를 사용합니다. 프로젝트 사이트의 경우 자체 STUN/TURN 서버를 사용해야 합니다. 등록된 사용자는 이미 컴파일되어 있고 서버에서 실행하는 데 필요한 모든 라이브러리가 포함된 Windows용 Coturn을 다운로드할 수 있습니다.

클라이언트는 WebRTC 연결을 지원하는 웹 브라우저여야 합니다.

구성

- 먼저 새 [TsgcWebSocketHTTPServer](#) 인스턴스를 생성하고, [Server Chat Demo](#)를 확인하십시오.
- 그런 다음 [TsgcWSPServer_AppRTC](#)의 새 인스턴스를 생성하십시오.
- 그 후 AppRTC Protocol을 WebSocket 서버에 할당하고 서버 호스트와 포트를 구성해야 합니다. WebRTC는 보안 연결이 필요하므로 PEM 인증서를 사용하고 구성 요소의 SSLOptions 속성을 구성해야 합니다.

```
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.RoomLink는 웹 브라우저가 room에 로그인하기 위해 리디렉션될 url입니다
- AppRTC.WebSocketURL은 websocket 연결의 url입니다.
- IceServers는 AppRTC Server 프로토콜에서 구성할 수 있습니다.

데모에는 AppRTC 첫 페이지를 디스패치하는 데 사용되는 인덱스 HTML 페이지가 필요하며, 이 페이지는 데모와 함께 제공됩니다.

WebBrowser에서 실행

서버를 구성한 후, 시작하고 사용 가능한 웹 브라우저 중 하나를 선택하십시오. 새 방에 참여하도록 요청하는 새 웹 브라우저 세션이 열립니다. 이 방에 참여하고 성공하면, 새 화상 회의를 시작하기 위해 다른 웹 브라우저에서 사용해야 하는 링크가 표시됩니다.

AppRTC

Please enter a room name.

959454873

JOIN

RANDOM

Recently used rooms:

Demos | Telegram Client

이 데모는 Telegram에 연결하고, 모든 연락처를 수신하고, 텍스트 메시지를 보내고, 이미지를 보내는 방법... 등 훨씬 더 많은 것을 보여줍니다.

구성

- 먼저 [TsgcTDLib_Telegram](#)의 새 인스턴스를 생성하십시오.
- 그런 다음, telegram에 연결을 시도하기 전에 API Hash, API Id 같은 일부 매개변수를 클라이언트 구성 요소에 전달해야 합니다. 필요한 모든 매개변수를 설정한 후, Active = true 속성을 설정하여 연결을 시작하십시오.

```
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;
```

- 클라이언트가 Telegram에 연결을 시도할 때, 일반적으로 코드가 필요하므로 **OnTelegramAuthenticationCode**를 처리하고 Telegram 계정에서 제공한 값으로 Code 매개변수를 반환해야 합니다.

```
private void OnAuthenticationCodeEvent(TObject *Sender, ref string Code)
{
    Code = InputBox("Telegram", "Introduce Telegram Code");
}
```

Send Telegram Messages

telegram 메시지(텍스트, 파일, 이미지...)를 보내려면 항상 먼저 메시지를 보낼 Chald를 설정한 다음, 텍스트 메시지, 파일 이름... 이 될 수 있는 매개변수를 설정해야 합니다.

```
// send text message
sgcTelegram->SendTextMessage("456413", "Hello From sgcWebSockets!!!");

// send file message
sgcTelegram->SendDocumentMessage("383784", "c:\yourfile.txt");
```

Receive Telegram Messages

Telegram 클라이언트가 수신한 메시지는 특정 이벤트 핸들러에서 처리됩니다. 다음 텍스트 메시지가 수신될 때, 새 문서가 수신될 때, 사진... 에 대한 이벤트가 있습니다.

```
private void OnMessageTextEvent(TObject *Sender, TsgcTelegramMessageText *MessageText)
{
    DoLogMessage(MessageText->ChatId, MessageText->SenderUserId->ToString(), MessageText->Text);
}
```

```
private void OnMessageDocumentEvent(TsgcTDLib_Telegram *Sender, TsgcTelegramMessageDocument *MessageDocument)
{
    DoLogMessage(MessageDocument->ChatId, MessageDocument->SenderUserId->ToString(), MessageDocument->FileName);
}
```

Coturn

Coturn

sgcWebSockets 4.5.2 ENTERPRISE Edition부터 Delphi/CBuilder를 사용하여 자체 STUN/TURN 서버를 빌드할 수 있습니다.

TURN 및 STUN 서버의 무료 오픈소스 구현입니다.

TURN Server는 VoIP 미디어 트래픽 NAT traversal 서버 및 게이트웨이입니다. 범용 네트워크 트래픽 TURN 서버 및 게이트웨이로도 사용할 수 있습니다.

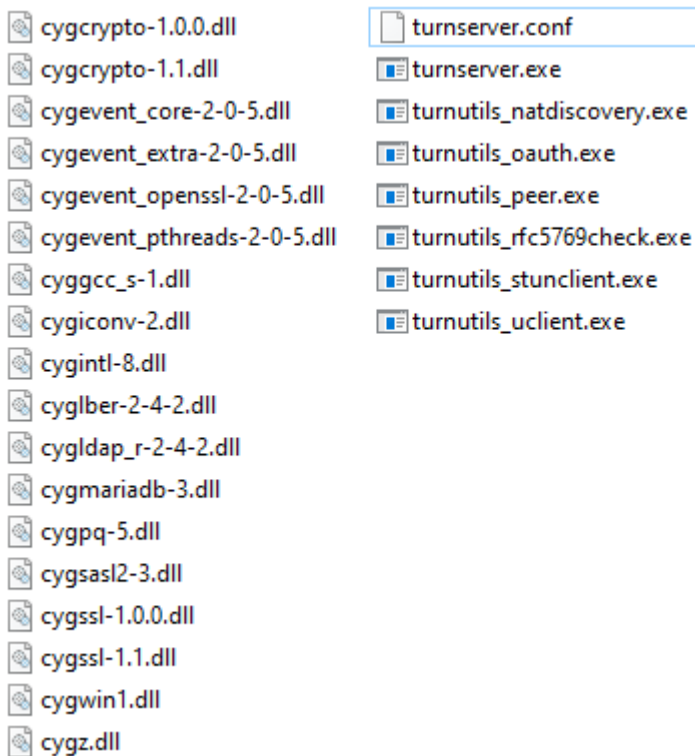
지원되는 프로젝트 대상 플랫폼은 다음과 같습니다:

- **Linux**
- **Mac OS X**
- **Windows (Cygwin):** 등록된 사용자를 위해 컴파일된 바이너리를 사용할 수 있습니다.

Windows 구성

먼저 계정에서 컴파일된 바이너리를 다운로드해야 합니다. 사용 가능한 2개의 버전이 있습니다: win32 및 win64. 원하는 플랫폼을 선택하고 바이너리를 폴더에 압축 해제하십시오. 다음 파일이 생성됩니다:

1. 애플리케이션을 실행하는 데 필요한 일부 cygwin 라이브러리는 coturn 서버와 함께 이러한 라이브러리를 배포해야 합니다.
2. 일부 콘솔 애플리케이션:
 - 2.1 turnserver.exe: TURN/STUN 서버를 실행하기 위한 메인 콘솔 애플리케이션입니다
 - 2.2 기타 애플리케이션: 구성 또는 테스트 목적으로 사용됩니다.
3. Turnserver.conf: coturn 서버의 구성 파일입니다.



turnserver.conf

이것은 coturn 서버의 구성 파일입니다. 열어 보면 기본 구성을 볼 수 있습니다.

Simple Configuration

서버에는 다음 공용 IP 80.15.44.123이 있으며 포트 80에서 수신 대기합니다. 연결을 위한 자격 증명은 다음과 같습니다:
username = demo, password = secret
다음 구성을 설정하십시오:

```
listening-ip=80.15.44.123
listening-port=80
realm=yourrealm.com
user=demo:secret
```

TLS가 활성화된 구성

서버는 다음 공용 IP 80.15.44.123을 가지며 포트 80 및 443(TLS 연결)에서 수신합니다. 연결을 위한 자격 증명은 username = demo, password = secret입니다. 인증서 이름(PEM 형식이어야 함)은 certificate.crt이고 개인 키는 private.key입니다.
다음 구성을 설정하십시오:

```
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
```

더 많은 구성을 사용할 수 있습니다. turnserver.conf를 열고 문서화된 섹션을 읽으십시오.

Run coturn

구성이 완료되면 turnserver.exe를 실행하기만 하면 서버를 실행할 수 있습니다. 새 콘솔 애플리케이션이 열리고 로그 파일이 생성됩니다. turnserver.conf 파일에서 "verbose"를 활성화하면 콘솔 애플리케이션의 상세 수준을 높일 수 있습니다(더 자세한 메시지를 얻을 수 있음).

```
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

eSeGeCe Components 최종 사용자 라이선스 계약

eSeGeCe Components("eSeGeCe") End-User License Agreement ("EULA")는 귀하(개인 또는 단일 법인)와 관련 소프트웨어 구성 요소, 미디어, 인쇄물, "온라인" 또는 전자 문서("eSeGeCe components")를 포함할 수 있는 모든 eSeGeCe 구성 요소에 대한 eSeGeCe의 작성자 사이의 법적 계약입니다. eSeGeCe components를 설치, 복사 또는 기타 방식으로 사용함으로써 귀하는 이 EULA의 조건에 구속되는 데 동의합니다. 이 라이선스 계약은 귀하와 eSeGeCe의 작성자("LICENSER"라고 함) 사이의 프로그램에 관한 전체 계약을 나타내며, 양 당사자 간의 모든 이전 제안, 표현 또는 이해를 대체합니다. 이 EULA의 조건에 동의하지 않는 경우 eSeGeCe components를 설치하거나 사용하지 마십시오.

eSeGeCe 구성 요소는 저작권법 및 국제 저작권 조약뿐만 아니라 기타 지적 재산권법 및 조약에 의해 보호됩니다. eSeGeCe 구성 요소는 판매되는 것이 아니라 라이선스가 부여됩니다.

소스 코드가 필요한 경우 등록 비용을 지불해야 합니다. eSeGeCe의 라이선스 키 및/또는 전체 에디션(DCU 에디션 및 Source 에디션 포함)을 제삼자 개인 및/또는 단체에 제공해서는 안 됩니다. 또한 제삼자 개인 및/또는 단체로부터 eSeGeCe의 라이선스 키 및/또는 전체 에디션을 사용해서도 안 됩니다.

1. 라이선스 부여

eSeGeCe 구성 요소는 다음과 같이 라이선스가 부여됩니다:

(a) 설치 및 사용.

LICENSER는 eSeGeCe 구성 요소가 설계된 운영 체제[예: Windows 2000, Windows 2003, Windows XP, Windows ME, Windows Vista, Windows 7, Windows 8, Windows 10]의 유효하게 라이선스된 복사본을 실행하는 컴퓨터에서 eSeGeCe 구성 요소의 복사본을 설치하고 사용할 권리를 귀하에게 부여합니다.

(b) 로열티 무료.

eSeGeCe 구성 요소를 기반으로 하는 상업용 애플리케이션을 생성하고 실행 파일과 함께 배포할 수 있으며, 로열티가 필요하지 않습니다.

(c) 수정(Source 에디션 전용).

이 라이선스 계약에 명시된 조건에 따라 귀하에게 제공된 라이선스 SOURCE CODE에 대해 수정, 개선, 파생 작업 및/또는 확장을 수행할 수 있습니다.

(d) 백업 사본.

백업 및 보관 목적으로 필요한 경우 eSeGeCe 구성 요소의 복사본을 만들 수도 있습니다.

2. 기타 권리 및 제한 사항에 대한 설명

(a) 저작권 고지 유지.

eSeGeCe 구성 요소의 모든 복사본에서 저작권 고지를 제거하거나 변경해서는 안 됩니다.

(b) 배포.

등록된 eSeGeCe 구성 요소의 복사본을 제3자에게 배포할 수 없습니다. eSeGeCe 공식 웹사이트에서 다운로드할 수 있는 평가판 에디션은 자유롭게 배포할 수 있습니다.

애플리케이션을 위해 eSeGeCe 구성 요소를 포함하는 구성 요소/ActiveX 컨트롤/라이브러리를 생성할 수 있지만 제3자에게 배포하거나 게시해서는 안 됩니다.

(c) SOURCE CODE 배포 금지(Source 에디션만 해당).

SOURCE CODE, 또는 그 수정, 향상, 파생 작업 및/또는 확장을 SOURCE CODE 형태로 제3자에게 배포하거나 게시해서는 안 됩니다.

SOURCE CODE의 어떤 부분도 제삼자에게 배포, 게시, 공개 또는 기타 방식으로 제공해서는 안 됩니다.

(d) 리버스 엔지니어링, 디컴파일 및 디스어셈블 금지.

적용 가능한 법률이 이 제한에도 불구하고 그러한 활동을 명시적으로 허용하는 경우 및 그 범위 내인 경우를 제외하고, eSeGeCe 구성 요소를 리버스 엔지니어링, 디컴파일 또는 디스어셈블할 수 없습니다.

(e) 임대.

eSeGeCe 구성 요소를 임대, 리스 또는 대여할 수 없습니다.

(f) 지원 서비스.

LICENSER는 eSeGeCe 구성 요소와 관련된 지원 서비스("지원 서비스")를 귀하에게 제공할 수 있습니다. 지원 서비스의 일부로 귀하에게 제공되는 모든 추가 소프트웨어 코드는 eSeGeCe 구성 요소의 일부로 간주되며 본 EULA의 조건 및 약관의 적용을 받습니다.

eSeGeCe는 한 번에 한 명의 개발자만 사용하도록 라이선스가 부여됩니다. 그리고 기술 지원은 한 명의 특정 개발자에게만 제공됩니다.

(g) 적용 가능한 법률 준수.

eSeGeCe 구성 요소 사용과 관련된 모든 해당 법률을 준수해야 합니다.

3. 종료

다른 권리를 침해하지 않고, 귀하가 이 EULA의 약관과 조건을 준수하지 않는 경우 LICENSER는 이 EULA를 종료할 수 있습니다. 그러한 경우 귀하는 소유하고 있는 eSeGeCe 구성 요소의 모든 사본을 폐기해야 합니다.

4. COPYRIGHT

저작권을 포함하되 이에 국한되지 않는 eSeGeCe 구성 요소 및 그 사본에 대한 모든 권리는 LICENSER 또는 그 공급업체가 소유합니다. eSeGeCe 구성 요소의 사용을 통해 접근할 수 있는 콘텐츠에 대한 모든 권리 및 지적 재산권은 해당 콘텐츠 소유자의 재산이며 해당 저작권 또는 기타 지적 재산권 법률 및 조약에 의해 보호될 수 있습니다. 이 EULA는 그러한 콘텐츠를 사용할 수 있는 권리를 부여하지 않습니다. 명시적으로 부여되지 않은 모든 권리는 LICENSER가 보유합니다.

5. 보증 없음

LICENSER는 eSeGeCe 구성 요소에 대한 모든 보증을 명시적으로 부인합니다. eSeGeCe 구성 요소는 상품성, 비침해 또는 특정 목적에의 적합성에 대한 보증을 포함하되 이에 국한되지 않는 어떠한 명시적 또는 묵시적 보증도 없이 "있는 그대로" 제공됩니다. LICENSER는 eSeGeCe 구성 요소 내에 포함된 정보, 텍스트, 그래픽, 링크 또는 기타 항목의 정확성이나 완전성에 대해 보증하거나 책임을 지지 않습니다. LICENSER는 컴퓨터 바이러스, 웜, 시한폭탄, 논리 폭탄 또는 기타 그러한 컴퓨터 프로그램의 전송으로 인해 발생할 수 있는 모든 손해에 관해 어떠한 보증도 하지 않습니다. LICENSER는 또한 Authorized Users 또는 제3자에 대한 모든 보증 또는 표현을 명시적으로 부인합니다.

6. 책임의 제한

LICENSER는 어떠한 경우에도 "Authorized Users"의 eSeGeCe 구성 요소 사용 또는 사용 불능으로 인해 발생하는 어떠한 손해(이익 손실, 사업 중단 또는 정보 손실을 포함하되 이에 국한되지 않음)에 대해서도, LICENSER가 그러한 손해의 가능성을 사전에 통보받았더라도 책임을 지지 않습니다. 어떠한 경우에도 LICENSER는 데이터 손실 또는 계약, 불법 행위 또는 기타에 근거한 간접적, 특별한, 부수적, 결과적(이익 손실 포함) 손해 또는 기타 손해에 대해 책임을 지지 않습니다. LICENSER는 eSeGeCe 구성 요소 또는 그 일부의 내용에 관하여, 그 안에 포함된 오류 또는 누락, 명예훼손, 퍼블리시티권 침해, 사생활 침해, 상표권 침해, 사업 중단, 인적 피해, 사생활 손실, 인격권 또는 기밀 정보의 공개를 포함하되 이에 국한되지 않는 어떠한 책임도 지지 않습니다.

색인

- AbstractApi Geolocation [1075](#)
- Add Telegram Proxy [642](#)
- ALPN [126](#)
- Amazon SQS [924](#)
- AMQP Channels [310](#)
- AMQP Consume Messages [318](#)
- AMQP Exchanges [312](#)
- AMQP Get Messages [320](#)
- AMQP Publish Messages [317](#)
- AMQP QoS [321](#)
- AMQP Queues [314](#)
- AMQP Transactions [322](#)
- AMQP1 Links [346](#)
- AMQP1 Read Message [353](#)
- AMQP1 Receiver Links [349](#)
- AMQP1 Send Message [351](#)
- AMQP1 Sender Links [347](#)
- AMQP1 Sessions [344](#)
- Android 애플리케이션 빌드 [74](#)
- Anthropic [729](#)
- Anthropic Batches [741](#)
- Anthropic Citations [748](#)
- Anthropic Documents [744](#)
- Anthropic Extended Thinking [742](#)
- Anthropic Files [753](#)
- Anthropic MCP Connector [755](#)
- Anthropic Messages [734](#)
- Anthropic Models [740](#)
- Anthropic Prompt Caching [746](#)
- Anthropic Structured Outputs [751](#)
- Anthropic Tool Use [738](#)
- Anthropic Vision [736](#)
- Anthropic Web Search [749](#)
- API [436](#), [496](#), [512](#), [559](#), [566](#), [587](#)
- API 3Commas [542](#)
- API Binance [414](#), [429](#), [435](#)
- API Binance Futures [429](#), [435](#)
- API Binance Futures Trade [435](#)
- API Bitfinex [512](#)
- API Bitget [594](#)
- API Bitmex [504](#)
- API Blockchain [569](#)
- API Bybit [555](#)
- API Cex [571](#)
- API Cex Plus [578](#)
- API Coinbase [438](#)
- API Coinbase Pro [438](#)
- API Crypto.com [601](#)
- API Deribit [598](#)
- API Discord [582](#)
- API Forex [559](#)
- API GatelO [596](#)
- API HTX [603](#)
- API Kraken [458](#), [460](#), [466](#), [469](#), [471](#), [474](#), [476](#), [483](#), [489](#), [491](#)
- API Kraken Futures [474](#), [476](#), [483](#), [489](#), [491](#)
- API Kraken Futures REST Private [491](#)
- API Kraken Futures REST Public [489](#)
- API Kraken REST Private [471](#)
- API Kraken REST Public [469](#)
- API Kucoin [516](#)
- API Kucoin Futures [529](#)
- API MEXC [587](#)
- API MEXC Futures [591](#)
- API OKX [546](#)
- API OpenAI [585](#)
- API Pusher [496](#)
- API SignalR [455](#)
- API SignalRCore [449](#)
- API SocketIO [436](#)
- API Telegram [627](#)
- API Whatsapp [605](#)
- API XTB [552](#)
- APIs [412](#), [414](#), [421](#), [424](#), [429](#), [435](#), [438](#), [442](#), [445](#), [449](#), [455](#), [458](#), [460](#), [466](#), [469](#), [471](#), [474](#), [476](#), [483](#), [489](#), [491](#), [504](#), [542](#), [569](#), [571](#), [582](#), [591](#), [627](#), [1074](#)
- APNs [827](#), [828](#), [829](#)
 - Certificate-Based Connection [829](#)
 - Token-Based Connection [828](#)
- APNs 원격 알림 [826](#)
- 생성 [826](#)

- Apple Push Notifications [825](#)
- Authentication [107](#), [163](#), [188](#)
- Authorization Code Grant [871](#)
- Authorization Code Grant (RFC 6749) [844](#)
- Authorization Code with PKCE (RFC 7636) [846](#)
- Auto Ban [236](#)
- Binance Connect [421](#)
- Binance Get Market Data [423](#)
- Binance Private Requests Time [427](#)
- Binance Private REST API [424](#)
- Binance Subscribe [422](#)
- Binance Trade Spot [425](#)
- Binance Withdraw [428](#)
- Bindings [116](#), [175](#)
- Bitmex Connect WebSocket API [508](#)
- Bitmex Subscribe WebSocket Channel [509](#)
- Bot [638](#)
- Bot Detection [244](#)
- Broadcast [115](#)
- Brute Force [236](#)
- Build [71](#), [72](#), [75](#)
- Build iOS Application [75](#)
- Build OSX Application [72](#)
- Certificate-Based Connection [829](#)
 - APNs [829](#)
- Certificates OpenSSL [155](#)
- Certificates SChannel [156](#)
- Channels [115](#), [310](#), [422](#), [443](#), [567](#), [992](#)
- CIDR [234](#)
- Client [146](#), [148](#), [149](#), [158](#), [159](#), [163](#), [167](#), [168](#), [169](#), [194](#), [293](#), [295](#), [296](#), [308](#), [309](#), [326](#), [327](#), [399](#), [816](#), [817](#), [819](#), [820](#), [980](#), [981](#), [982](#), [989](#), [990](#), [991](#), [992](#), [1078](#), [1082](#), [1087](#), [1097](#)
- Client AMQP Connect [308](#)
- Client AMQP Disconnect [309](#)
- Client AMQP1 Authentication [341](#)
- Client AMQP1 Azure MessageBus [342](#)
- Client AMQP1 Disconnect [338](#)
- Client AMQP1 Idle Timeout Connection [339](#)
- Client Authentication [163](#), [820](#)
- Client Chat [1078](#)
- Client Close Connection [148](#), [816](#)
- Client Credentials Grant [873](#)
- Client Credentials Grant (RFC 6749) [848](#)
- Client Kafka Connect [326](#)
- Client Kafka Disconnect [327](#)
- Client Keep Connection Active [817](#)
- Client Keep Connection Open [149](#)
- Client MQTT Connect [293](#)
- Client MQTT Sessions [295](#)
- Client MQTT Version [296](#)
- Client Open Connection [146](#)
- Client Pending Requests [819](#)
- Client Proxies [169](#)
- Client Register Protocol [168](#)
- Client Send Binary Message [159](#)
- Client Send Text [158](#)
- Client Send Text Message [158](#)
- Client Snapshots [1087](#)
- Client SocketIO [1082](#)
- Client WebSocket HandShake [167](#)
- Clients [295](#), [308](#), [326](#), [327](#), [399](#), [817](#), [819](#), [1078](#)
 - Send Files [399](#)
- Coinbase Pro Connect [442](#)
- Coinbase Pro Get Market Data [444](#)
- Coinbase Pro Place Orders [447](#)
- Coinbase Pro Private Requests Time [446](#)
- Coinbase Pro Private REST API [445](#)
- Coinbase Pro SandBox Account [448](#)
- Coinbase Pro Subscribe [443](#)
- CommitOffset [332](#)
- CommitSync [330](#)
- Configure Install [54](#)
- Configure ZLib [62](#)
- Connect Mosquitto [294](#)
- Connect Secure Server [154](#)
- Connect TCP Server [151](#)
- Connect WebSocket Server [145](#)
- Connections TIME_WAIT [152](#)
- Consume [329](#)
- Consumer [329](#)
- Consumer Group [330](#)
- Coturn [1099](#)
- Crawler Detection [244](#)
- CreateTopic [331](#)
- Cross-Site Scripting [238](#)
- CryptoHopper [645](#)
- Custom Objects [121](#)
- Custom Sub [106](#)
- Datacenter Detection [244](#)

- Datasnap [1009](#)
- DeepSeek [760](#)
- DeepSeek Messages [762](#)
- DeepSeek Models [764](#)
- DeepSeek Vision [763](#)
- Deflate-Frame [657](#)
- DeleteTopic [331](#)
- Device Authorization Grant (RFC 8628) [852](#), [879](#)
- DLL 사용 [103](#)
- DNSBL [244](#)
- DPoP - Demonstrating Proof of Possession (RFC 9449) [854](#)
- DPoP Validation (RFC 9449) [885](#)
- Embeddings ChatBot [725](#)
- Embeddings Create Vectors [724](#)
- EPOLL [125](#)
- Error [205](#)
- Extensions [655](#)
- Fast Performance Server [77](#)
- FCrDNS [244](#)
- Files [117](#), [140](#), [198](#), [398](#), [399](#), [400](#), [813](#), [1088](#)
- Fired [225](#)
- Firewall Blacklist [234](#)
- Firewall Whitelist [234](#)
- Flash [120](#)
- Flood Protection [241](#)
- Flow [69](#)
 - Threading [69](#)
- Forex Account [564](#)
- Forex Connect [561](#), [566](#)
- Forex Subscribe [567](#)
- Forex Trade [562](#)
- Forex 시장 데이터 [565](#)
- Forward HTTP Requests [127](#)
- Found [638](#)
- Fragmented Messages [142](#)
- Gemini [757](#)
- Gemini CountTokens [772](#)
- Gemini EmbedContent [773](#)
- Gemini Embeddings [773](#)
- Gemini Function Calling [774](#)
- Gemini Messages [767](#)
- Gemini Models [770](#)
- Gemini Structured Outputs [771](#)
- Gemini Token Counting [772](#)
- Gemini Tool Use [774](#)
- Gemini Vision [769](#)
- GetCommittedOffset [332](#)
- GetEarliestOffset [332](#)
- GetLatestOffset [332](#)
- GetMetadata [331](#)
- Google Calendar [947](#), [953](#), [954](#), [955](#)
- Google Calendar RefreshToken [955](#)
- Google Calendar Service Account [956](#)
- Google Calendar Sync Calendars [953](#)
- Google Calendar Sync Events [954](#)
- Google Cloud FCM [957](#)
- Google Cloud Pub [939](#)
- Google OAuth2 Keys [929](#)
- Google Service Accounts [935](#)
- Grok [782](#)
- Grok Messages [784](#)
- Grok Models [786](#)
- Grok Vision [785](#)
- GroupId [330](#)
- Groups [122](#)
- HeartBeat [111](#)
- HighPerfAcceptsPerWorker [216](#)
- History [564](#)
- How to Place a Bitmex Order [510](#)
- HTTP [67](#), [114](#), [127](#), [197](#), [198](#), [205](#), [206](#), [220](#), [807](#), [811](#)
- HTTP Dispatch Files [198](#)
- HTTP Server Stream Video [208](#)
- HTTP/2 [199](#), [200](#), [202](#), [812](#), [813](#), [814](#), [815](#), [818](#), [821](#)
- HTTP/2 Alternate Service [202](#)
- HTTP/2 Download File [813](#)
- HTTP/2 Headers [815](#)
- HTTP/2 Partial Responses [814](#)
- HTTP/2 Reason Disconnection [818](#)
- HTTP/2 Server Push [200](#), [812](#)
- HTTP/2 Server Threads [203](#)
- HTTP1 [831](#)
- HTTP2 [808](#)
- HTTPAPI [218](#), [220](#), [221](#), [225](#)
- HTTPAPI Custom Headers [221](#)
- HTTPAPI Disable HTTP [220](#)
- HTTPAPI FineTune [216](#)
- HTTPAPI OnDisconnect [225](#)
- HTTPAPI Send File Response [223](#)

- HTTPAPI Send Text Response [222](#)
- HTTPAPI URL Reservation [214](#)
- ICE [996](#)
- ICE Gather Candidates [998](#)
- ICE Pair Candidates [999](#)
- In HTML [865](#)
- Indy [99](#)
- Indy SChannel [180](#), [209](#)
- Install Errors [50](#)
- Install Setup [32](#)
- IOCP [124](#)
- IoT [794](#), [795](#), [802](#)
- IoT Amazon MQTT Client [795](#)
- IoT Azure MQTT Client [802](#)
- IP Ban [236](#)
- IP Filtering [234](#)
- JWT [894](#)
- Kafka [326](#), [327](#), [328](#), [329](#), [330](#), [331](#), [332](#)
- Kafka Consume Messages [329](#)
- Kafka Consumer Groups [330](#)
- Kafka Offsets [332](#)
- Kafka Produce Messages [328](#)
- Kafka Topics [331](#)
- KendoUI_Grid [1091](#)
- Kucoin Connect WebSocket API [522](#)
- Kucoin Futures Connect WebSocket API [534](#)
- Kucoin Futures Get Market Data [536](#)
- Kucoin Futures Private Requests Time [541](#)
- Kucoin Futures Private REST API [537](#)
- Kucoin Futures Subscribe WebSocket Channel [535](#)
- Kucoin Futures Trade [538](#)
- Kucoin Get Market Data [524](#)
- Kucoin Private Requests Time [528](#)
- Kucoin Private REST API [525](#)
- Kucoin Subscribe WebSocket Channel [523](#)
- Kucoin Trade Spot [526](#)
- License [1101](#)
- Lightstreamer [566](#), [567](#)
- LoadBalancing [139](#)
- Logs [113](#)
- MaxConnectionsPerIP [241](#)
- MaxMessagesPerSec [241](#)
- MCP [658](#)
- MCP Client Elicitation [683](#)
- MCP Client Prompts [677](#)
- MCP Client Roots [681](#)
- MCP Client Sampling [682](#)
- MCP Client Tools [675](#)
- MCP Server Elicitation [673](#)
- MCP Server Prompts [665](#)
- MCP Server Resources [668](#)
- MCP Server Roots [671](#)
- MCP Server Sampling [672](#)
- MCP Server Sessions [661](#)
- MCP Server Tools [662](#)
- MCP 클라이언트 리소스 [679](#)
- Memory Manager [80](#)
- Message Filtering [238](#)
- Method [811](#)
- Mistral AI [787](#)
- Mistral Embeddings [793](#)
- Mistral Messages [790](#)
- Mistral Models [792](#)
- Mistral Vision [791](#)
- MQTT [293](#), [294](#), [295](#), [296](#), [297](#), [298](#), [299](#), [300](#), [301](#), [302](#), [303](#), [795](#), [802](#), [1080](#)
- MQTT Clear Retained Messages [303](#)
- MQTT Publish [297](#), [300](#), [302](#)
- MQTT Publish Message [300](#)
- MQTT Publish Subscribe [297](#)
- MQTT Subscribe [299](#)
- MQTT Topics [298](#)
- MQTT 메시지 수신 [301](#)
- Notification Requests [827](#)
- Sending [827](#)
- OAuth2 [821](#), [835](#), [861](#), [865](#), [866](#), [867](#), [868](#), [869](#), [870](#), [929](#)
- OAuth2 Client for Desktop Applications [841](#)
- OAuth2 Client for Web Applications [840](#)
- OAuth2 Customize Sign [865](#)
- OAuth2 None Authenticate URLs [870](#)
- OAuth2 Provider Authentication [891](#)
- OAuth2 Provider Azure AD [889](#)
- OAuth2 Provider Private Endpoints [890](#)
- OAuth2 Provider Requests [893](#)
- OAuth2 Recover Access Tokens [868](#)
- OAuth2 Register Apps [867](#)
- Ollama [776](#)
- Ollama Embeddings [781](#)
- Ollama Messages [779](#)
- Ollama Models [780](#)

- ompClassic [216](#)
- ompHighPerf [216](#)
- OpenAI [684](#)
- OpenAI Applications [703](#)
- OpenAI Audio [695](#), [712](#)
- OpenAI Batch [701](#)
- OpenAI Chat [693](#)
- OpenAI Completion [692](#)
- OpenAI Edit [694](#)
- OpenAI Fine-Tuning [700](#)
- OpenAI Moderation [696](#)
- OpenAI RealTime [697](#)
- OpenAI Responses [698](#)
- OpenAI Speech [699](#)
- OpenAI Uploads [702](#)
- OpenAPI [1014](#)
- OpenAPI Additional Properties [1021](#)
- OpenAPI Amazon AWS [1035](#)
- OpenAPI Amazon AWS Credentials [1041](#)
- OpenAPI Amazon AWS S3 [1043](#)
- OpenAPI API [1027](#)
- OpenAPI Client [1028](#)
- OpenAPI Command Line [1023](#)
- OpenAPI Google Cloud [1044](#)
- OpenAPI Google Cloud Calendar [1057](#)
- OpenAPI Google Cloud OAuth2 [1049](#)
- OpenAPI Google Cloud PubSub [1056](#)
- OpenAPI Google Cloud Service Accounts [1052](#)
- OpenAPI Google Drive [1058](#)
- OpenAPI Library [1025](#)
- OpenAPI Microsoft [1060](#)
- OpenAPI Microsoft Graph [1073](#)
- OpenAPI Microsoft OAuth2 Code [1069](#)
- OpenAPI Microsoft OAuth2 자격 증명 [1071](#)
- OpenAPI Microsoft Register Application [1066](#)
- OpenAPI Microsoft Tenant [1065](#)
- OpenAPI Parser Pascal [1016](#)
- OpenAPI Server Code-First [1034](#)
- OpenAPI Server Spec-First [1033](#)
- OpenAPI 서버 [1031](#)
- OpenSSL [83](#), [86](#), [88](#), [90](#), [91](#), [155](#)
- OpenSSL Android [90](#)
- OpenSSL iOS [91](#)
- OpenSSL Load Additional Functions [97](#)
- OpenSSL OSX [88](#)
- OpenSSL Own CA Certificates [93](#)
- OpenSSL P12 Certificates [95](#)
- OpenSSL Verify Certificate [96](#)
- OpenSSL Windows [86](#)
- OperatingMode [216](#)
- Orders [562](#), [564](#)
- PDF-Back-Cover [1112](#)
- PDF-Front-Cover [1](#)
- PerMessage-Deflate [656](#)
- Pinecone [726](#)
- Poll [329](#)
- Positions [564](#)
- Post Big Files [117](#)
- Produce [328](#)
- Producer [328](#)
- Protocol AMQP [304](#)
- Protocol AMQP1 [333](#)
- Protocol AppRTC [358](#)
- Protocol Dataset [385](#), [389](#), [392](#), [393](#)
- Protocol Dataset Javascript [389](#)
- Protocol Dataset Notify Updates [393](#)
- Protocol Dataset Replicate Table [392](#)
- Protocol Default [377](#), [381](#)
- Protocol Default Javascript [381](#)
- Protocol E2EE [408](#)
- Protocol Files [394](#)
- Protocol Kafka [323](#)
- Protocol MQTT [290](#)
- Protocol Presence [401](#), [405](#)
- Protocol Presence Javascript [405](#)
- Protocol STOMP [354](#)
- Protocol WAMP [363](#), [366](#)
- Protocol WAMP Javascript [366](#)
- Protocol WAMP2 [374](#)
- Protocol WebRTC [360](#)
- Protocols [106](#), [168](#), [287](#), [290](#), [304](#), [323](#), [354](#), [358](#), [360](#), [363](#), [366](#), [374](#), [377](#), [381](#), [385](#), [389](#), [392](#), [393](#), [394](#), [401](#), [405](#)
- Protocols Javascript [287](#)
- Proxy [141](#), [169](#), [642](#)
- Quality [128](#)
- Service [128](#)
- Quality Of Service [128](#)
- QueueLength [216](#)
- Queues [130](#), [314](#)

- QuickStart HTTP [67](#)
- QuickStart WebSockets [65](#)
- Rate Limiting [241](#)
- RCON [644](#)
- Receive Binary Messages [162](#)
- Receive Text Messages [161](#), [192](#)
- Refresh Token Grant [877](#)
- Register [643](#)
- Register Telegram User [643](#)
- Request HTTP [811](#)
- Resource Owner Password Credentials Grant [875](#)
- Resource Owner Password Credentials Grant (RFC 6749) [850](#)
- Response Body [205](#)
- REST API [561](#)
- Reverse DNS [244](#)
- RTCMultiConnection [650](#)
- RTCPeerConnection Data [1008](#)
- RTCPeerConnection DTLS [1007](#)
- RTCPeerConnection ICE [1006](#)
- RTCPeerConnection Signaling [1005](#)
- RTCPeerConnection STUN TURN [1004](#)
- RTCPeerConnection WebSocket Client [1003](#)
- RTCPeerConnection WebSocket Server [1002](#)
- SChannel [180](#), [209](#)
- SChannel Get Connection Info [157](#)
- SChannel Server [180](#), [209](#)
- Secure Connections [109](#)
- Send Big Files [400](#)
- Send Files [398](#), [399](#)
 - Clients [399](#)
 - Server [398](#)
- Send Files To Clients [399](#)
- Send Files To Server [398](#)
- Send Telegram Invoice Message [640](#)
- Send Telegram Message Bold [637](#)
- Send Telegram Message With Buttons [635](#), [636](#)
- Send Telegram Message With Inline Buttons [635](#)
- Sending [827](#)
 - Notification Requests [827](#)
- Server [174](#), [175](#), [176](#), [177](#), [178](#), [184](#), [185](#), [186](#), [188](#), [190](#), [191](#), [192](#), [194](#), [199](#), [206](#), [218](#), [294](#), [398](#), [812](#), [861](#), [866](#), [869](#), [984](#), [985](#), [994](#), [995](#), [1076](#), [1083](#), [1086](#), [1094](#), [1095](#)
 - Send Files [398](#)
- Server AppRTC [1095](#)
- Server Authentication [188](#), [869](#)
- Server Bindings [175](#)
- Server Chat [1076](#)
- Server Close Connection [186](#)
- Server Endpoints [866](#)
- Server Example [861](#)
- Server Keep Active [177](#)
- Server Keep Connections Alive [184](#)
- Server Monitor [1083](#)
- Server Plain TCP [185](#)
- Server Read Headers [194](#)
- Server Sessions [206](#)
- Server Snapshots [1086](#)
- Server SSL [178](#), [218](#)
- Server SSL SChannel [180](#), [209](#)
- Server Start [174](#)
- Server Startup Shutdown [176](#)
- Server Verify Certificate [183](#)
- Server-Sent Events [137](#)
- ServerSentEvents [1093](#)
- Service [128](#), [202](#), [935](#)
 - Quality [128](#)
- sgcIndy 패키지 설치 [56](#)
- SkipIOCPOnSuccess [216](#)
- SQL Injection [238](#)
- Stop Limit [562](#)
- STUN [977](#), [980](#), [981](#), [982](#), [984](#), [985](#)
- STUN Client Attributes [982](#)
- STUN Client Long Term Credentials [981](#)
- STUN Client UDP Retransmissions [980](#)
- STUN Server Alternate Server [985](#)
- STUN Server Long Term Credentials [984](#)
- Sub [939](#)
- SubProtocol [135](#)
- Subscribe [329](#)
- TCP Connections [134](#)
- Telegram Chat [638](#)
- Telegram Client [1097](#)
- Telegram Get SuperGroup Members [641](#)
- Telegram Sponsored Messages [639](#)
- Threading [69](#)
 - Flow [69](#)

- Throttle [136](#)
- Token Introspection (RFC 7662) [883](#)
- Token Revocation (RFC 7009) [881](#)
- Token-Based Connection [828](#)
 - APNs [828](#)
- Transactions [132](#)
- TsgcAIChat [718](#)
- TsgcAIDatabaseVectorFile [722](#)
- TsgcAIDatabaseVectorPinecone [723](#)
- TsgcAIOpenAIAssistant [704](#)
- TsgcAIOpenAIAssistant File Search [706](#)
- TsgcAIOpenAIAssistant Function Calling [710](#)
- TsgcAIOpenAIAssistant Streaming [708](#)
- TsgcAIOpenAIChatBot [719](#)
- TsgcAIOpenAIEmbeddings [721](#)
- TsgcAIOpenAITranslator [720](#)
- TsgcAudioPlayerMCI [714](#)
- TsgcAudioRecorderMCI [713](#)
- TsgcBotClassification [244](#)
- TsgcHTTP_API_Anthropic [733](#)
- TsgcHTTP_API_DeepSeek [761](#)
- TsgcHTTP_API_Gemini [759](#)
- TsgcHTTP_API_Grok [783](#)
- TsgcHTTP_API_Mistral [789](#)
- TsgcHTTP_API_Ollama [778](#)
- TsgcHTTP_API_OpenAI [691](#)
- TsgcHTTP_JWT_Client [896](#)
- TsgcHTTP_JWT_Server [899](#)
- TsgcHTTP_OAuth2_Client [836](#)
- TsgcHTTP_OAuth2_Client_Google [842](#)
- TsgcHTTP_OAuth2_Client_Microsoft [843](#)
- TsgcHTTP_OAuth2_Server [858](#)
- TsgcHTTP_OAuth2_Server_Provider [887](#)
- TsgcHTTP2Client [809](#)
- TsgcHTTP2ConnectionClient [822](#)
- TsgcHTTP2RequestProperty [823](#)
- TsgcHTTP2ResponseProperty [824](#)
- TsgcICEClient [997](#)
- TsgcIWebsocketClient [281](#)
- TsgcIWWSPClient_Dataset [388](#)
- TsgcIWWSPClient_sgc [380](#)
- TsgcRTCPeerConnection [1000](#)
- TsgcServerHTTPAPI_FineTune [216](#)
- TsgcSTUNClient [978](#)
- TsgcSTUNServer [983](#)
- TsgcTCPClient [170](#)
- TsgcTextToSpeechAmazon [717](#)
- TsgcTextToSpeechGoogle [716](#)
- TsgcTextToSpeechSystem [715](#)
- TsgcTURNClient [987](#)
- TsgcTURNServer [993](#)
- TsgcUDPClient [975](#)
- TsgcUDPServer [976](#)
- TsgcWebPush_Client [654](#)
- TsgcWebSocketClient [143](#)
- TsgcWebSocketClient_WinHTTP [226](#)
- TsgcWebSocketFirewall [227](#), [234](#), [236](#), [238](#), [241](#), [244](#)
- TsgcWebSocketHTTPServer [195](#)
- TsgcWebSocketHTTPServer_Sessions [206](#)
- TsgcWebSocketLoadBalancerServer [279](#)
- TsgcWebSocketProxyServer [280](#)
- TsgcWebSocketServer [172](#)
- TsgcWebSocketServer_HTTPAPI [212](#)
- TsgcWebView2 [960](#)
- TsgcWSAPIClient_MCP [674](#)
- TsgcWSAPIKeyManager [268](#)
- TsgcWSAPIServer_MCP [659](#)
- TsgcWSAPIServer_WebAuthn [901](#)
- TsgcWSAPIServer_WebPush [653](#)
- TsgcWSCircuitBreaker [258](#)
- TsgcWSConnection [283](#)
- TsgcWSHTTP2WebBrokerBridgeServer [1012](#)
- TsgcWSHTTPWebBrokerBridgeServer [1010](#)
- TsgcWSMessageFile [397](#)
- TsgcWSPClient_AMQP [305](#)
- TsgcWSPClient_AMQP1 [334](#)
- TsgcWSPClient_Dataset [387](#)
- TsgcWSPClient_E2EE [411](#)
- TsgcWSPClient_Files [396](#)
- TsgcWSPClient_Kafka [324](#)
- TsgcWSPClient_MQTT [292](#)
- TsgcWSPClient_Presence [404](#)
- TsgcWSPClient_sgc [379](#)
- TsgcWSPClient_STOMP [355](#)
- TsgcWSPClient_STOMP_ActiveMQ [357](#)
- TsgcWSPClient_STOMP_RabbitMQ [356](#)
- TsgcWSPClient_WAMP [365](#)
- TsgcWSPClient_WAMP2 [375](#)
- TsgcWSPPresenceMessage [403](#)
- TsgcWSPServer_AppRTC [359](#)
- TsgcWSPServer_Dataset [386](#)

- TsgcWSPServer_E2EE [410](#)
- TsgcWSPServer_Files [395](#)
- TsgcWSPServer_Presence [402](#)
- TsgcWSPServer_sgc [378](#)
- TsgcWSPServer_WAMP [364](#)
- TsgcWSPServer_WebRTC [361](#)
- TsgcWSRateLimiter [247](#)
- TsgcWSServer_HTTPAPI_WebBrokerBridge [1013](#)
- TURN [986](#), [989](#), [990](#), [991](#), [992](#), [994](#), [995](#)
- TURN Client Allocate IP Address [989](#)
- TURN Client Create Permissions [990](#)
- TURN Client Send Indication [991](#)
- TURN Server Allocations [995](#)
- TURN Server Long Term Credentials [994](#)
- WAMP [369](#), [370](#), [371](#), [372](#)
- WAMP Publishers [370](#)
- WAMP RPC Progress Results [372](#)
- WAMP Simple RPC [371](#)
- WAMP Subscribers [369](#)
- WatchDog [112](#)
- Web Browser Test [105](#)
- WebAuthn [900](#)
- WebAuthn Authentication [911](#)
- WebAuthn Authentication Request [913](#)
- WebAuthn Authentication Response [914](#)
- WebAuthn Authorization [917](#)
- WebAuthn Authorization HTTP [918](#)
- WebAuthn Authorization WebSocket [919](#)
- Webauthn Javascript Client [920](#)
- WebAuthn MDS [916](#)
- WebAuthn Registration [904](#)
- WebAuthn Registration Request [907](#)
- WebAuthn Registration Response [908](#)
- WebAuthn 등록 결과 [910](#)
- WebAuthn 인증 결과 [915](#)
- WebPush [652](#)
- WebRTC [1094](#)
- WebSocket Events [101](#)
- WebSocket Parameters Connection [102](#)
- WebSocket Redirections [153](#)
- WebSockets [65](#), [101](#), [102](#), [145](#), [153](#), [167](#), [421](#), [422](#), [442](#), [443](#), [460](#), [466](#), [476](#), [483](#), [566](#), [567](#)
- WebSockets Private [466](#), [483](#)
- WebSockets Public [460](#), [476](#)
- WebView2 Cookies [966](#)
- WebView2 Downloads [968](#)
- WebView2 JavaScript [964](#)
- WebView2 Navigation [962](#)
- WebView2 Settings [970](#)
- WebView2 고급 기능 [972](#)
- WhatsApp Create App [608](#)
- WhatsApp Download Media [626](#)
- WhatsApp Phone Number Id [609](#)
- WhatsApp Receive Messages and Status Notifications [622](#)
- WhatsApp Security [612](#)
- WhatsApp Send Files [624](#)
- WhatsApp Send Messages [613](#)
- WhatsApp Send Template Messages [620](#)
- WhatsApp Token [610](#)
- WhatsApp Webhook [611](#)
- WhatsApp 대화형 메시지 보내기 [616](#)
- Windows Service [104](#)
- XSS [238](#)
- 가격 바 [565](#)
- 가격 톱 [565](#)
- 개요 [63](#)
- 끊긴 연결 [150](#)
- 바이너리 메시지 [160](#)
- 바이너리 메시지 수신 [193](#)
- 빌드 [74](#)
- 생성 [826](#)
- APNs 원격 알림 [826](#)
- 서버 [193](#), [197](#), [1090](#)
- 서버 바이너리 메시지 전송 [191](#)
- 서버 요청 [197](#)
- 서버 인증 [1090](#)
- 서버 텍스트 메시지 전송 [190](#)
- 설치 [29](#)
- 소개 [23](#)
- 압축 [119](#)
- 에디션 [27](#)
- 응답 대기 [302](#)
- 자체 서명 인증서 [219](#)
- 클라이언트 [160](#), [166](#), [1079](#), [1080](#), [1080](#)
- 클라이언트 AMQP1 연결 [337](#)
- 클라이언트 AMQP1 연결 상태 [340](#)
- 클라이언트 연결 [187](#)
- 클라이언트 예외 [166](#)
- 클라이언트 텍스트 전송 [160](#)
- 파일 업로드 [1088](#)

패키지 설치 [43](#)
프로토콜 [285](#), [362](#)

프로토콜 WebRTC [362](#)
프로토콜 WebRTC Javascript [362](#)

